Aplikace steganografie na platformě Android

Jan Baslar

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan Baslar**
Osobní číslo:	**A20002**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Aplikace steganografie na platformě Android**
Téma práce anglicky:	**Steganography Application on Android platform**

Zásady pro vypracování

1. Nastudujte a popište problematiku steganografie.
2. Popište a porovnejte stávající řešení aplikace steganografie na platformě Android.
3. Navrhněte steganografickou mobilní aplikaci pro platformu Android.
4. Zvolte vhodné technologie a nástroje pro implementaci navržené aplikace.
5. Implementuje aplikaci se zvolenou steganografickou metodou ukrytí dat do obrázku.
6. Implementované řešení vhodně otestujte.

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

Seznam doporučené literatury:

1. WU, Min a Bede LIU. Multimedia data hiding. New York: Springer, c2003, xvii, 218 s. ISBN 9780387954264.
2. SINGH, Simon. Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v českém jazyce. Praha: Dokořán, 2009, 382 s. Aliter. ISBN 9788073632687
3. KNUDSEN, Lars a Matthew ROBSHAW. The block cipher companion. Berlin: Springer, c2011, xiv, 267 s. Information security and cryptography. Dostupné z: doi:9783642173424
4. CID, Carlos, Sean MURPHY a Matthew ROBSHAW. Algebraic aspects of the Advanced Encryption Standard. New York: Springer, c2006, vi, 146 s. ISBN 9780387243634. Dostupné také z: https://proxy.k.utb.cz/login?url=https://doi.org/10.1007/978-0-387-36842-9
5. LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015, 472 s. ISBN 9788025143476.
6. PETITCOLAS, Fabien A. P. a Stefan KATZENBEISSER, ed. Information hiding techniques for steganography and digital watermarking: Stefan Katzenbeisser, Fabien A.P. Petitcolas, editors. Boston: Artech House, 2000, xviii, 220 s. Artech House computer security series. ISBN 1580530354.
7. WAYNER, Peter. Disappearing cryptography: information hiding : steganography & watermarking. 2nd ed. Amsterdam: MK/Morgan Kaufmann Publishers, c2002, xvii, 413 s. ISBN 1558607692. Dostupné také z: http://www.loc.gov/catdir/toc/els031/2001099790.html

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou 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 – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Jan Baslar, v. r.
podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá problematikou steganografie a vývojem mobilní aplikace pro platformu Android, která implementuje obrazovou steganografii. V teoretické části je popsána problematika steganografie obrazu a dále je zde přehled a testování současných steganografických aplikací na platformě Android. Praktická část se zabývá návrhem aplikace včetně originální steganografického algoritmu a jejím vývojem ve frameworku Flutter a následným testováním.

Klíčová slova:

Steganografie, Obrazová steganografie, Mobilní aplikace, Android, Flutter

ABSTRACT

This bachelor's thesis deals with the problem of steganography and the development of a mobile application for the Android platform that implements image steganography. In the theoretical part, the problem of image steganography is described, followed by an overview and testing of current steganography applications on the Android platform. The practical part deals with the design of the application including the original steganography algorithm and its development in the Flutter framework and subsequent testing.

Keywords:

Steganography, Image steganography, Mobile app, Android, Flutter

Děkuji Ing. Petru Žáčkovi, Ph.D. za vstřícnost a ochotu, kterou projevoval po celou dobu vedení bakalářské práce. Také svojí rodině a kolegům z práce za schovívavost a trpělivost se mnou v těchto nelehkých časech. Nakonec svému spolubydlícímu Tomovi za kávu, čaj a zapůjčení omláceného telefonu s Androidem 8.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

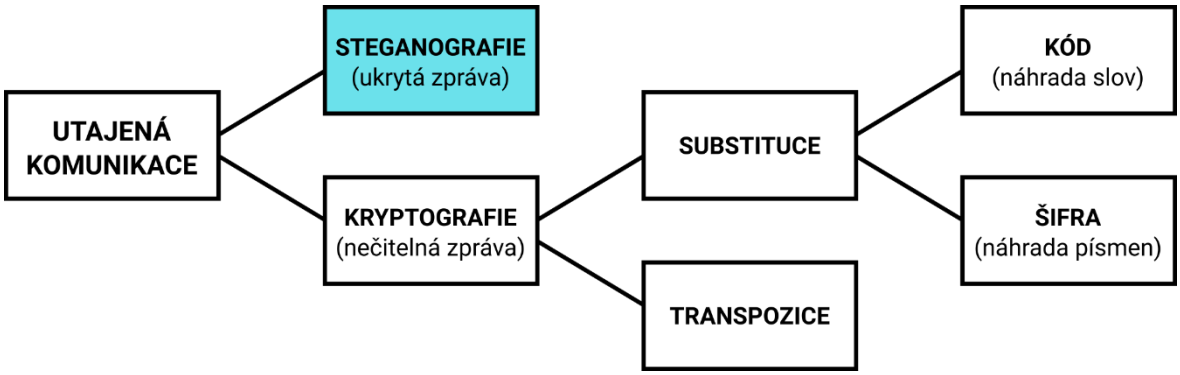
Pokud se zadíváme na jakoukoliv fotografii v našem telefonu nebo na sociální síti, tak z ní můžeme vyčíst několik věcí. Vidíme zde naše přátele, jak se baví, jak cestují a jak tráví svůj čas. Ne všechno však může být pouhým okem viditelné. V rámci dat obrázku se může skrývat tajná zpráva. Možná něco, co chceme skrýt před ostatními, soubor, jehož obsah by nás mohl dostat do problémů nebo nějaký nebezpečný malware. Schovávání souborů do obrázků se může využít ať už k dobrým účelům, jako je ochrana citlivých informací, tak i k daleko horším a nekalým účelům. I tak by bylo výhodné tuto možnost mít a používat k tomu počítač, který v dnešní době nosíme skoro všichni pořád s sebou – mobilní telefon.

Cílem této práce bylo nejenom probrat problematiku steganografie a najít vhodný algoritmus k implementaci, ale také navrhnout a vytvořit mobilní aplikaci pro platformu Android, která bude umět ukrývat soubory do obrázku a z obrázků je opět získávat. Tato aplikace bude zároveň přehledná a uživatelsky přívětivá. Díky tomu bude moct steganografii používat opravdu každý a ukrývat tak cokoliv do zdánlivě neškodných fotek.

I. TEORETICKÁ ČÁST

1 STEGANOGRAFIE

Steganografie je věda zabývající se ukrýváním tajných zpráv a informací tak, aby nebyly odhalitelné [1]. Slovo steganografie se skládá z řeckých slov *steganos* (schovaný) a *graphein* (psát) Společně s kryptografií a kryptoanalýzou spadá pod vědní obor kryptologie. Zatímco kryptografie se snaží zprávu zašifrovat tak, aby její přečtení nebylo možné bez klíče, tak existenci samotné zprávy se nesnaží nijak skrývat. Steganografie naopak obsah zprávy nijak nešifruje, ale její existenci se snaží zatajit úplně. Tady ovšem přichází největší slabina steganografie. Pokud někdo ukrytou zprávu odhalí, tak je schopen ji okamžitě celou přečíst. Z tohoto důvodu se doporučuje steganografii kombinovat s kryptografií. Pokud je i taková zpráva odhalena, tak je pořád chráněna šifrou, takže její přečtení vyžaduje buď znalost tajného klíče, nebo provést kryptoanalýzu [2].



Obrázek 1. Rozdělení utajené komunikace [2]

1.1 Historie steganografie

Steganografie se postupně vyvíjela společně s kryptografií a kryptoanalýzou, i když byla často pro svoji „jednoduchost“ kryptografy neprávem opomíjena, tak měla v historii a má i v současné době důležitý význam [3].

První zmínky o použití steganografie pochází z 5. století př. n. l. od řeckého historika Hérodota. Ten zmiňuje případ, kdy během války mezi Řeky a Peršany připravoval perský král Xerxes překvapivý útok na řecké loďstvo. Jeho přípravy však zpozoroval Řek Demaratus žijící v té době ve vyhnanství v perském městě Susy. Ten chtěl Sparťany varovat, věděl však, že by zprávu mohly zachytit perské hlídky. Rozhodl se tedy seškrábat vosk ze dvou psacích destiček a napsat zprávu přímo na dřevo. Tyto destičky pak znova zalil voskem a

poslal. Destičky se tak zdály na první pohled prázdné a tak nevzbudily u perských strážných žádné podezření. Řekové tedy věděli o útoku Peršanů předem a podařilo se jim v zálivu u Salaminy perskou flotilu odrazit [2].

Obrázek 2. Psací destička [4]

Hérodotos dále popisuje událost, kdy řecký vládce Histiaios nechal oholit hlavu svého nejvěrnějšího otroka a nechal na ni vytetovat zprávu, která měla pobouzet ke vzpouře proti perskému králi. Až vlasy zase dorostly, tak mohl otrok cestovat do cíle, kde mu byla hlava znova oholena a zpráva zase přečtena. Tenhle způsob ukrytí zprávy používali ještě němečtí špioni začátkem 20. století [5].

Dále se začalo rozvíjet používání neviditelných inkoustů, které se ze začátku vyráběly z organických substancí jako moč či mléko. Po zahřátí papíru se zprávou napsanou neviditelným inkoustem se zpráva odhalila. Později byly tyto inkousty vyráběny chemicky [5].

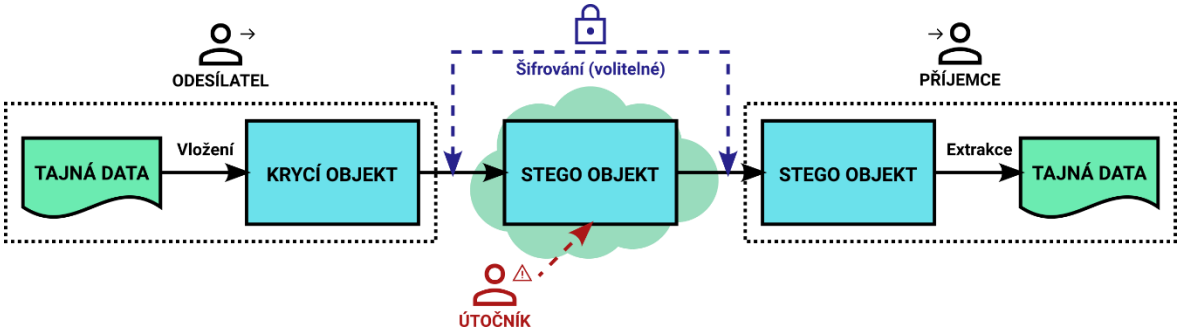
V období renesance se začala rozvíjet lingvistická steganografie. Italský matematik Gerolamo Cardano přišel s vynálezem zvaným Cardanova mřížka (též šifrovací mřížka). Jednalo se o papírovou kartu s proraženými otvory. Ta se přiložila na list papíru s textem a tím odhalila ukrytou zprávu v textu. Ačkoliv s touhle technikou přišel v Evropě až Cardano, v Číně byla tahle technika známa daleko dříve [5].

Během druhé světové války začali němečtí agenti v Latinské Americe používat k ukrytí tajných zpráv tzv. mikrotečky. Fotografickou cestou dokázali zmenšit celou stránku textu a

tu pak následně ukrýt do tečky ve zdánlivě neškodném dopise. Americká FBI tyto mikrotečky odhalila roku 1941 pomocí jemného odlesku, který způsoboval filmový materiál použitý v mikrotečce. Němečtí agenti však v některých případech zprávu před ukrytím šifrovali, takže byla zpráva i po odhalení chráněna před přečtením [2].

1.2 Digitální steganografie

S rozvojem a rozšířením počítačů se steganografie začala rozšiřovat i v digitální podobě. Princip digitální steganografie je poměrně jednoduchý a funguje podobně pro jakýkoliv typ mediálního souboru. Do originálního mediálního souboru, který nazýváme *krycí objekt*, vložíme *tajná data* nebo *watermark*. Tímto procesem vytvoříme tzv. *stego objekt*. Tento objekt se z většiny podobá originálnímu mediálnímu souboru a na první pohled by měl být nerozlišitelný od originálu. *Tajná data* pak lze ze *stego objektu* opět extrahovat [6]. Jako *krycí objekt* je vhodné použít jakékoliv médium s velkým množstvím redundance bitů. Silná steganografická technika by měla splňovat dvě zásady. Jakákoliv změna ve *stego objektu* by neměla porušit integritu *tajných dat* a *stego objekt* by měl vypadat stejně jako originální soubor, jinak by mohl vzbudit podezření [7].



Obrázek 3. Princip steganografie [7]

Digitální steganografii lze rozdělit podle *krycího objektu* na pět typů [8].

1.2.1 Steganografie textu

K urytí informací do textových souborů se dá použít mnoho různých metod jako například změna formátu u existující textu či změna slov v textu. Také lze v textových souborech generovat náhodné posloupnosti znaků či použít bezkontextovou gramatiku pro generování čitelných textů [8].

1.2.2 Steganografie obrazu

Díky tomu, že jsou obrázkové soubory tvořeny velkým množstvím bitů, jsou obrázky široce používané ke steganografii. Informace do obrázků lze ukrýt pomocí metod jako *Least Significant Bit Insertion* či *Redundant Pattern Encoding* [8].

1.2.3 Steganografie audia

U audio souborů se tajná zpráva ukrývá do audio signálu. Ukrytí zprávy do audia bývá v porovnání s jinými typy steganografie daleko obtížnější [8].

1.2.4 Steganografie videa

Velká výhoda steganografie videa je to, že u ní lze kombinovat jak steganografii obrazu, tak i steganografii audia, takže lze dosáhnout velké kapacity ukryté informace [8].

1.2.5 Síťová steganografie

Pomocí této technologie můžeme ukrývat tajné informace do řídících síťových protokolů jako například TCP, UDP či ICMP. Některé informace lze například ukrýt do hlavičky TCP/IP paketu [8].

1.3 Využití steganografie

Mezi základní využití steganografie patří především svobodná komunikace v prostředích, kde jsou kryptografií šifrované zprávy zakázané nebo by jejich použití vzbuzovalo podezření. Mezi další obory, které úzce souvisí se steganografií, lze řadit *watermarking* a *fingerprinting* [9].

Watermarking slouží především k ochraně autorských práv. Soubor je označen podpisem, který označuje původ souboru nebo jeho vlastnictví [9]. Watermark by měl být pořád detekovatelný i po změně souboru jako je například komprese dokumentu [5].

Fingerprinting naopak slouží jako ochrana proti porušení licenčních podmínek. Každá kopie souboru je opatřena unikátním fingerprintem pro každého zákazníka. Podle fingerprintu je pak možné dohledat konkrétního zákazníka, který licenční podmínky porušil [9].

Watermark i fingerprint může být navíc viditelný, zatímco u steganografie musí být tajná data vždy skryta [9].

Tabulka 1. Srovnání watermarkingu, fingerprintingu a steganografie [9].

	Watermarking	Fingerprinting	Steganografie
Účel	Ochrana duševního vlastnictví a autorských práv	Ochrana duševního vlastnictví a identifikace stran, které porušily licenční podmínky	Přenos tajných zpráv bez vzbuzení podezření
Nutnost neviditelnosti	Žádaná, ale ne zásadní	Žádaná, ale ne zásadní	Zásadní, aby informace nebyly viditelné
Odolnost proti odstranění, zničení či padělání	Zásadní, aby se nedaly odstranit vložené informace	Zásadní, aby se nedaly odstranit vložené informace	Žádaná, ale ne zásadní
Velká kapacita	Není důležitá, protože podpisy autorských práv bývají malé	Není důležitá, protože podpisy autorských práv bývají malé	Velmi důležitá, protože přenášené zprávy mohou být velké

1.4 Útoky na steganografii

Podobně jako kryptografické algoritmy, tak i ty steganografické mohou podléhat různým útokům. Sílu jednotlivých algoritmů můžeme měřit právě podle odolnosti vůči útokům, ačkoliv to není jediné měřítko, podle kterého bychom měli algoritmy soudit. Existuje mnoho různých útoků [1]. Následuje výčet těch nejčastějších:

1.4.1 Pouze soubor

Tento útok patří mezi jeden z nejslabších útoků. Útočník má přístup pouze k jednomu souboru, který může eventuálně obsahovat tajná data. Statickou analýzou obrázku či zvukového souboru se může pokusit zjistit, zda je v souboru obsažena tajná zpráva [1].

1.4.2 Soubor a kopie originálního souboru

Někdy může mít útočník přístup ke kopii originálního souboru a souboru se skrytou zprávou. Jednoduchým porovnáním těchto souborů může zjistit, zda byly do souboru přidána nějaká data či nikoliv. Útočník pak může soubor se skrytou zprávou zničit a nahradit originálem, či se pokusit zprávu přečíst nebo nahradit vlastní [1].

