Vývoj open-source aplikace pro správu hesel

Tomáš Adámek

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:	**Tomáš Adámek**
Osobní číslo:	**A20408**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Vývoj open-source aplikace pro správu hesel**
Téma práce anglicky:	**Development of an Open-Source Password Management Application**

Zásady pro vypracování

1. Nastudujte a popište problematiku open-source vývoje.
2. Rozepište možnosti zabezpečení přístupu k datům a jejich bezpečné ukládání.
3. Vyberte a popište vhodné technologie pro implementaci vlastní aplikace.
4. Navrhněte a implementujte desktopovou aplikaci pro správu hesel.
5. Implementované řešení řádně otestujte a popište výsledky.

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

Seznam doporučené literatury:

1. FOGEL, Karl. Tvorba open source softwaru: jak řídit úspěšný projekt svobodného softwaru. Praha: CZ.NIC, [2012]. CZ.NIC. ISBN 978-80-904248-5-2.
2. BURDA, Karel. Aplikovaná kryptografie. Brno: VUTIUM, 2013. ISBN 978-80-214-4612-0.
3. VIRIUS, Miroslav. Programování v C: od základů k profesionálnímu použití. Praha: Grada Publishing, 2018. Myslíme v.. ISBN 978-80-271-0502-1.
4. VIRIUS, Miroslav. Pokročilé C. Praha: České vysoké učení technické v Praze, 2022. ISBN 978-80-01-06951-6.
5. PRATA, Stephen. Mistrovství v C. 4., aktualiz. vyd. Brno: Computer Press, 2013. Bestseller (Computer Press). ISBN 978--80-251-3828-1.
6. PEČIVA, Jan. Seriál Tutoriál Vulkan [online]. Praha: Internet Info, 2021 [cit. 2022-10-11]. ISSN 1212-8309. Dostupné z: https://www.root.cz/serialy/tutorial-vulkan/

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**

doc. Ing. Jiří Vojtěšek, Ph.D.
děkan

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

Ve Zlíně dne 7. prosince 2022

Jméno, příjmení: Tomáš Adámek

Název bakalářské práce: Vývoj open-source aplikace pro správu hesel

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.

V e Zlíně, dne ……..

podpis studenta

ABSTRAKT

Tato diplomová práce se zabývá vývojem open-source aplikace pro správu hesel. Základním cílem práce bylo vytvořit desktopovou aplikaci, která by byla bezpečná, spolehlivá a jednoduše použitelná pro uživatele operačního systému Linux.

V teoretické části byly prozkoumány základy open-source vývoje, problematika bezpečnosti dat a potenciální technologie pro implementaci aplikace. Byly diskutovány různé přístupy k zabezpečení dat, včetně šifrování, a představeny technologie, které byly využity při vývoji aplikace, jako je programovací jazyk C++23. Dále byl detailně popsán použitý šifrovací a dešifrovací proces, včetně jeho klíčových prvků a způsobu jejich aplikace.

V praktické části byla navržena a implementována aplikace pro správu hesel. Aplikace byla pečlivě otestována a byly popsány její klíčové funkce, jako je jádro aplikace a grafické uživatelské prostředí.

Klíčová slova: Open-source, C++, Vývoj aplikace, Správce hesel

ABSTRACT

This thesis deals with the development of an open-source password management application. The primary goal of the work was to create a desktop application that would be secure, reliable and easy to use for Linux operating system users.

The theoretical part explored the basics of open-source development, data security issues and potential technologies for implementing the application. Different approaches to data security, including encryption, were discussed and the technologies that were used in the development of the application, such as the C++23 programming language, were introduced. Furthermore, the encryption and decryption process used was described in detail, including its key elements and how they were applied.

In the practical part, a password management application was designed and implemented. The application was thoroughly tested and its key features such as the application kernel and graphical user interface were described.

Keywords: Open-source, C++, Application development, Password manager

Touto cestou bych chtěl moc poděkovat zejména vedoucímu mé diplomové práce Ing. Petru Žáčkovi, Ph.D. za cenné rady a konzultace ohledně diplomové práce. Dále bych rád poděkoval webové stránce app.typopo.org za poskytované služby v oblasti typografických úprav.

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

ÚVOD

Heslo, jakožto utajovaný řetězec znaků umožňující uživateli prokázat svou identitu, příslušnost či pověření, se v lidské historii používá již po staletí. První počítačová hesla byla vytvořena v počátcích 60. let 20. století na systému CTSS, kde byla uložena v souboru jako prostý text. V roce 1962 došlo k úniku všech hesel v systému, což mnohé inspirovalo ke snaze o bezpečnější ukládání hesel. V roce 1974 přešel Unix na ukládání hesel ve formě hashů.[1][2][3]

Dobré heslo by mělo kombinovat malá a velká písmena, číslice a ideálně i speciální znaky. Optimalizovaná délka činí minimálně osm znaků a znaky by měly být v hesle náhodně rozloženy. Rovněž není vhodné používat identické heslo pro přístup k různým službám. Navzdory těmto zásadám se setkáváme s webovými službami, které uchovávají hesla v podobě prostého textu nebo zasílají hesla emailem, jak je to například případ služby https://archive.org ke dni 2023-03-23.