1.4.3 Více zakódovaných souborů

Může se stát, že útočník získá více kopií souborů, přičemž v každé kopii souboru je zakódována jiná zpráva. To děje například v případě personalizovaných watermarků používaných společnostmi, které distribuují zvukové soubory. Útočník pak může daný watermark odstranit či nahradit vlastním, popřípadě může dané souboru spojit dohromady, čímž vytvoří hybridní soubor bez watermarku [1].

1.4.4 Přístup k souboru a algoritmu

V případě, že útočník ví, jaký algoritmus byl k ukrytí zprávy použit, může se pokusit zprávu ze souboru extrahovat. Některé steganografické algoritmy ovšem používají při ukrývání informace klíč, bez kterého je získání informace velmi obtížné [1].

1.4.5 Zničení všeho

Velmi primitivní, leč účinný útok, kdy útočník jednoduše zničí soubor, který by mohl obsahovat tajnou zprávu [1].

1.4.6 Útoky náhodného „vylepšení“

Tento typ útoku se nesnaží nijak zjistit existenci tajné zprávy, ale pouze se ji snaží poškodit. Původní soubor „vylepší“ novými náhodnými daty a doufá, že tajnou zprávu natolik poškodí, že bude pro příjemce nečitelná. Pokud ovšem steganografický algoritmus používá nějaký samoopravovací kód, tak může tomuto útoku snadno odolat [1].

1.4.7 Přidání nové informace

Jedná se o jednoduchý útok, který za použití stejného softwaru přepíše původní tajnou zprávu v souboru jinou [1].

1.4.8 Útok reformátováním

Jeden z možných útoků se dá provést tím, že změníme formát souboru, čímž zprávu zničíme, jelikož různé formáty mají rozdílné rozložení dat. Dobře navržené watermarky však tomuto typu útoku odolají, jelikož například u počítačové grafiky dochází při práci se soubory ke změnám formátu poměrně často [1].

1.4.9 Útok kompresí

Poslední útok na steganografii je použití komprese. Nebezpečná je především tzv. ztrátová komprese, u které při dekompresi nedochází k přesnému, ale pouze k přibližnému obnovení původního souboru. Tento typ komprese používá například formát JPEG. Některé watermarky jsou navrženy tak, aby tomuto typu komprese odolaly [1].

2 TECHNIKY STEGANOGRAFIE OBRAZU

Technik, kterými lze ukrýt tajná data do obrázkových souborů existuje hned několik. Lze je rozdělit do tří kategorií – *prostorové*, *frekvenční* a *kompresní* [10]. V této kapitole se na některé z těchto technik podíváme blíž.

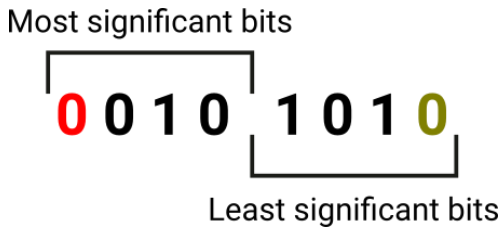
2.1 Prostorové techniky

Tyto techniky kódují zprávu přímo do jednotlivých pixelů obrázku a to jak do černobílých, tak i barevných. Mezi prostorové techniky patří *Least Significant Bit (LSB)*, *Bit Plane Complexity (BPC)*, *Histogram-based data hiding*, *Pixel Value Differencing (PVD)* a *Gray Level Modification data (GLM)* [10].

2.1.1 Least Significant Bit (LSB)

Jedná se o jeden z prvních steganografických systémů. Nazývá se tak podle způsobu, jakým se tajná data do obrázku schovávají [10]. Pouze s drobnými změnami v původním souboru můžeme ukrýt až překvapivě velké množství informací [5].

Slovem Least Significant Bit označujeme bit, který má na binární číslo nejmenší efekt. Opakem toho je Most Significant Bit (MSB), jehož změna dané binární číslo ovlivní nejvíce [11]. Rozdíl mezi MSB a LSB lze nejlépe demonstrovat na obrázku níže.



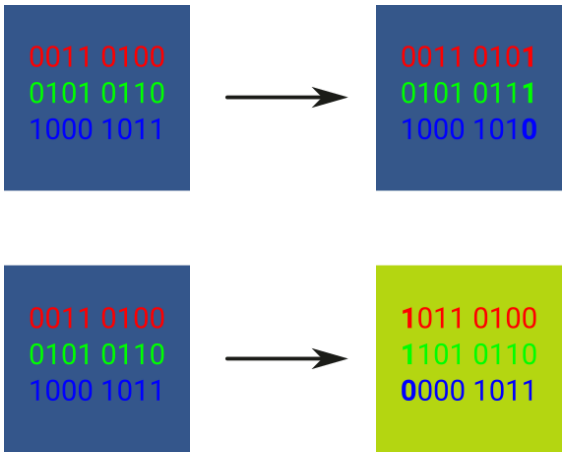
Obrázek 4. Rozdíl mezi MSB a LSB

Máme-li například binární číslo 0010 1010, jehož decimální hodnota je 42, tak změnou LSB dostaneme číslo 0010 101**1**, což je 43. Naopak změna MSB by nám dala binární číslo **1**010 1010, které má decimální hodnotu 170.

Většina monitorů i obrázkových formátů jako JPEG či PNG používají k zobrazení barev tzv. RGB (Red Green Blue) systém. Ten skládá jednotlivý pixel ze tří barevných kanálů: červené, zelené a modré. Počet odstínů, kterých můžou pixely nabývat, uvádí tzv. barevná hloubka. Ta může být 1 bitová (2 barvy), 4 bitová (16 barev), 8 bitová (265 barev), 16 bitová

(65536 barev), která se nazývá *High Color*, 24 bitová (16,7 miliónů barev), nazývaná *True Color* a 32 bitová (4,3 miliardy barev), taktéž nazývaná *Super True Color*. Právě 24 bitová hloubka se používá pro popis barev nejčastěji, kdy je každá barva prezentována právě 8 bity [12].

Pokud máme například barvu Classic Blue, tak ji můžeme v RGB systému s 24 bitovou hloubkou reprezentovat takto: 0011 0100 0101 0110 1000 1011 [13]. Po změně LSB v každém kanálu nám vyjde barva s RGB hodnotou: 0011 0101 0101 0111 1000 1010, která je od původní barvy pouhým lidským okem téměř nerozeznatelná. Zatímco při změně MSB by nám vyšla barva s hodnotou: 1011 0100 1101 0110 0000 1011, která se od původní barvy rapidně liší.



Obrázek 5. Rozdíl mezi změnou LSB a MSB

Používat k ukrytí zprávy MSB nedává smysl. Výsledný obrázek by byl od originálu k nepoznání. Zato změna LSB provede na výsledném obrázku minimální změny.

Pokud máme obrázek o velikosti 4032 x 3024 a ke kódování informace použijeme všechny tři barevné kanály, tak lze do tohohle obrázku ukrýt až 36 578 304 bitů informací, což je více než 4,36 MB. Pro srovnání, k urytí všech Shakespearových divadelních her a básní by nám stačily pouze dva takové obrázky [14].

Data do obrázku pomocí techniky LSB lze ukrýt dvěma způsoby: *sekvenčně* a *náhodně*. Sekvenční způsob ukládá data od prvního pixelu obrázku a bity tajných dat ukrývá postupně. Tenhle způsob je jednoduchý, ale dá se snadněji odhalit pomocí stegoanalýzy. Náhodné ukládání rozdělí zprávu do několika pseudonáhodných segmentů a ta se pak do obrazu

vkládá do různých oddělených částí, které jsou rozděleny pseudonáhodně, což výrazně stěžuje následnou stegoanalýzu [10].

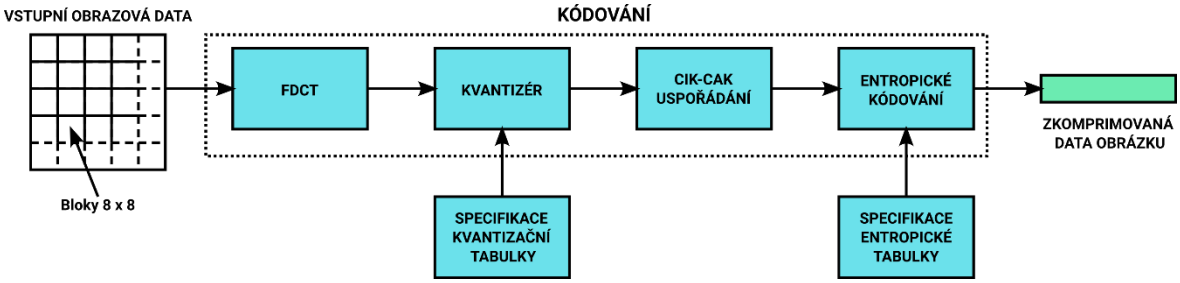
Pouhou změnou odstínu některých pixelů u metody LSB nijak nezměníme výslednou velikost souboru. Díky tomu lze ukrýt velké množství informací a přitom zachovat stejnou velikost krycího obrázku. Problém s metodou LSB spočívá v tom, že ve výsledném obrázku musí být pixely reprezentovány přesně, jinak dojde ke ztrátě informací [1]. Proto nelze u této metody použít formáty se ztrátovou kompresí jako JPEG, ale je nutné používat formáty s bezztrátovou kompresí jako PNG či BMP [15].

2.2 Frekvenční techniky

U těchto technik počítáme se ztrátovou kompresí u JPEG obrázků, takže je tajná zpráva čitelná i po kompresi souboru. Do frekvenčních technik patří *DCT steganografie*, *DWT steganografie* a *DFT steganografie* [10].

2.2.1 DCT steganografie

U této techniky se využívá diskrétní kosinové transformace, která se používá při kompresi u JPEG souborů. Obrázek je rozdělen do bloků o velikosti 8 x 8, kde jsou hodnoty pixelů převedeny na DCT koeficienty. Bloky 8 x 8 jsou následně kvantovány podle kvantizační tabulky [10].



Obrázek 6. Základní JPEG komprese [16]

Při DCT steganografii se používají dvě základní metody – *JSteg* a *JHide*. JSteg ukrývá data tak, že DCT koeficienty sekvenčně nahrazuje bity tajné zprávy. JHide zase nahrazuje LSB a second least significant bit v obrázku [10].

2.3 Techniky kompresního algoritmu

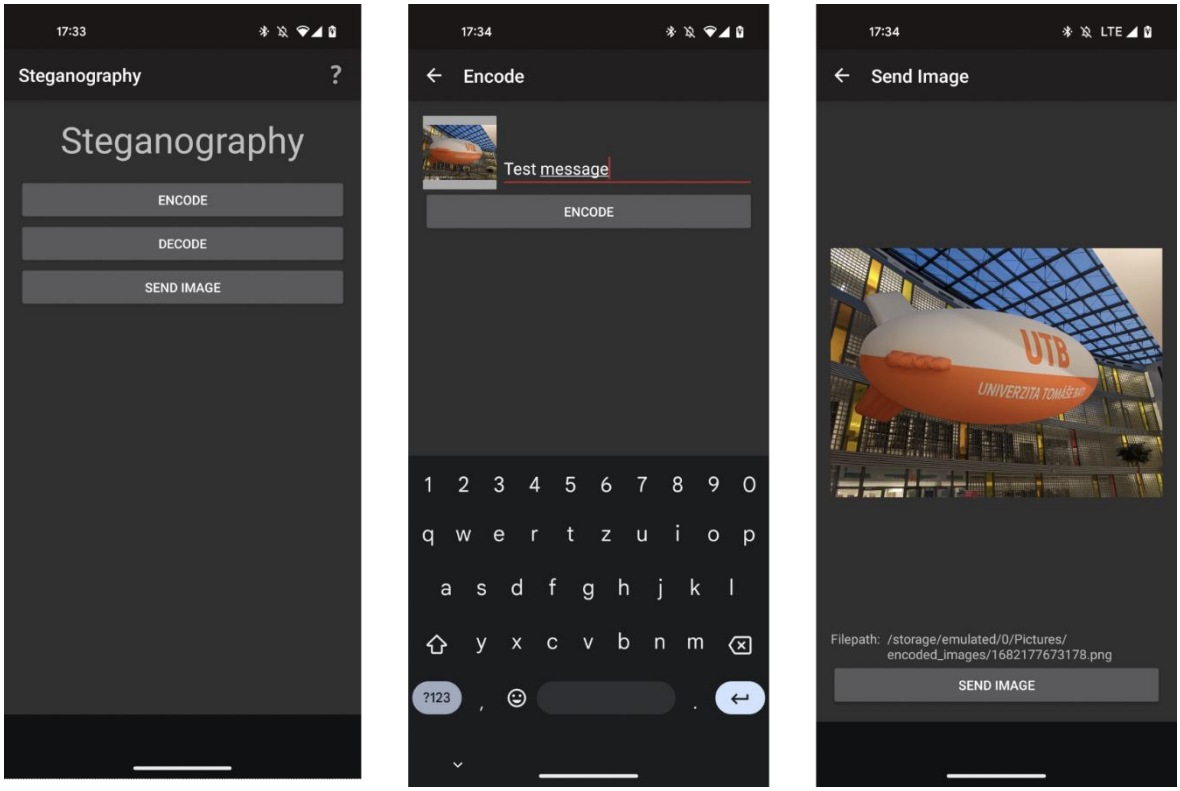
Existují i techniky, u kterých je proces vkládání tajných dat implementován přímo v kompresním algoritmu JPEG souboru. Tuto metodu používá například steganografický nástroj Jpeg-Jsteg [10].

3 STEGANOGRAFIE NA PLATFORMĚ ANDROID

V této kapitole se zaměříme na mobilní aplikace na platformě Android, které aplikují některou z technik steganografie obrazu. Budu testovat a porovnávat nejstahovanější aplikace z obchodu Google Play. U každé se pokusím ukrýt zprávu či soubor do obrázku a úspěšně ji zase z obrázku získat.

3.1 Steganography

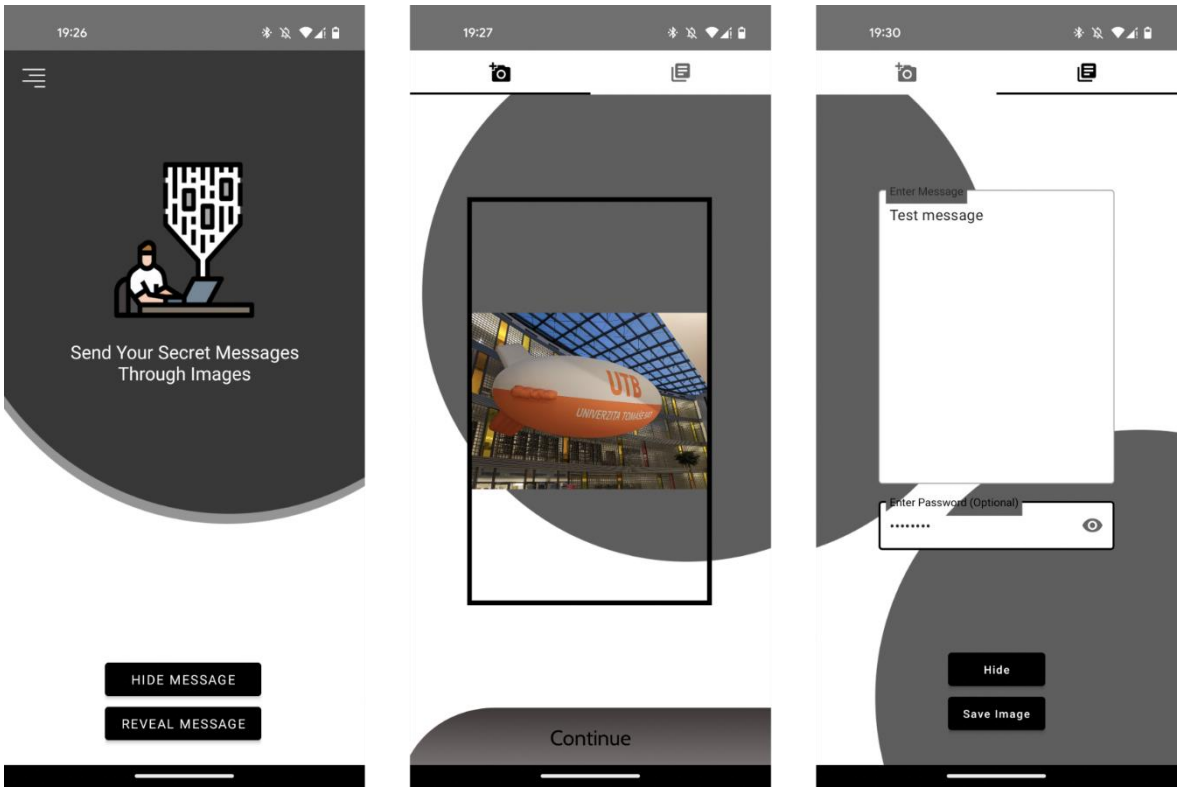
Jako první tu mám aplikaci Steganography, která je s více jak 10 000 staženími nejstahovanější steganografickou aplikací v obchodě Google Play [17]. I přesto je poměrně jednoduchá a umožňuje pouze ukrýt textovou zprávu do obrázku bez možnosti použít jakýkoliv klíč pro zašifrování zprávy. Uživatelské rozhraní je hodně jednoduché a vypadá dost zastarale. Aplikace při kódování navíc zanechává v telefonu v galerii šedé obdélníky, které nejspíš používá během šifrování a po sobě je neodstraňuje.



Obrázek 7. Ukázka z aplikace Steganography

3.2 NoClue

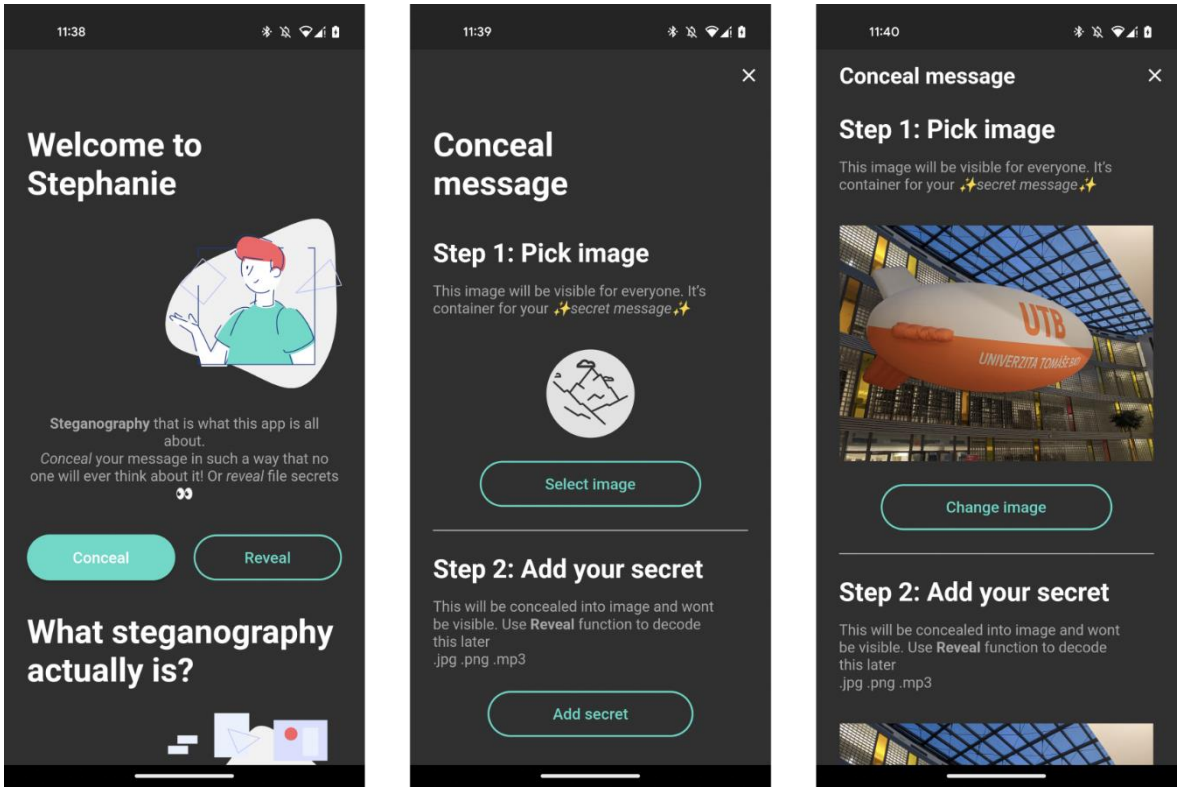
Jako další na řadě tu mám aplikaci NoClue [18]. U té jde na rozdíl od té předchozí vidět snaha autorů o lépe zpracované uživatelské rozhraní, které je místy bohužel pořád dost nedotažené. Aplikace umožňuje šifrovat jednoduché textové zprávy a to jak s klíčem, tak i bez klíče. Zprávu lze zase z obrázku získat, pokud použijeme správný klíč.



Obrázek 8. Ukázka z aplikace NoClue

3.3 Stephanie

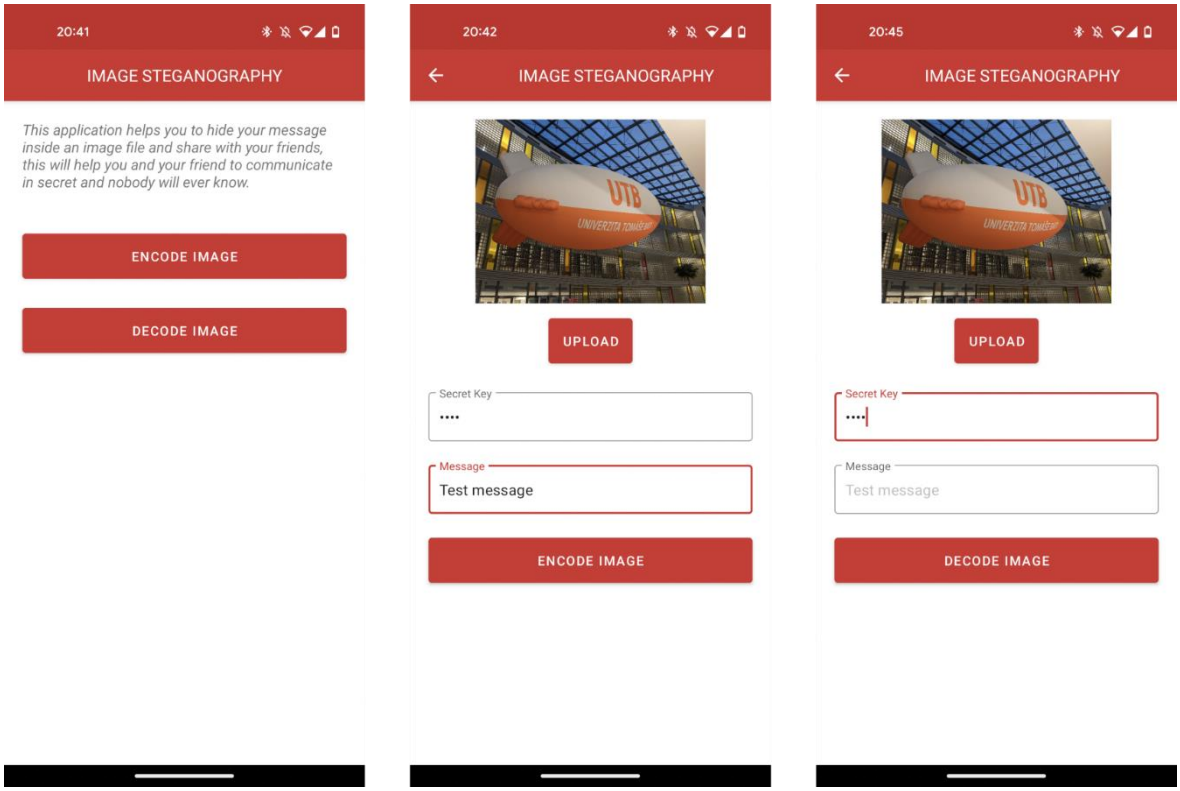
Další aplikace, kterou jsem testoval, je aplikace Stephanie [19]. Ta na rozdíl od těch předchozích nenabízí steganografii textových zpráv, ale celých souborů a to konkrétně souborů JPEG, PNG a MP3. Je také z hlediska uživatelského rozhraní velmi moderně a přehledně zpracovaná. Soubory ukrývá do JPEG obrázků a bohužel neumožňuje k ukrytí použít tajný klíč. Během testování jsem použil obrázek UTB vzducholodě a pokusil jsem ho ukrýt sám do sebe. To se aplikaci sice úspěšně podařilo, ale velikost původního obrázku se zvětšila ze 3 MB na 16 MB, což je obrovský rozdíl oproti originálnímu obrázku a rozhodně budí podezření. Pokud zkusíme aplikací odhalit skrytý soubor v obrázku, kde žádný skrytý soubor není, tak aplikace odhalí MP3 soubor, který ovšem nejde přehrát.



Obrázek 9. Ukázka z aplikace Stephanie

3.4 Image Steganography

Poslední aplikace, kterou jsem testoval, se nazývá Image Steganography [20]. Ta je svojí funkcionalitou hodně podobná aplikaci NoClue a opět umožňuje ukrývat pouze textové zprávy šifrované klíčem. Klíč je u této aplikace povinný, takže nelze ukrývat data bez zadání klíče. Uživatelské rozhraní je zpracováno jednoduše a přehledně, ačkoliv má v jistých oblastech pořád rezervy. Zpráva se dá úspěšně ukrýt do krycího obrázku a po zadání správného klíče zase získat zpět.



Obrázek 10. Ukázka z aplikace Image Steganography

3.5 Shrnutí

Rozdíly mezi jednotlivými aplikacemi ještě shrnuji v následující tabulce.

Tabulka 2. Srovnání steganografických aplikací na platformě Android

Název	Steganography	NoClue	Stephanie	Image Steganography
Formát tajných dat	Pouze text	Pouze text	JPEG, PNG, MP3	Pouze text
Klíč	Žádný	Volitelný	Žádný	Povinný
Formát stego obrázku	PNG	PNG	JPEG	PNG
Uživatelské rozhraní	Zastaralé	Nedotažené	Moderní a přehledné	Moderní, ale nepřehledné

Ačkoliv má každá z aplikací své rezervy ve zpracování, tak každá splňuje základní funkcionalitu – možnost ukrýt tajnou zprávu či data do obrázku a zase je odhalit. Jako nejlépe

zpracovaná na mě působí aplikace Stephanie, která je nejen nejlépe provedená z hlediska uživatelského rozhraní, ale umožňuje ukrýt celý soubor do JPEG obrázku. Její hlavní nevýhodou je nemožnost použít klíč k dodatečnému zašifrování souboru a také několikanásobné zvětšení stego obrázku oproti originálu, které bezesporu budí podezření.

U aplikace, kterou budu vyvíjet, chci především možnost šifrovat jakýkoliv binární soubor, což žádná z testovaných aplikací nenabízí. Dále pro zvýšení bezpečnosti chci dát uživateli možnost použít šifrovací klíč. V neposlední řadě chci přehledné a moderní uživatelské rozhraní, které ne všechny testované aplikace nabízely.

II. PRAKTICKÁ ČÁST

4 NÁVRH MOBILNÍ APLIKACE

V téhle části bakalářské práce se zaměřím na podrobnější návrh celé mobilní aplikace. Analyzuji si funkční a nefunkční požadavky, dále si navrhnu přehledné a moderní uživatelské rozhraní a v neposlední řadě vypracuji návrh vlastního steganografického algoritmu, který bude vycházet z techniky LSB řízené klíčem.

4.1 Analýza požadavků

Jako první krok, který provedu při návrhu aplikace, je analýza funkčních a nefunkčních požadavků. Pomocí nich si podrobněji upřesníme to, jak má aplikace vypadat a co všechno má umět.

4.1.1 Funkcionální požadavky

Pod funkcionální požadavky bude spadat veškerá funkcionalita, kterou bude aplikace schopna vykonávat tak, aby plnila svůj účel.

1. **Ukrývání souboru do obrázku**
   - Aplikace bude schopna ukrýt jakýkoliv binární soubor ze zařízení, který nebude překračovat kapacitu obrázku, do jakéhokoliv souboru formátu PNG či JPEG ze zařízení pomocí vlastního steganografického algoritmu.
2. **Odhalování souborů z obrázků**
   - Aplikace bude schopna detekovat a získat binární soubor z obrázku ve formátu PNG, u kterého byl použit k ukrytí steganografický algoritmus aplikace.
3. **Šifrování tajných dat**
   - Aplikace bude schopna šifrovat ukrývaný binární soubor pomocí uživatelem zadaného klíče. Bez správného klíče taky nebude možné daný binární soubor získat. Steganografický algoritmus bude fungovat i bez zadání šifrovacího klíče.
4. **Zpracování chyb**
   - Aplikace bude schopna efektivně a pružně reagovat na chyby vzniklé při ukrývání či odhalování souboru, nebo při volbě špatných vstupních souborů, tak, aby nedocházelo k pádu či zamrznutí aplikace.
5. **Uživatelské rozhraní**
   - Aplikace bude nabízet přehledné a moderní uživatelské rozhraní, které se bude snadno používat a bude rovněž obsahovat tmavý i světlý motiv. Aplikace bude také

schopna uživateli jasně a srozumitelně sdělit, pokud dojde k nějaké chybě během ukrývání či odhalování souboru.

6. **Jazyk aplikace**
   - Aplikace bude obsahovat dva jazyky – Angličtinu a Češtinu. Defaultní jazyk aplikace bude Angličtina. Uživatel bude mít možnost si jazyk přepnout v nastavení aplikace.

4.1.2 Nefunkcionální požadavky

Nefunkcionální požadavky nám budou specifikovat vlastnosti a omezení aplikace tak, aby aplikace byla spolehlivá a použitelná pro uživatele.

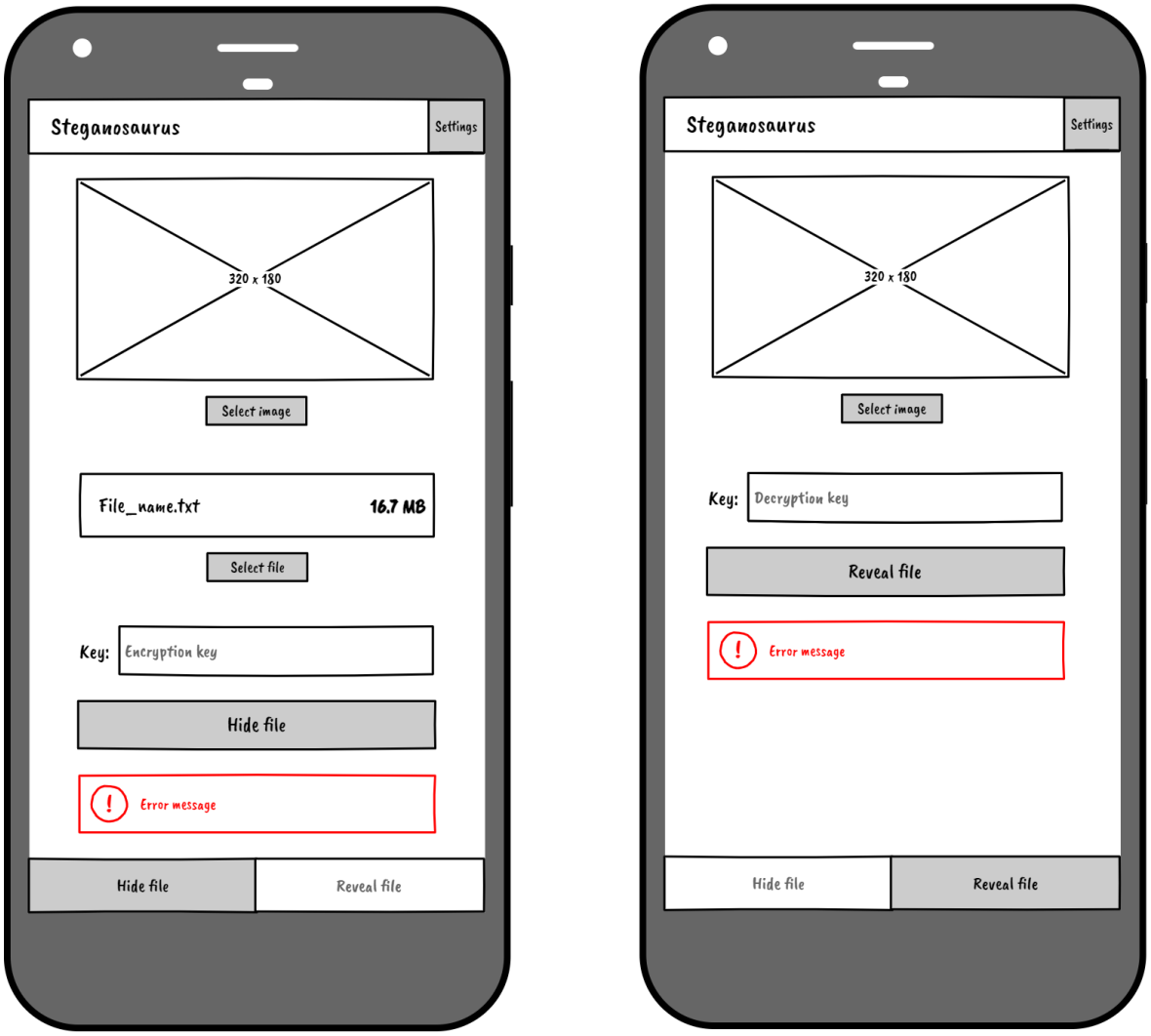
1. **Operační systém**
   - Aplikace bude schopna plynule a spolehlivě fungovat na operačním systému Android 9 (API 28) a vyšším.
2. **Bezpečnost**
   - Aplikace bude po operačním systému požadovat pouze přístup k úložišti zařízení. Aplikace bude zaručovat, že při práci s obrázky a binárními soubory nedojde k poškození či smazání původních souborů.
3. **Použitelnost**
   - Aplikaci by měl být schopen používat i uživatel, který má minimální, nebo žádné zkušenosti se steganografií.
4. **Plynulost**
   - Aplikace by měla být schopna bezproblémově běžet i na slabších zařízeních.
5. **Škálovatelnost**
   - Projekt by měl být rozumně a škálovatelně rozvržen tak, aby bylo snadné jeho pozdější rozšíření o nové funkcionality, jazykové balíčky, atd.

4.2 Návrh uživatelského rozhraní

V této části si navrhnu uživatelské rozhraní pomocí wireframů. Dále si navrhnu ikonu aplikace, která bude mou aplikaci vhodně reprezentovat.

4.2.1 Wireframy

Wireframy jsem si navrhnul pomocí open source aplikace Pencil [21], která nabízí základní nástroje pro tvorbu wireframů a prototypů. Hlavní obrazovka aplikace vypadá následovně:

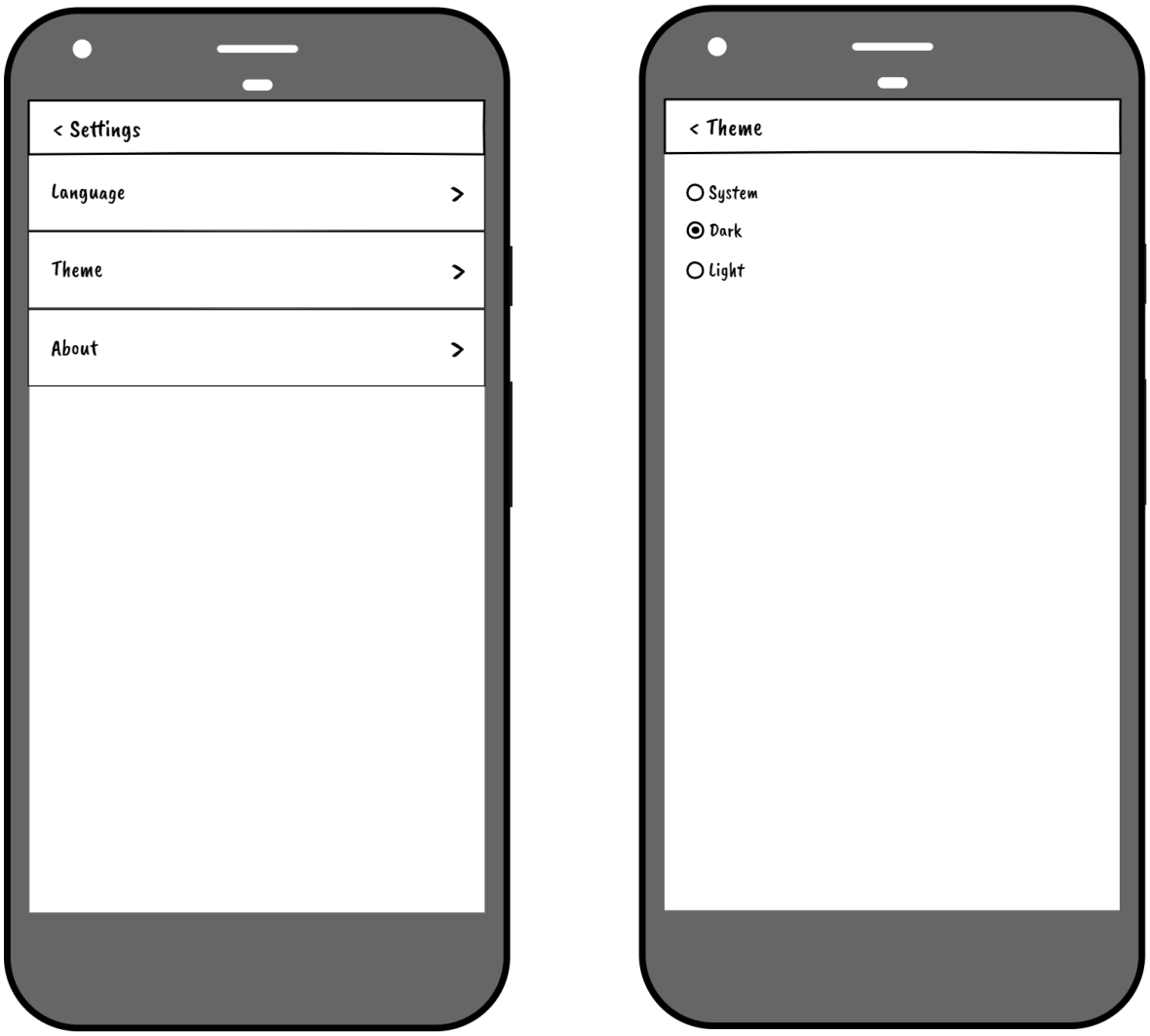


Obrázek 11. Wireframy hlavní obrazovky

Hlavní obrazovka je rozdělena na dvě části, mezi kterými lze přepínat spodním navigačním panelem. První z nich je formulář na ukrytí souboru. Ten obsahuje Náhled vybraného obrázku s tlačítkem, které otevře průzkumník souborů v telefonu, pomocí kterého uživatel vybere obrázek. Dále tu je tlačítko, pomocí kterého uživatel vybere binární soubor. Informace o souboru se pak zobrazí v náhledu souboru, kde bude vypsán název souboru a jeho velikost. Níže se nachází textové pole na zadání klíče k zašifrování tajných dat. Nakonec je zde tlačítko na spuštění procesu ukrytí souboru do obrázku. Pokud vybrané soubory neexistují, nebo velikost binárního souboru přesahuje kapacitu obrázku, tak se pod tlačítkem objeví chybová hláška a proces se nespustí.

Formulář na odhalení souboru v obrázku je obdobný, akorát zde chybí náhled souboru a tlačítko k vybrání souboru, protože to při odhalování souboru nedává smysl. Opět se tu objeví chybová hláška, pokud vybraný obrázek nebude existovat.

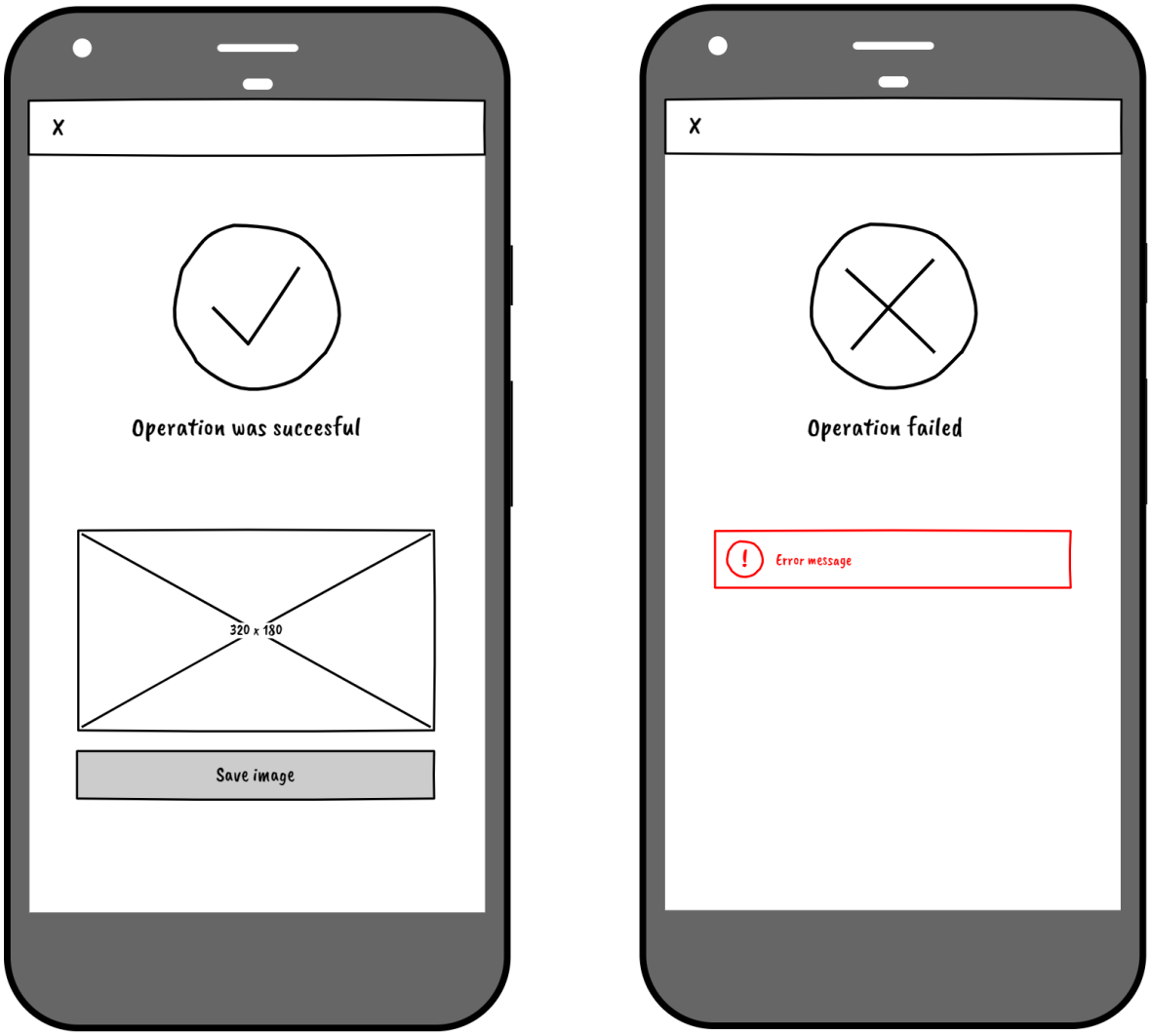
V pravém horním rohu se nachází tlačítko, které uživatele přesměruje do nastavení aplikace. Obrazovka nastavení aplikace vypadá následovně:



Obrázek 12. Wireframy obrazovky nastavení

Obrazovka nastavení je rozdělena do sekcí podle typu nastavení. Po rozkliknutí sekce *Language* nebo *Theme* se zobrazí další obrazovka, kde bude mít uživatel na výběr z několika přepínacích tlačítek pro nastavení jazyka či motivu. Nastavené změny se projeví okamžitě po zatrhnutí tlačítka.

Jako poslední tady mám wireframe, který se zobrazí po úspěšném ukrytí souboru do obrázku či po úspěšném odhalení souboru v obrázku:



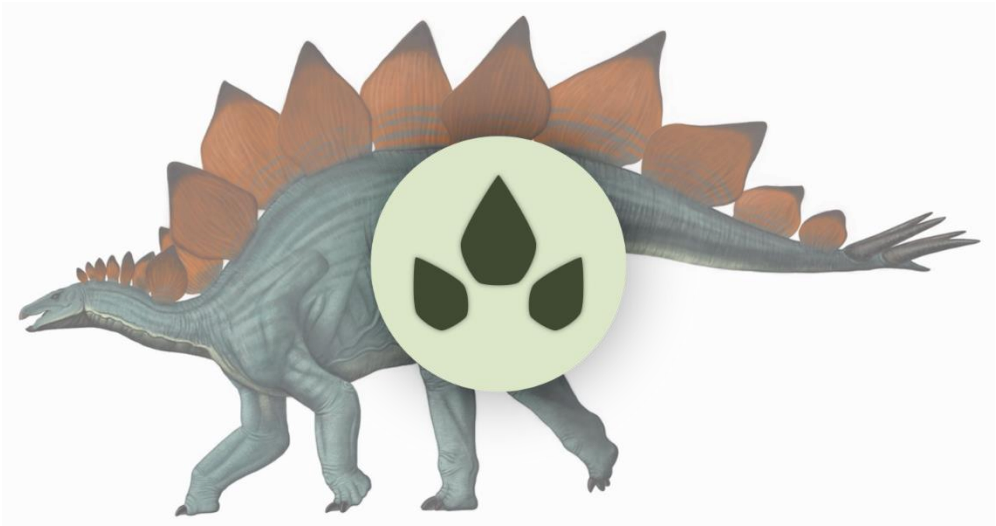
Obrázek 13. Wireframy obrazovky s výsledkem operace

Na obrazovce je napsán výsledek operace – buď byla úspěšná, nebo selhala. Pokud byla operace úspěšná, tak je na obrazovce i náhled stego obrázku či odhaleného souboru a tlačítko, které stáhne daný obrázek či soubor. Pokud operace selže, tak se na obrazovce objeví chybová hláška s podrobnějším popisem chyby. Vlevo nahoře na obrazovce je křížek, který po stisknutí vrátí uživatele zpět na hlavní obrazovku.

4.2.2 Návrh ikony aplikace

Vhodná ikona aplikace je rovněž důležitá. Ikona je to, co aplikaci reprezentuje na ploše a první věc, které si uživatel všimne v obchodě s aplikacemi. Protože ikony bývají zpravidla malé, není dobré dělat je přehnaně detailní či složité.

Svoji aplikaci jsem pojmenoval Steganosaurus, což je přesmyčka slov steganografie a Stegosaurus. Proto by bylo vhodné ikonu aplikace odvodit od Stegosaura samotného. Celý dinosaurus je na ikonu aplikace příliš detailní, tak jsem se rozhodl použít na ikonu aplikace požít pouze ploché hřbetní desky, kterými je tento ještěr známý.



Obrázek 14. Návrh ikony aplikace [22]

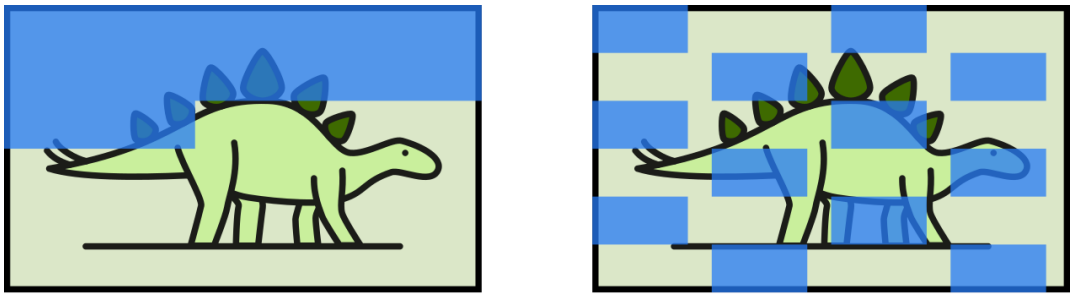
Na ikoně jsou vyobrazeny tři stylizované hřbetní desky, které vrhají nepatrný stín kvůli kontrastu s pozadím. Ikona je velmi minimalistická a je barevně laděna v zelených odstínech, do kterých chci stylizovat celou aplikaci. Všechny potřebné soubory pro ikonu Android aplikace jsem vygeneroval pomocí on-line nástroje *IconKitchen* [23].

4.3 Návrh steganografického algoritmu

Poslední a důležitá část návrhu je samotný algoritmus. Ten bude vycházet z metody LSB řízené klíčem, ale jeho provedení jsem se rozhodl zpracovat po svém tak, aby nebylo podobné žádnému existujícímu algoritmu.

4.3.1 Rozložení tajných dat v obrázku

Jeden s prvních problémů, který nastává při zapisování dat do obrázku, že pokud začneme data bit po bitu zapisovat od levého horního rohu, tak budou veškerá tajná data zapsána v horní části obrázku. To může případnému útočníkovi zjednodušit stegoanalýzu a tajná data tak mohou být odhalena. Tento problém se dá částečně odstranit tím, že obrázek rozdělíme na několik menších sekcí, například na 12 a data se v něm pak rozloží rovnoměrněji. Takové rozložení je demonstrováno na obrázku níže, kde modré části představují tajná data.



Obrázek 15. Demonstrace rozložení tajných dat v obrázcích

Ve svém algoritmu jsem se rozhodl data do obrázku rozložit na 13 částí. Tím, že jsem zvolil prvočíslo, tak budou data v obrázku rozprostřena na první pohled náhodně. Data jsou tedy v obrázku rovnoměrněji rozložená, avšak jejich pozice by byla v každém stego obrázku stejná, takže útočník znalý algoritmu by mohl data z obrázku snadno získat. Proto by bylo dobré měnit startovní pozici, na které se data začnou zapisovat. Tuto pozici nám bude určovat právě klíč.

4.3.2 Trojí použití klíče

Klíč jako takový bude tvořit textový řetězec zadaný uživatelem. Klíč nám bude určovat tři různé klíčové hodnoty:

- **Pozice startovního pixelu pro vkládání dat**
- **Startovní barevný kanál**
- **Binární řetězec pro modifikaci tajných dat**

Tyto hodnoty budeme moct poskládat z různých vlastností klíče. Nejlépe si to můžeme demonstrovat na příkladu. Řekněme, že máme obrázek o rozměrech 4032 x 3024 pixelů a

klíč zadaný uživatelem: *„Key042*“*. Každý znak klíče má svou ASCII hodnotu. Tyto hodnoty můžeme právě využít na sestavení klíčových hodnot. ASCII hodnoty našeho klíče jsou uvedeny v následující tabulce.

Tabulka 3. ASCII reprezentace klíče

Znak	**K**	**e**	**y**	**0**	**4**	**2**	*****
Decimální ASCII hodnota	75	101	121	48	52	50	42
Binární ASCII hodnota	0100 1011	0110 0101	0111 1001	0011 0000	0011 0100	0011 0010	0010 1010

Z těchto ASCII hodnot si pak vytvoříme jednotlivé klíčové hodnoty.

4.3.2.1 Pozice startovního pixelu pro vkládání dat

Tato klíčová hodnota nám bude udávat pozici pixelu, od kterého začneme vkládat tajná data. Bude to celočíselná hodnota od 0 do počtu pixelů obrázku, takže jeho šířka krát výška. Pozici startovního pixelu vypočteme tak, že součet decimálních hodnot všech znaků klíče na druhou vynásobíme počtem všech bitů s hodnotou 1 ve všech znacích. Toto číslo poté zmodulujeme počtem pixelů v obrázku. Pokud použijeme náš modelový příklad, tak je výpočet následující:

$P_s = (489^2 * 24) \bmod (4\,032 * 3\,024) = 5\,738\,904 \bmod 12\,192\,768 =$ ***5 738 904***

Algoritmus tedy začne zapisovat data do pixelu na pozici $P_S(1368, 1423)$, tj. v 1369. sloupci zleva a 1424. řádku shora (Pokud jako počáteční pixel se souřadnicí $P_P(0, 0)$ bereme levý horní pixel).

4.3.2.2 Startovní barevný kanál

Barevné kanály v každém pixelu máme celkem tři – červený, zelený a modrý. Náš algoritmus bude využívat pro zapisování dat všechny tři tyto kanály a při zapisování je bude po každém bitu střídat v pořadí červený, zelený a modrý. Startovní bit se však bude měnit podle délky klíče modulované počtem kanálů. Výpočet startovního kanálu je následující:

$C_s = 7 \bmod 3 =$ ***1***

Tajná data teda začneme zapisovat ve 2. kanálu, tj. v zeleném kanálu (červený kanál má index 0).

4.3.2.3 Binární řetězec pro modifikaci tajných dat

Tento řetězec bude modifikovat zapisované bity. Získáme ho jednoduše tak, že vezmeme binární ASCII hodnotu všech znaků klíče. Pro náš modelový případ bude binární řetězec vypadat takto:

01001011 01100101 01111001 00110000 00110100 00110010 00101010

Při zapisování tajných dat budeme postupovat bit po bitu binárním řetězcem. Pokud bude mít bit v řetězci hodnotu 0, tak zapisovaný bit tajných dat nijak nemodifikujeme. Pokud bude mít hodnotu 1, tak jej změníme. Provádíme vlastně logický XOR mezi bitem tajných dat a bitem binárního řetězce. Pokud dojdeme na konec binárního řetězce, tak začneme znova od začátku.

4.3.3 Demonstrace fungování algoritmu

Máme tedy obrázek o rozměrech 4032 x 3024 pixelů a klíč ve tvaru: „Key042*“. Řekněme, že do obrázku budeme chtít ukrýt 2 byty tajných dat ve tvaru: 01000110 01010101. Podle klíče si zvolíme počáteční pozici P_S(1368, 1423) a počáteční RGB kanál zelený. Tyto data pak začneme zapisovat a zároveň modifikovat binárním řetězcem klíče. Po každém zapsání změníme oblast zápisu a barevný kanál. Těchto oblastí bude celkem 13 a budou se periodicky opakovat tak, že budeme k předchozí pozici přičítat číslo 937 905. Po každém třináctém opakování se pozice nastaví na počáteční pozici, ale posune se o jedno doprava. Celý tento cyklus lze opakovat 3x, přičemž pokaždé se nastaví jiná hodnota startovního barevného kanálu. Postupný zápis je demonstrován v následující tabulce:

Tabulka 4. Demonstrace fungování algoritmu

Pozice	Barevný kanál	Binární řetězec	Tajná data	Zapisovaná data
P_0(1368, 1423)	Green	0	0	**0**
P_1(3849, 1655)	Blue	1	1	**0**
P_2(2298, 1888)	Red	0	0	**0**
P_3(747, 2121)	Green	0	0	**0**
P_4(3228, 2353)	Blue	1	0	**1**

P_5(1677, 2586)	Red	0	1	**1**
P_6(126, 2819)	Green	1	1	**0**
P_7(2607, 27)	Blue	1	0	**1**
P_8(1056, 260)	Red	0	0	**0**
P_9(3537, 492)	Green	1	1	**0**
P_{10}(1968, 725)	Blue	1	0	**1**
P_{11}(435, 958)	Red	0	1	**1**
P_{12}(2916, 1190)	Green	0	0	**0**
P_{13}(1369, 1423)	Blue	1	1	**0**
P_{14}(3850, 1655)	Red	0	0	**0**
P_{15}(2299, 1888)	Green	1	1	**0**

Díky takovému systému můžeme využít každý pixel 3x. Kapacita tohoto obrázku tedy bude *{(4032 * 3024) – [(4032 * 3024) mod 13]} * 3 =* ***36 578 395*** bitů. 3 pixely nebudeme moct použít, protože nejdou rozdělit na 13 částí.

Pokud uživatel nezadá žádný klíč, tak se klíčové hodnoty nastaví na 0, 0 a 0. Tudíž algoritmus začne zapisovat data na pozici P_S(0, 0), v červeném kanálu a zapisovaná data nebude nijak modifikovat.

5 IMPLEMENTACE NÁVRHU

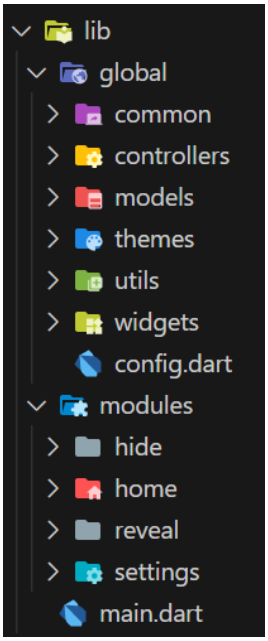
V této části bakalářské práci budu popisovat, jak jsem postupoval při implementaci navržené aplikace.

5.1 Použité technologie

Pro vývoj jsem si zvolil multiplatformní open-source Framework Flutter. Ten nabízí rychlý a snadný vývoj moderních aplikací na mobilní, desktopové platformy i web. Navíc chytré funkce jako Hot Reload, která umožňuje sestavit aplikaci za běhu bez nutnosti ji restartovat [24].

5.2 Struktura projektu

Základem dobrého projektu je dobře navržená adresářová struktura tak, aby byl kód přehledný a dal se bez problémů dál rozšiřovat. U Flutteru existuje víc přístupů, jak takovou strukturu vytvořit. Vybral jsem si tzv. *Feature-first* přístup, která radí dělit projekt podle jednotlivých funkcí. Každá funkce má pak vlastní *view* i *controller*. Dále je tu složka *global*, která obsahuje komponenty sdílené napříč celou aplikací. Podrobnější struktura je zobrazena na obrázku níže.



Obrázek 16. Struktura projektu

Ve složce *common* se nacházejí jednoduché komponenty využívané napříč aplikací jako tlačítka či navigační lišty. V *controllers* pak univerzální controllery používané na více místech. V *models* pak třídy jednotlivých datových struktur. V *themes* definice tmavého a světlého motivu. Složka *utils* obsahuje sdílené konstanty a složka *widgets* složitější widgety používané napříč aplikací.

Složka modules obsahuje části aplikace rozdělené podle funkcí. Složka *home* obsahuje domovskou obrazovku, složky *hide* a *reveal* view a controllery pro ukrývání souborů do obrázků a jejich odhalování. V *settings* je pak nastavení aplikace rozdělené podle sekcí. Hlavní činnost aplikace spouští soubor *main.dart*.

5.3 Nastavení aplikace

Uživatel si v aplikaci může nastavit jazyk aplikace a její motiv. Tyto nastavení si musí aplikace pamatovat i po vypnutí a opětovném zapnutí. Proto je dobré používat tzv. local storage, které umožňuje uložit jednoduché proměnné do paměti zařízení.

5.3.1 Jazyk

K implementaci jazyka jsem se rozhodl využít package *easy_localization*. Ten umožňuje rychlou a snadnou implementaci podporovaných jazyků a jejich přepínání. V souboru main.dart je třeba před spuštěním aplikace definovat podporované jazyky a inicializovat je.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await EasyLocalization.ensureInitialized();
  runApp(EasyLocalization(
      supportedLocales: SupportedLocales.all,
      path: "assets/translations",
      saveLocale: true,
      child: const SteganosaurusApp()));
}
```

Jednotlivé soubory s překlady jsou pak ve formě JSONu uloženy ve složce *assets* a v podsložce *translations*. Tam lze podle jednotlivých klíčů dohledat konkrétní překlady. V kódu se pak takový klíč přeloží pomocí funkce `tr('key')`. Tento package také automaticky ukládá nastavený jazyk do local storage, takže se o to nemusíme starat sami. Jazyk aplikace lze nastavit funkcí `context.setLocale(locale)`.

5.3.2 Motiv

Motiv aplikace lze ve Flutteru nastavit v souboru *main.dart* při vytváření aplikace v property `themeMode`. Také tam specifikujeme podobu jednotlivých motivů v properties `theme` a `darkTheme`. Ty jsem si definoval ve třídě *CustomThemes*.

```
class CustomThemes {
  /// Dark Material 3 theme
  static ThemeData dark = ThemeData(
      colorSchemeSeed: Styles.seedColor,
      brightness: Brightness.dark,
      useMaterial3: true);

  /// Light Material 3 theme
  static ThemeData light = ThemeData(
      colorSchemeSeed: Styles.seedColor,
      brightness: Brightness.light,
      useMaterial3: true);
}
```

Property `colorSchemeSeed` nám určuje, jaká barva se použije jako výchozí pro odvozování odstínů barev jednotlivých komponent jako tlačítek, pozadí aplikace a písma. V tomto případě jsem zvolil světle zelenou barvu. Díky tomu je pak celé uživatelské rozhraní sladěné v jedné barvě. O přepínání motivů a uložení jejich nastavení do local storage se nám stará vlastní třída *ThemeHolder*.

```
class ThemeHolder with ChangeNotifier {
  /// Holds current theme mode.
  ThemeMode _currentThemeMode = ThemeMode.system;

  /// Checks if theme was already loaded from locale storage.
  bool _wasLoaded = false;

  /// Key in local storage.
  final _key = 'currentTheme';

  /// Loads theme in constructor.
  ThemeHolder() {
    loadTheme();
  }

  /// Loads theme from locale storage.
  Future<void> loadTheme() async {
    final prefs = await SharedPreferences.getInstance();
    if (prefs.containsKey(_key)) {
      String savedTheme = prefs.getString(_key)!;
```

```
      switch (savedTheme) {
        case ThemeKeys.dark:
          _currentThemeMode = ThemeMode.dark;
          break;
        case ThemeKeys.light:
          _currentThemeMode = ThemeMode.light;
          break;
        default:
          _currentThemeMode = ThemeMode.system;
          break;
      }
    } else {
      prefs.setString(_key, ThemeKeys.system);
    }
    _wasLoaded = true;
  }

  /// Gets current theme.
  ThemeMode currentThemeMode() {
    if (!_wasLoaded) {
      loadTheme().then((_) => notifyListeners());
    }
    return _currentThemeMode;
  }

  /// Change current theme.
  Future<void> changeMode(ThemeMode mode) async {
    final prefs = await SharedPreferences.getInstance();
    _currentThemeMode = mode;
    switch (mode) {
      case ThemeMode.dark:
        prefs.setString(_key, ThemeKeys.dark);
        break;
      case ThemeMode.light:
        prefs.setString(_key, ThemeKeys.light);
        break;
      default:
        prefs.setString(_key, ThemeKeys.system);
        break;
    }
    notifyListeners();
  }
}
```

Jako defaultní nastavení motivu je použito systémové nastavení, tj. aplikace nastaví motiv podle motivu operačního systému. Pomocí package *shared_preferences* třída ukládá nastavovaný motiv a zároveň jej opět načítá po znovuspuštění aplikace. O každé změně motivu dá třída vědět pomocí funkce `notifyListeners()`.

5.4 Uživatelské rozhraní

U uživatelského rozhraní je důležitý nejen moderní zpracování, ale i jeho jednotnost napříč celou aplikací. Návrh takového rozhraní může být časově náročný proces, na kterém se obvykle podílí grafik. Proto jsem se místo vlastního grafického rozhraní rozhodl řídit grafickým manuálem *Material 3* [25]. Ten specifikuje, jak by měla současná Android aplikace vypadat. Flutter navíc většinu těchto komponent už v základu obsahuje, takže není třeba psát velké množství kódu.

5.4.1 Tvorba vlastních komponent

Kromě základních komponent jako tlačítek a navigační lišty jsem se rozhodl implementovat vlastní, které mi pomohou k lepšímu uživatelskému zážitku. Konkrétně se jedná o komponenty, které uživateli zprostředkují náhled vybraného obrázku a souboru.

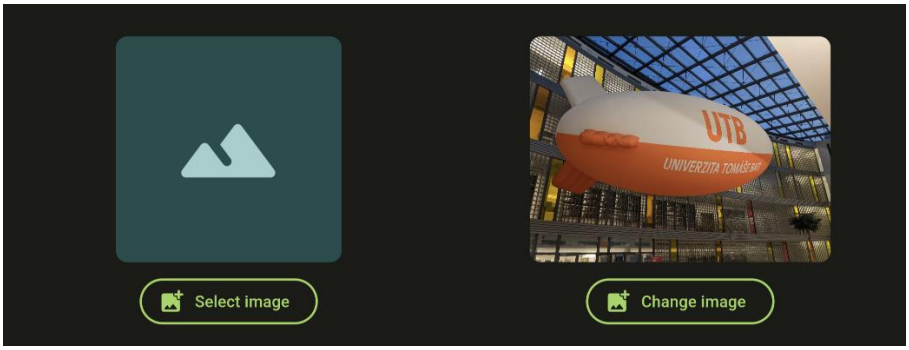
5.4.1.1 Náhled obrázku

Uživatel stiskne tlačítko a pomocí průzkumníka souborů si vybere obrázek, který se mu zobrazí v aplikaci. Tuto funkcionalitu jsem implementoval pomocí package *file_picker*. Kód komponenty, která zobrazí obrázek v aplikaci, vypadá následovně.

```
class ImagePreview extends StatelessWidget {
  /// Widget for picking Images with image preview.
  const ImagePreview(this.imgPath, {super.key});

  final String? imgPath;

  @override
  Widget build(BuildContext context) {
    double size = MediaQuery.of(context).size.width * 0.5;

    Container imageNotFound = Container(
      width: size,
      height: size,
      decoration: BoxDecoration(
        color: Theme.of(context).colorScheme.tertiaryContainer,
      ),
```

```
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            Icons.image_not_supported_rounded,
            color: Theme.of(context).colorScheme.tertiary,
            size: size / 4,
          ),
          const Text("err.imgNotFound").tr()
        ],
      ),
    );

    ClipRRect showImage(String path) {
      return ClipRRect(
        borderRadius: BorderRadius.circular(Styles.borderRadius),
        child: SizedBox.fromSize(
          child: Image.file(
            File(path),
            height: size,
            errorBuilder: (context, error, stackTrace) => imageNotFound,
          ),
        ),
      );
    }

    Container noImage = Container(
      width: size,
      height: size,
      decoration: BoxDecoration(
        color: Theme.of(context).colorScheme.tertiaryContainer,
        borderRadius: const BorderRadius.all(
          Radius.circular(Styles.borderRadius),
        ),
      ),
      child: Icon(
        Icons.landscape_rounded,
        color: Theme.of(context).colorScheme.tertiary,
        size: size / 2,
      ),
    );

    return imgPath != null ? showImage(imgPath!) : noImage;
  }
}
```

Pokud je cesta k obrázku platná a obrázek existuje tak je obrázek zobrazen v aplikaci. Pokud neuživatel ještě žádný soubor nevybral, tak je obrázek nahrazen čtvercem s ikonou. V případě, že by při nahrávání obrázku došlo k chybě, tak se ve čtverci objeví ikona přeškrtnutého obrázku a chybová hláška, že obrázek nebyl nalezen.



Obrázek 17. Komponenta s náhledem obrázku

5.4.1.2 Náhled souboru

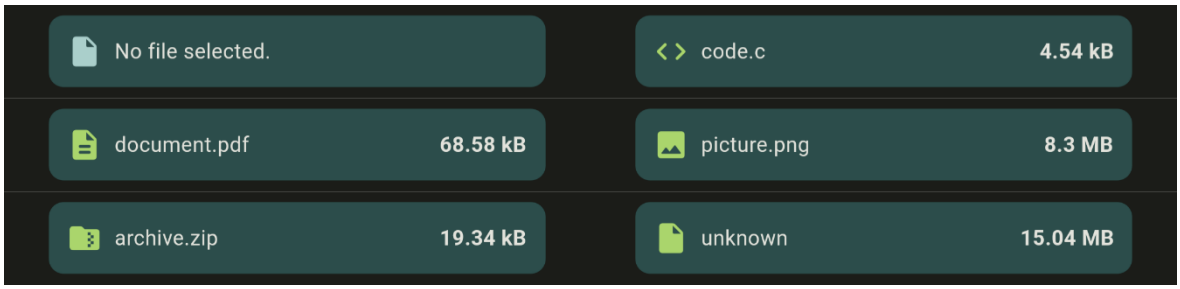
Uživatel si také bude muset vybrat soubor, do který bude chtít do obrázku ukrýt. O zobrazení základních informací o vybraném souboru jako název a jeho velikost se bude starat další komponenta. Soubor vybere ze zařízení opět pomocí package *file_picker*.

```
class FilePreview extends StatefulWidget {
  /// Widget used for displaying basic info about selected file
  const FilePreview(this.filePath, {super.key});
  final String? filePath;

  @override
  State<FilePreview> createState() => _FilePreviewState();
}

class _FilePreviewState extends State<FilePreview> {
  // Field is used for re-rendering if language is changed
  // ignore: unused_field
  Locale _currentLocale = SupportedLocales.english;

  @override
  Widget build(BuildContext context) {
    _currentLocale = context.locale;

    String fileName = tr('err.noFile');
    String fileSize = '';
    Icon fileIcon = Icon(Icons.insert_drive_file_rounded,
        color: Theme.of(context).colorScheme.tertiary);
```

```
if (widget.filePath != null) {
  try {
    final File = File(widget.filePath!);

    fileName = file.uri.pathSegments.last;
    final int fileBytes = file.lengthSync();

    fileIcon = chooseIcon(getExtension(fileName), context);
    fileSize = countFileSize(fileBytes);
  } catch (e) {
    fileName = tr('err.fileNotFound');
    fileIcon = Icon(
      Icons.error_rounded,
      color: themeHolder.currentThemeMode() == ThemeMode.light
          ? Colors.red
          : Colors.redAccent,
    );
  }
}

return Container(
  decoration: BoxDecoration(
    color: Theme.of(context).colorScheme.tertiaryContainer,
    borderRadius: const BorderRadius.all(
      Radius.circular(Styles.borderRadius),
    ),
  ),
  child: Padding(
    padding: const EdgeInsets.all(12),
    child: Row(
      children: [
        fileIcon,
        Expanded(
          child: Padding(
            padding: const EdgeInsets.fromLTRB(8, 0, 8, 0),
            child: Text(fileName),
          ),
        ),
        Text(
          fileSize,
          style: TextStyle(
              fontFamily: GoogleFonts.robotoCondensed().fontFamily,
              fontWeight: FontWeight.bold),
        ),
      ],
    ),
  ),
);
```

```
  }
}
```

Velikost souboru si přepočítám pomocí vlastní metody `countFileSize(fileBytes)`, která bere velikost souboru v bytech a přepočítává je na čitelnější podobu jako kB či MB. Komponenta rovněž mění ikonu u názvu souboru podle typu souboru, který určí podle přípony.



Obrázek 18. Komponenta s náhledem souboru

5.4.2 Hlavní obrazovka

Hlavní obrazovka celé aplikace je to, kde se skrývá primární funkcionalita aplikace. Tvoří ji dva formuláře, mezi kterými lze přepínat spodním navigačním panelem. Jeden slouží k vybrání krycího obrázku, tajného souboru a zadání šifrovacího klíče, ten druhý zase k vybrání stego obrázku a zadání klíče k dešifrování. Kód takového formuláře k ukrytí dat vypadá takto.

```
class HideFileForm extends StatefulWidget {
  /// Form used for hiding files into images.
  const HideFileForm({super.key});

  @override
  State<HideFileForm> createState() => _HideFileFormState();
}

class _HideFileFormState extends State<HideFileForm> {
  // Field is used for re-rendering if language is changed
  // ignore: unused_field
  Locale _currentLocale = SupportedLocales.english;

  HideEnvelope _envelope = mainPageStatesHolder.hideEnvelope;
  final TextEditingController _keyController = TextEditingController();
  bool _validating = false;
  String? _errorMessage;
```

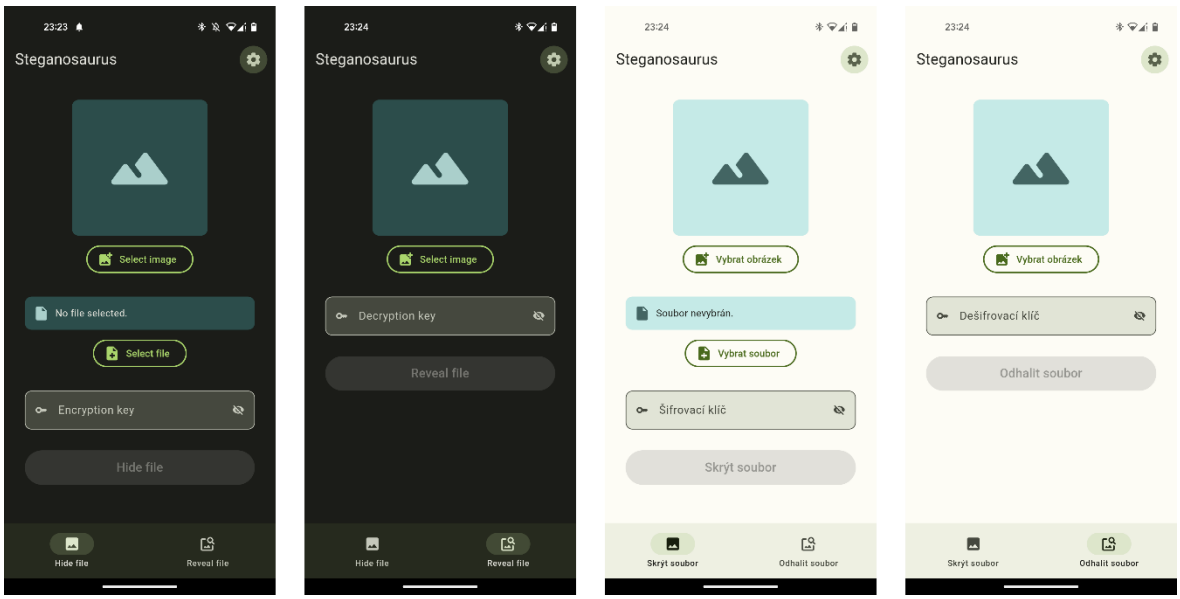
```
@override
void initState() {
  _keyController.text = _envelope.encryptKey ?? '';
  super.initState();
}

@override
Widget build(BuildContext context) {
  _currentLocale = context.locale;
  _envelope = mainPageStatesHolder.hideEnvelope;
  return ScrollableWrapper([
    ImagePreview(_envelope.imgPath),
    const ColumnSpacer(Styles.smallGap),
    SelectButton(
      onPressed: () {
        pickImgPath(_envelope.imgPath).then((path) => setState(() => {
              _envelope.imgPath = path,
              _errorMessage = null,
            }));
      },
      icon: const Icon(Icons.add_photo_alternate_rounded),
      label: _envelope.imgPath != null
          ? const Text('btn.changeImg').tr()
          : const Text('btn.selectImg').tr(),
    ),
    const ColumnSpacer(Styles.bigGap),
    FilePreview(_envelope.filePath),
    const ColumnSpacer(Styles.smallGap),
    SelectButton(
      onPressed: () {
        pickFilePath(_envelope.filePath).then((path) => setState(() => {
              _envelope.filePath = path,
              _errorMessage = null,
            }));
      },
      icon: const Icon(Icons.note_add),
      label: _envelope.filePath != null
          ? const Text('btn.changeFile').tr()
          : const Text('btn.selectFile').tr(),
    ),
    const ColumnSpacer(Styles.bigGap),
    KeyField(
      icon: const Icon(Icons.key_rounded),
      onChange: (value) {
        setState(() {
          _envelope.encryptKey = value;
        });
      },
```

```
        label: tr('lbl.encryptKey'),
        hintText: tr('lbl.encryptKeyHint'),
        controller: _keyController,
      ),
      const ColumnSpacer(Styles.bigGap),
      ConfirmButton(
        label: tr('btn.hideFile'),
        onPressed: _envelope.areFilesSelected() && !_validating
            ? () => setState(() {
                  _validating = true;
                  _envelope.validate().then((result) => {
                        if (result.isValid)
                          {
                            _errorMessage = null,
                            Navigator.push(
                              context,
                              MaterialPageRoute(
                                  builder: (context) => const HidingPage()),
                            ),
                          }
                        else
                          {
                            _errorMessage = result.message,
                          }
                      });
                  _validating = false;
                })
            : null,
      ),
      const ColumnSpacer(Styles.smallGap),
      if (_errorMessage != null) ...[ErrorMessage(_errorMessage!)]
    ]);
  }
}
```

Cesty k souborům a šifrovací klíč jsou drženy v datové struktuře `HideEnvelope`. Ta při stisku tlačítka zavolá metodu `validate()`, která zkontroluje, zda zvolené soubory existují či zda nebyla překročena velikost informace, kterou lze skrýt do obrázku. Pokud ano, tak zobrazí chybovou hlášku pod tlačítkem a proces ukrývání nespustí. Pokud je vše v pořádku, tak je uživatel přesměrován na obrazovku s loadingem. Aplikace mezitím na pozadí nechává běžet steganografický algoritmus. Po jeho dokončení je uživatel přesměrován na obrazovku s výsledkem operace, která může skončit buď úspěšně, nebo nějakou chybou, která je uživateli oznámena v chybové hlášce.

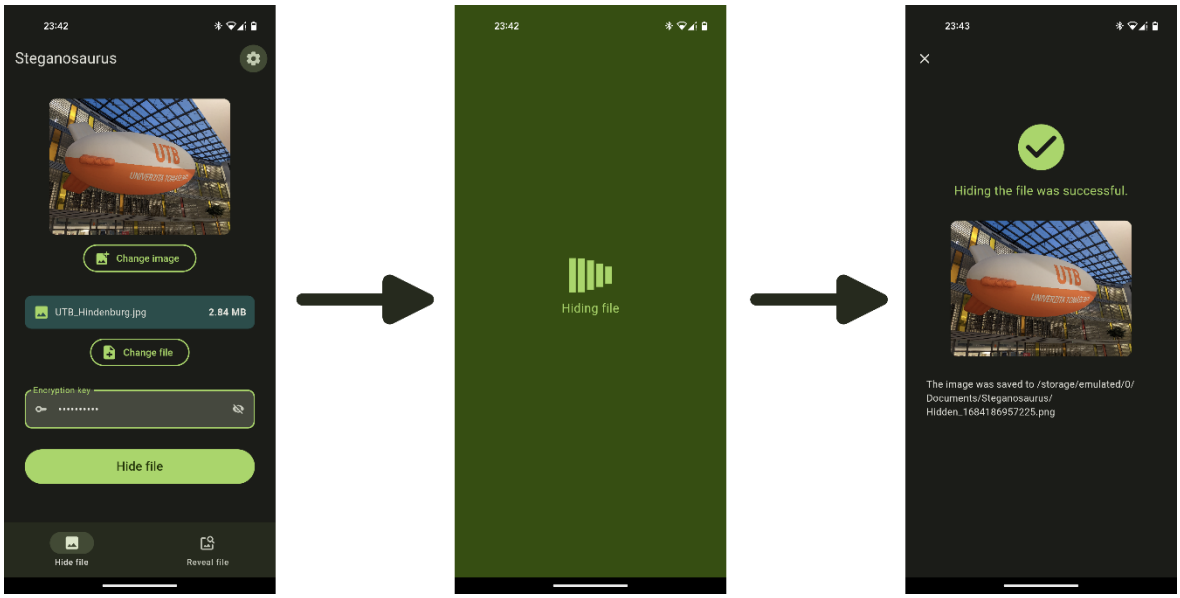
Formulář k odhalení souboru je řešen podobným způsobem. Chybí možnost vybrat soubor a cesta k stego obrázku a dešifrovací klíč jsou drženy v datové struktuře `RevealEnvelope`.

Oba formuláře jsou lokalizované jak do češtiny tak angličtiny a podporují tmavý i světlý motiv.



Obrázek 19. Hlavní obrazovka aplikace

Po zadání platných vstupních souborů a stisknutí tlačítka pro ukrytí souboru vypadá posloupnost obrazovek následovně.



Obrázek 20. Posloupnost obrazovek při ukrývání souboru

Oproti původnímu wireframu jsem na poslední obrazovku s výsledkem nepřidal tlačítko na stáhnutí obrázku, ale aplikace jej automaticky uloží do vlastní složky a uživateli pouze oznámí polohu souboru v úložišti. Na hlavní obrazovku se uživatel vrátí stisknutím křížku v levém horním rohu.

5.5 Steganografický algoritmus

Nejdůležitější částí celé aplikace je samotný steganografický algoritmus. Ten jsem rozdělil do dvou samostatných controllerů. Jeden slouží ke schování souboru a druhý k jeho odhalení. Oba tyto conrollery však potřebují ke své činnosti klíč.

5.5.1 Klíč

Steganografický klíč jsem implementoval do samostatné třídy StegoKey. Ta ve svém konstruktoru vezme textový řetězec zadaný uživatelem a předělá jej na svou instanci podle pravidel popsaných v předchozí kapitole.

```
class StegoKey {
  int offset = 0;
  int startChannel = 0;
  List<int> bits = [0];

  /// Creates Key object from key string
  StegoKey(String? stringKey, int imgWeight, int imgHeight) {
    if (stringKey == null || stringKey.isEmpty) {
      return;
    }

    List<int> asciiValues = [];
    for (int i = 0; i < stringKey.length; i++) {
      asciiValues.add(stringKey.codeUnitAt(i));
    }

    List<int> tmpBits = [];
    for (int i = 0; i < asciiValues.length; i++) {
      tmpBits.addAll(_decToBin(asciiValues.elementAt(i)));
    }

    int oneBits = tmpBits.reduce((x, y) => x + y);
    int asciiSum = asciiValues.reduce((x, y) => x + y);

    offset = (pow(asciiSum, 2).toInt() * oneBits) % (imgWeight * imgHeight);
    startChannel = stringKey.length % 3;
```

```
    bits = tmpBits;
  }
```

Pomocnou primární funkcí `_decToBin(int)` se převádí ASCII hodnota znaku na seznam bitů o délce 8.

5.5.2 Ukrývání souboru

Metoda na ukrytí souboru se nachází v controlleru *HidingController*. Jednotlivé kroky jako nahrání souborů, vytvoření klíče, kontrola počtu kanálů a samotný algoritmus probíhají samostatně a pokud je v jakémkoliv z těchto kroků vyhozena výjimka, tak je odchycena a operace je vyhodnocena jako neúspěšná a společně s chybovou hláškou poslána dál. Pokud vše proběhne úspěšně, tak je vyhodnocena jako úspěšná a společně s cestou ke stego obrázku vrácena ve třídě *ProcessingResult*. Metoda také používá package *image*, která umožňuje načíst a uložit obrázek a také v něm přistupovat k jednotlivým pixelům obrázku, číst a měnit jejich hodnotu.

```
Future<ProcessingResult> _processHiding(HideEnvelope envelope) async {
    Image;
    int w, h;
    try {
      image = (await decodeImageFile(envelope.imgPath!))!;
      w = image.width;
      h = image.height;
    } catch (e) {
      return ProcessingResult.fail('err.imgLoadFail');
    }

    if (image.numChannels < 3) {
      return ProcessingResult.fail('err.only3chan');
    }

    Uint8List fileBytes;
    try {
      fileBytes = File(envelope.filePath!).readAsBytesSync();
    } catch (e) {
      return ProcessingResult.fail('err.fileLoadFail');
    }

    StegoKey key;
    try {
      key = StegoKey(envelope.encryptKey, w, h);
    } catch (e) {
      return ProcessingResult.fail('err.keyFail');
    }
```

```
final int fileSize = fileBytes.length;
final double imgCapacity = ((w * h * 3) - 160) / 8;
if (fileSize > imgCapacity) {
  return ProcessingResult.fail('err.fileTooLarge');
}

try {
  String? extension = _get5Extension(envelope.filePath!);
  if (extension == null) {
    return ProcessingResult.fail('err.extTooLong');
  }

  Uint8List uint8Ext = _stringToUint8List(extension);
  String fileSizeStr = fileSize.toString().padLeft(10, ' ');
  Uint8List uint8Size = _stringToUint8List(fileSizeStr);
  BytesBuilder builder = BytesBuilder();
  builder.add(uint8Ext);
  builder.add(uint8Size);
  builder.add(fileBytes);
  Uint8List secretBytes = builder.toBytes();

  // Main algorithm
  int pos = key.offset;
  int pixels = w * h;
  int shift = (pixels / 13).floor();
  int channel = key.startChannel;
  int keyIndex = 0;
  int loop = 0;
  int cycle = 1;
  int writtenInCycle = 0;
  int stop = pixels - (pixels % 13);
  for (int i = 0; i < secretBytes.length; i++) {
    List<int> bits = _byteToBits(secretBytes.elementAt(i));
    for (int j = 0; j < 8; j++) {
      if (writtenInCycle % 13 == 0 && writtenInCycle != 0) {
        loop++;
        pos = key.offset + loop;
      }

      int secretBit = (bits[j] + key.bits[keyIndex]) % 2;
      _hideBit((pos % w), (pos / w).floor(), channel, secretBit, image);
      writtenInCycle++;
      channel = (channel++) % 3;
      pos = (pos + shift) % pixels;
      keyIndex = keyIndex++ % key.bits.length;

      if (writtenInCycle == stop) {
```

```
            writtenInCycle = 0;
            pos = key.offset;
            channel = (key.startChannel + cycle) % 3;
            loop = 0;
            cycle++;
            if (cycle == 4) {
              return ProcessingResult.fail('err.bigFail');
            }
          }
        }
      }
    } catch (e) {
      return ProcessingResult.fail('err.bigFail');
    }

    // Writing image to folder.
    try {
      await encodePngFile(envelope.resultPath!, image);
    } catch (e) {
      return ProcessingResult.fail('err.imgSaveFail');
    }

    return ProcessingResult.succeed(envelope.resultPath!);
  }
```

Tajný soubor je načten jako seznam bytů a ty jsou pak bit po bitu uloženy do obrázku. Tento seznam ale neobsahuje všechny důležité informace jako například příponu souboru. Proto příponu převedu na ASCII hodnoty a uložím do prvních pěti bytů seznamu. Z tohoto důvodu aplikace nepodporuje ukrývání souborů s delší příponou než 5 znaků. Do dalších 10 bytů si uložím velikost souboru v bytech, takže bude algoritmus při odhalování souboru vědět, kdy má přestat číst tajná data z obrázku.

I přesto, že se algoritmus spustí asynchronně, tak způsobí výrazné zaseknutí uživatelského rozhraní, protože je procesor vytížen naplno zapisováním dat do obrázku. Z toho důvodu je nutné tuto metodu spustit ve funkci `compute`, Která spustí proces izolovaně, takže k zaseknutí nedojde.

```
  Future<ProcessingResult> hideIntoImage(HideEnvelope envelope) async {
    return await compute(_processHiding, envelope);
  }
```

5.5.3 Odhalování souboru

Algoritmus pro odhalení souboru z obrázku funguje obdobně, ale místo zapisování tajných dat, tajná data čte. Algoritmus nejdříve přečte prvních 5 bytů, kde je ukrytá přípona a uloží si ji na pozdější použití. Poté přečte dalších 10, kde je uložena velikost souboru. Pokud velikost na převedení zpět z ASCII hodnot nevytvoří smysluplné číslo ale směs náhodných znaků, tak obrázek buď tajná data neobsahuje, nebo je zadán nesprávný klíč. V tomto případě algoritmus ukončí svou činnost předčasně a oznámí to uživateli.

```
Future<ProcessingResult> _processRevealing(RevealEnvelope envelope) async {
    Image;
    int w, h;
    try {
      image = (await decodeImageFile(envelope.imgPath!))!;
      w = image.width;
      h = image.height;
    } catch (e) {
      return ProcessingResult.fail('err.imgLoadFail');
    }

    if (image.numChannels < 3) {
      return ProcessingResult.fail('err.only3chan');
    }

    StegoKey key;
    try {
      key = StegoKey(envelope.decryptKey, w, h);
    } catch (e) {
      return ProcessingResult.fail('err.keyFail');
    }

    List<int> bytesHolder = [];
    String extension = '';
    try {
      // Main algorithm
      int pos = key.offset;
      int pixels = w * h;
      int shift = (pixels / 13).floor();
      int channel = key.startChannel;
      int keyIndex = 0;
      int loop = 0;
      int cycle = 1;
      int readInCycle = 0;
      int stop = pixels - (pixels % 13);

      int endByte = 20;
```

```
      int readBytes = 0;
      while (readBytes < endByte) {
        List<int> bits = [];
        for (int j = 0; j < 8; j++) {
          if (readInCycle % 13 == 0 && readInCycle != 0) {
            loop++;
            pos = key.offset + loop;
          }

          int bit = _getBit((pos % w), (pos / w).floor(), channel, image);
          bits.add((bit + key.bits[keyIndex]) % 2);
          readInCycle++;
          channel = (channel++) % 3;
          pos = (pos + shift) % pixels;
          keyIndex = keyIndex++ % key.bits.length;

          if (readInCycle == stop) {
            readInCycle = 0;
            pos = key.offset;
            channel = (key.startChannel + cycle) % 3;
            loop = 0;
            cycle++;
            if (cycle == 4) {
              return ProcessingResult.fail('err.bigFail');
            }
          }
        }
        bytesHolder.add(_bitsToByte(bits));
        readBytes++;
        if (readBytes == 5) {
          extension = String.fromCharCodes(bytesHolder).trim();
          if (extension.isNotEmpty) {
            extension = '.$extension';
          }
          bytesHolder.clear();
        } else if (readBytes == 15) {
          try {
            int fileSize = int.parse(String.fromCharCodes(bytesHol-
der).trim());
            endByte = fileSize + 15;
            bytesHolder.clear();
          } catch (e) {
            return ProcessingResult.fail('err.revealFail');
          }
        }
      }
    } catch (e) {
      return ProcessingResult.fail('err.revealFail');
```

```
    }

    // Writing file to folder.
    try {
      Uint8List fileBytes = Uint8List.fromList(bytesHolder);
      File(envelope.resultPathNoExt! + extension).writeAsBytesSync(fileBy-
tes);
    } catch (e) {
      return ProcessingResult.fail('err.fileSaveFail');
    }

    return ProcessingResult.succeed(envelope.resultPathNoExt! + extension);
  }
```

Tato metoda se opět spustí ve funkci `compute`, aby nedocházelo k zamrznutí uživatelského rozhraní.

6 TESTOVÁNÍ MOBILNÍ APLIKACE

Poslední, ale neméně důležitou částí aplikace je její testování. To jsem rozdělil na 2 části. První z nich bude systémové testování aplikace, kdy na fyzickém zařízení s Androidem provedu manuální testy aplikace. Druhá část testování bude originální a stego obrázek analyzovat pomocí nástrojů na detekci steganografie a vyhodnotí výsledky.

6.1 Systémové testování

Na zařízení s nejnovější verzí Androidu 13 provedu několik testovacích scénářů. Poslední test budu provádět na starším telefonu s Androidem 8. Testovací případy jsou následující.

Tabulka 5. Přehled testovacích případů

ID	Název	Popis
TC01	Úspěšné skrytí	Aplikace úspěšně skryje vybraný soubor do obrázku a obrázek uloží do zařízení.
TC02	Úspěšné odhalení	Aplikace úspěšně odhalí soubor schovaný v obrázku po předchozím testu.
TC03	Špatný klíč	Aplikace neodhalí skrytý soubor při zadání špatného hesla
TC04	Příliš velký soubor	Aplikace při pokusu schování příliš velkého souboru zobrazí chybovou hlášku a akci neprovede
TC05	Starší OS	Aplikace bude fungovat i na starším zařízení s Androidem 8

6.1.1 TC01 – Úspěšné skrytí

- Testuje úspěšné skrytí vybraného souboru do obrázku.

Preconditions: Nainstalovaná aplikace na Androidu 13

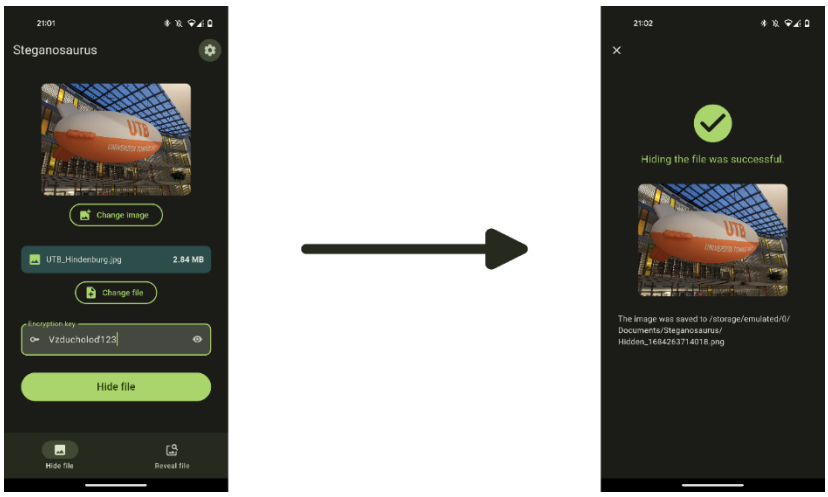
Postconditions: Žádné

Kroky:

1. Uživatel vybere ze zařízení obrázek a soubor k ukrytí
2. Uživatel zadá klíč k zašifrování „Vzducholoď123“
3. Uživatel klikne na tlačítko: Hide file
4. Aplikace schová soubor a výsledný obrázek uloží

5. Aplikace oznámí výsledek operace.

Očekávaný výsledek: Nový stego obrázek v zařízení



Obrázek 21. Průběh TC01 – Úspěšné skrytí

6.1.2 TC02 – Úspěšné odhalení

- Testuje odhalení souboru z obrázku ukrytém v předchozím testu.

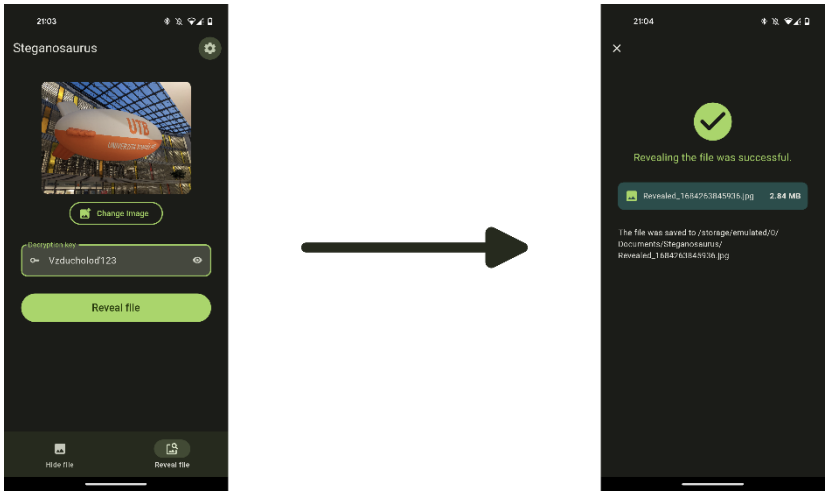
Preconditions: Nainstalovaná aplikace na Androidu 13

Postconditions: Žádné

Kroky:

1. Uživatel vybere obrázek se skrytým souborem ze zařízení
2. Uživatel zadá klíč k dešifrování: „Vzducholoď123“
3. Uživatel klikne na tlačítko: Reveal file
4. Aplikace odhalí soubor a uloží jej
5. Aplikace oznámí výsledek operace uživateli

Očekávaný výsledek: Nový odhalený soubor v zařízení



Obrázek 22. Průběh TC02 – Úspěšné odhalení

6.1.3 TC03 – Špatný klíč

- Testuje reakci aplikace po zadání špatného dešifrovacího klíče.

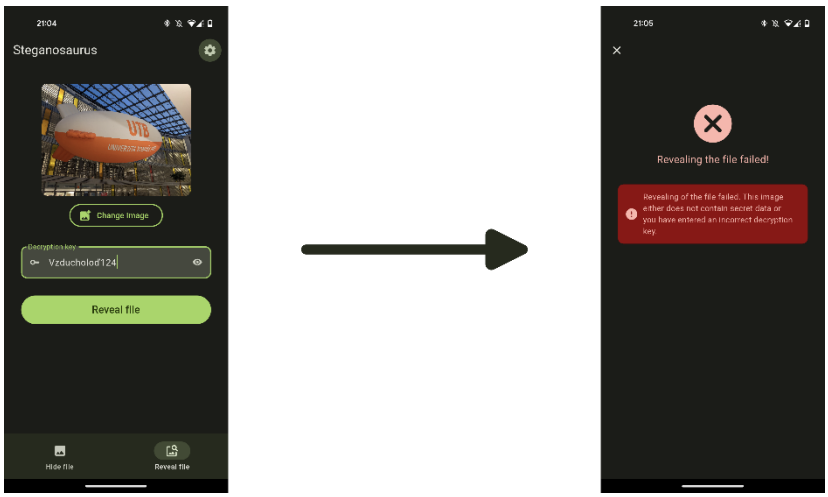
Preconditions: Nainstalovaná aplikace na Androidu 13

Postconditions: Žádné

Kroky:

1. Uživatel vybere obrázek se skrytým souborem ze zařízení
2. Uživatel zadá klíč k dešifrování: „Vzducholoď124“
3. Uživatel klikne na tlačítko: Reveal file
4. Aplikace oznámí, že při odhalování souboru došlo k chybě

Očekávaný výsledek: Chybová hláška v aplikaci



Obrázek 23. Průběh TC03 – Špatný klíč

6.1.4 TC04 – Příliš velký soubor

- Testuje reakci aplikace při vybrání příliš velkého souboru k ukrytí.

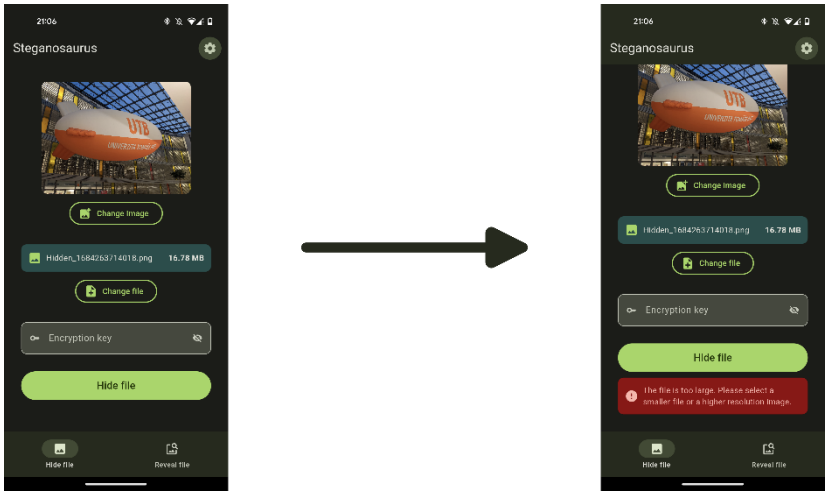
Preconditions: Nainstalovaná aplikace na Androidu 13

Postconditions: Žádné

Kroky:

1. Uživatel vybere obrázek a velký soubor ze zařízení
2. Uživatel klikne na tlačítko: Hide file
3. Aplikace oznámí, že soubor je příliš velký a nespustí proces ukrývání

Očekávaný výsledek: Chybová hláška v aplikaci



Obrázek 24. Průběh TC04 – Příliš velký soubor

6.1.5 TC05 – Starší OS

- Testuje úspěšné skrytí vybraného souboru do obrázku i na starším zařízení

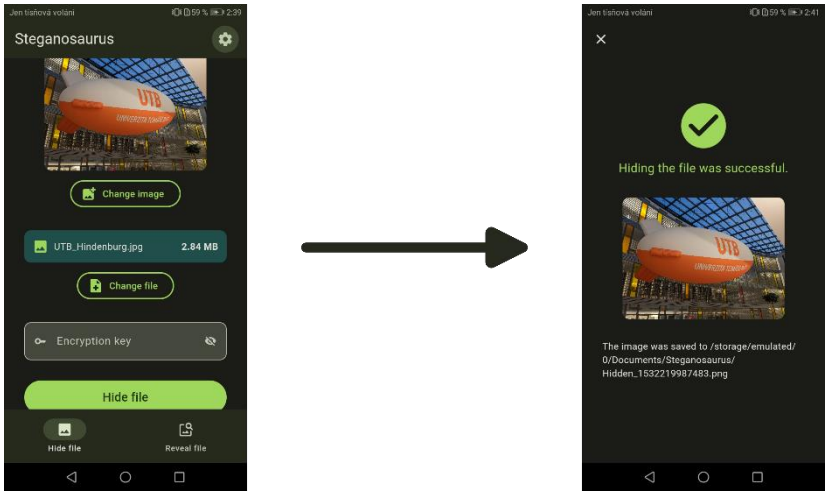
Preconditions: Nainstalovaná aplikace na Androidu 8

Postconditions: Žádné

Kroky:

1. Uživatel vybere ze zařízení obrázek a soubor k ukrytí
2. Uživatel klikne na tlačítko: Hide file
3. Aplikace schová soubor a výsledný obrázek uloží
4. Aplikace oznámí výsledek operace.

Očekávaný výsledek: Nový stego obrázek v zařízení



Obrázek 25. Průběh TC05 – Starší OS

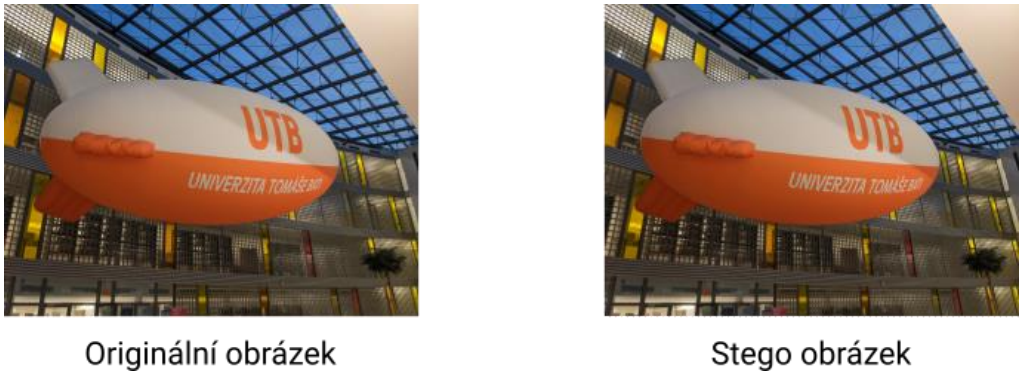
Všechny testy proběhly úspěšně. Aplikace ve všech případech reagovala podle očekávání a byla plně funkční i na starším zařízení.

6.2 Stegoanalýza

Otázkou ovšem zůstává, jestli je mnou navržený steganografický algoritmus odolný proti stegoanalýze. Na tuto otázku mi pomůže odpovědět software StegSpy, ten dokáže detekovat steganografické programy jako *Hiderman*, *JPHideandSeek*, *Masker*, *JPegX* a *Invisible Secrets* [26]. Před tím, než budu analýzu provádět, tak testovaný obrázek zmenším na velikost 280 x 210 a převedu na formát PNG, jelikož aplikace tento formát používá jako výstupní typ souboru. Ukryji do něj malý textový soubor o velikosti 872 B. K zašifrování jsem nepoužil žádný klíč

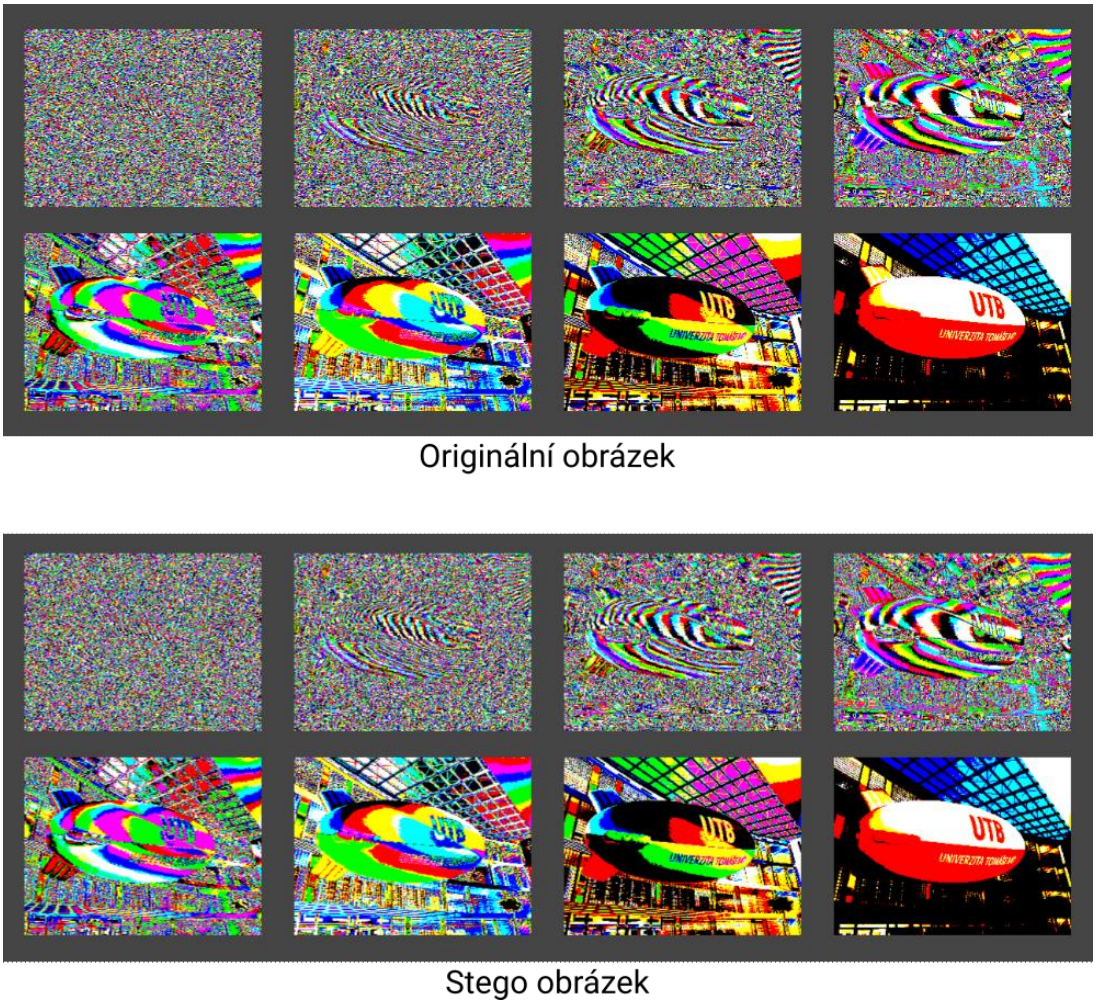
6.2.1 Porovnání originálního a stego obrázku

Před samotnou analýzou pomocí softwaru si ještě originální obrázek- porovnám s výstupem z aplikace. Oba obrázky jsou ve formátu PNG, mají rozlišení 280 x 210 a bitovou hloubku 32. Mírně se liší velikost souboru, původní obrázek má 123.4 kB, zatímco stego obrázek má velikost 125.9 kB. Také dpi se u jednotlivých obrázků liší. Originální má dpi 5, zatímco stego obrázek má dpi 96. To by mohlo vysvětlovat drobný rozdíl ve velikosti obrázku.



Obrázek 26. Porovnání originálního a stego obrázku

Na první pohled vypadají oba obrázky stejně, ačkoliv ten napravo obsahuje skrytý soubor. Oba obrázky jsem analyzoval ještě víc do hloubky. U každého bytu jsem zvýraznil dominantní barvu od LSB po MSB. Ani tady není vidět na první pohled znatelný rozdíl.



Obrázek 27. Porovnání dominantní barvy v každé bitové vrstvě

6.2.2 Analýza pomocí StegSpy

Nakonec obrázek analyzuji pomocí softwaru StegSpy. Tento software bere obrázek ze zařízení a snaží se v něm najít stopy po použití známých steganografických algoritmů.

U originálního obrázku je výstup z aplikace následující:

```
You Selected C:\Users\jbaslar\Desktop\airship.png to be opened. Sorry no Steg found.
```

U stego obrázku nám StegSpy zahlásí následující hlášku:

```
You Selected C:\Users\jbaslar\Desktop\airshipH.png to be opened. Sorry no Steg found.
```

Nejspíš proto, že se jedná o originální steganografický algoritmus, tak aplikace nebyla schopna nic detekovat.

ZÁVĚR

Cílem teoretické části této bakalářské práce bylo čtenáře seznámit se základní problematikou obrazové steganografie. Je zde probrána stručná historie steganografie a rovněž se zde zaměřuji na digitální steganografii a útoky na ni. Samostatnou kapitolou jsou pak techniky steganografie obrazu. Poslední část se pak věnuje steganografickým aplikacím na platformě Android a jejich vzájemnému porovnání.

Praktická část je pak zaměřena na návrh a implementaci steganografické aplikace na platformě Android. Aplikaci jsem se navrhl tak, aby nejen správně aplikovala nově navržený steganografický algoritmus, ale aby byla i uživatelsky přívětivá, moderní a snadno použitelná pro každého uživatele. Aplikace byla vytvořena v multiplatformním frameworku Flutter, avšak přizpůsobena primárně na platformu Android (ačkoliv je aplikace spustitelná a plně funkční i na platformě Windows). Poslední kapitola praktické části se věnuje testování samotné aplikace a na výstupních obrázcích provádí stegoanalýzu.

Výstupem z praktické části je aplikace pro platformu Android, která umožňuje ukrýt jakýkoliv binární soubor ze zařízení do obrázku formátu PNG. Také umožňuje tento soubor šifrovat klíčem, takže jej nemůže jiný uživatel aplikace odhalit bez znalosti klíče. Aplikace nabízí funkcionalitu, kterou v současné době nenabízí žádná jiná aplikace na platformě Android. Rovněž použití vlastního steganografického algoritmu stěžuje útok na výsledný stego obrázek pomocí stegoanalýzy.

SEZNAM POUŽITÉ LITERATURY

[1] WAYNER, Peter. Disappearing cryptography. 2nd ed. Amsterdam: MK/Morgan Kaufmann Publishers, c2002. ISBN 1558607692.

[2] SINGH, Simon, Dita ECKHARDTOVÁ a Petr KOUBSKÝ. Kniha kódů a šifer. 3. vyd. Praha: Dokořán, 2017. ISBN 9788073632687.

[3] VONDRUŠKA, Pavel a Bára BUCHALOVÁ. Kryptologie, šifrování a tajná písma. Praha: Albatros, 2006. ISBN 8000018888.

[4] SLUIJS, Peter van der. Table with was and stylus Roman times. In: Wikimedia Commons [online]. -: Wikimedia Foundation, 2012 [cit. 2023-03-30]. Dostupné z: https://commons.wikimedia.org/wiki/File:Table_with_was_and_stylus_Roman_times.jpg

[5] PETITCOLAS, Fabien A. P. a Stefan KATZENBEISSER. Information hiding techniques for steganography and digital watermarking. Boston: Artech House, 2000. ISBN 1580530354.

[6] WU, Min a Bede LIU. Multimedia data hiding. New York: Springer, c2003. ISBN 9780387954264.

[7] JAGETIYA, Anurag a C Rama KRISHNA. Digital Image Steganography: "Seeing is always NOT believing". The Journal of the Computer Society of India [online]. 2014, 14 - 17 [cit. 2023-04-01]. Dostupné z: https://www.researchgate.net/publication/313678274_Digital_Image_Steganography

[8] CHOUDARY, Archana. Steganography Tutorial – A Complete Guide For Beginners. Edureka [online]. -: Brain4ce Education Solutions Pvt., 2023 [cit. 2023-04-02]. Dostupné z: https://www.edureka.co/blog/steganography-tutorial

[9] MORKEL, Tayana. Image steganography applications for secure communication [online]. Pretoria, 2012 [cit. 2023-04-10]. Dostupné z: https://repository.up.ac.za/handle/2263/29906. Dissertation. University of Pretoria.

[10] ABDELHAMID, Awad Attaby, F. M. MONA, Ahmed MURSI a K. Alsammak ABDELWAHAB. Data hiding inside JPEG images with high resistance to steganalysis using a novel technique: DCT-M3. Ain Shams Engineering Journal [online]. 2018, 9(4), 1965-1974 [cit. 2023-04-12]. ISSN 2090-4479. Dostupné z: https://www.sciencedirect.com/science/article/pii/S209044791730031X

[11] What is MSB and LSB?. Morningstar [online]. Morningstar Engineering, 2021 [cit. 2023-04-13]. Dostupné z: https://www.morningstar.io/post/2016/12/25/midi-msb-and-lsb

[12] Jak je to s barvami v obrázcích. Aldebaran [online]. 2023 [cit. 2023-04-13]. Dostupné z: https://www.aldebaran.cz/onlineskola/etapy/grafika/barvy.htm

[13] Colors of the Year. W3Schools [online]. Refsnes Data AS, 2021 [cit. 2023-04-16]. Dostupné z: https://www.w3schools.com/colors/colors_trends.asp

[14] SMIL, Vaclav. Numbers don't lie: 71 things you need to know about the world. UK: Viking, Penguin Books, 2020. ISBN 978-0241454411.

[15] SCHENKER, Marc. What Is the Difference Between JPG, PNG, BMP, and TIFF Images?. Brush up [online]. Seattle: Creative Market, 2021 [cit. 2023-04-19]. Dostupné z: https://creativemarket.com/blog/difference-between-jpg-png-bmp-tiff-images

[16] ACHARYA, Tinku a Ping-Sing TSAI. JPEG2000 standard for image compression: concepts, algorithms and VLSI architectures. New Jersey: John Wiley & Sons (Wiley), 2005. ISBN 9780471484226.

[17] Steganography. In: Google Play [online]. Google, 2016 [cit. 2023-04-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.akseltorgard.steganography&hl=cs&gl=US

[18] NoClue: Steganography App. In: Google Play [online]. Google, 2020 [cit. 2023-04-25]. Dostupné z: https://play.google.com/store/apps/details?id=hamza.app.steganography&hl=cs&gl=US

[19] Stephanie: Steganography App. In: Google Play [online]. Google, 2022 [cit. 2023-04-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.piekarskipiotr.stephanie&hl=cs&gl=US

[20] Image Steganography. In: Google Play [online]. -: Google, 2019 [cit. 2023-04-25]. Dostupné z: https://play.google.com/store/apps/details?id=com.pulkit.imagesteganography&hl=cs&gl=US

[21] PENCIL PROJECT: An open-source GUI prototyping tool that's available for ALL platforms. [online]. Evolus, 2023 [cit. 2023-05-03]. Dostupné z: https://pencil.evolus.vn

[22] WALTERS, Bob a Tess KISSINGER. Stegosaurus ungulates. In: National Park Service [online]. U.S. Department of the Interior, 2023, 2020 [cit. 2023-05-03]. Dostupné z: https://www.nps.gov/dino/learn/nature/stegosaurus-ungulates.htm

[23] NURIK, Roman. IconKitchen [online]. 2023 [cit. 2023-05-03]. Dostupné z: https://icon.kitchen

[24] Flutter [online]. Google, 2023 [cit. 2023-05-14]. Dostupné z: https://flutter.dev/

[25] Material 3 [online]. Google, 2023 [cit. 2023-05-14]. Dostupné z: https://m3.material.io/

[26] StegSpy. Spy Hunter [online]. SpyHunter, 2009 [cit. 2023-05-17]. Dostupné z: http://www.spy-hunter.com/stegspy

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASCII American Standard Code for Information Interchange

BMP Microsoft Windows Bitmap

DCT Discrete Cosine Transform

DFT Discrete Fourier Transform

DWT Discrete Wavelet Transform

ICMP Internet Control Message Protocol

IP Internet Protocol

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation

LSB Least Significant Bit

MSB Most Significant Bit

PNG Portable Network Graphics

RGB Red Green Blue

TCP Transmission Control Protocol

UDP User Datagram Protocol

XOR eXclusive OR

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura přiloženého flashdisku

- Soubor **app-release.apk** – Instalační soubor android aplikace
- Adresář **steganosaurus** – Projekt se zdrojovým kódem k aplikaci