Uživatelé internetu často využívají mnoho účtů, jež vyžadují přihlášení, někdy i v řádu desítek. Pamatovat si takové množství hesel a měnit je několikrát do roka není v lidských silách. Uživatelé mají následující možnosti:

1. Neuposlechnout zásady pro dobrá hesla.
2. Svěřit svá hesla aplikaci s uzavřeným kódem (např. Google).
3. Zapisovat si hesla na fyzické médium (např. papír).
4. Svěřit svá hesla aplikacím s otevřeným kódem, jež využívají webové rozhraní.

Každá z těchto možností má svá rizika, a proto je téma této práce zaměřeno na správce hesel s otevřeným zdrojovým kódem. Takový správce umožňuje dostatečně zdatným uživatelům ověřit bezpečnost svých hesel a zkontrolovat, že nejsou odesílána na nežádoucí místa.

I. TEORETICKÁ ČÁST

1. PROBLEMATIKA OPEN-SOURCE

Pod názvy open-source a svobodný software (free software) se skrývá poměrně složitá problematika, jenž si zasluhuje větší pozornost. Vedení úspěšného open-source projektu není jen o tom mít dobrý software. Úspěšný open-source musí vytvořit stabilní a zapálenou komunitu, která bude ochotná věnovat svůj čas a znalosti pro dobro projektu a společenské uznání. Zároveň takovýto software není pouze o samotném softwaru, ale také o filozofii. Aby bylo možné nahlédnout do toho proč svobodný software vzniká, je třeba se ohlédnout až do doby, kdy počítače mezi sebou nebyli kompatibilní a lidé se stávali odborníky pouze na jednotlivé architektury.

1. 1. Historie

V šedesátých letech dvacátého století se zrodila první snaha společnosti IBM o koncipování licencování distribuovaného softwaru, který byl dodáván spolu s hardwarovými komponentami. Ambice IBM spočívala v premisách, že zákazník by nabyl pouze licenci programového vybavení a ne samotný program. Tyto úsilí se staly základem pro současný způsob licencování softwaru, avšak do doby, kdy hardware stále nebyl obecně dostupný, byly podmínky užití velice přívětivé. Uživatelům bylo dovoleno vytváření zálohových kopií, reverzní inženýrství pro porozumění fungování, modifikace a tvorba nových verzí programů.[4][5][6]

Nicméně, situace se začala proměňovat, když se počítače stávaly dostupnějšími a svět se propojoval prostřednictvím internetu. S nástupem unifikované architektury začali výrobci hardwaru ztrácet své výhody, které spočívaly v poskytnutí možnosti uživatelům modifikovat a sdílet kód, neboť tímto způsobem již nezískávali konkurenční převahu.

1. 1. 1. GNU

Proměna situace ve sféře softwarového průmyslu se nelíbila charismatickému Richardu Stallmanovi, jenž byl pevně přesvědčen, že software by měl zůstat svobodným. Tuto svobodu Stallman konkretizuje ve čtyřech základních pravidlech, jejichž indexace začíná od nuly.

Svoboda 0: Svoboda spouštět program podle vlastního rozhodnutí a za libovolným účelem.
Svoboda 1: Svoboda prostudovat, jak program operuje, a upravit jej tak, aby prováděl výpočty dle vašeho přání.
Svoboda 2: Svoboda šířit kopie, abyste mohli pomáhat ostatním.
Svoboda 3: Svoboda šířit kopie svých upravených verzí ostatním. Tím můžete dát celé komunitě možnost těžit z vašich změn. Předpokladem je přístup ke zdrojovému kódu.

Projekt GNU je vybudován na těchto svobodách. V kontextu, kde, jak již dobře víme, se software stal licencovaným zbožím, bylo nezbytné, aby GNU přišla s vlastní licencí, která tyto svobody zajišťuje. [7]

GNU usilovně směřovalo k sestavení kompletního souboru softwarových nástrojů, který by spadal pod licenci GNU Public License. Všechny komponenty již měli připraveny, avšak jediný chybějící prvek představovalo jádro operačního systému. Zde se však vývojáři GNU vydali na klikatou stezku, na níž je ve finále předstihl finský student Linus Torvalds, který vytvořil monolitické jádro, jenž později dostalo název Linux. [4]

1. 1. 2. Free versus open-source

GNU již od svého zrodu prezentuje svůj software jako svobodný software. Avšak problémem tohoto přístupu je, že v anglickém jazyce slovo „free“ znamená nejen svobodný, ale též zdarma. Tato shoda významů zejména v minulosti často způsobovala značná nedorozumění, neboť v rámci firem a korporací slovo „free“ evokovalo něco bezplatného a tudíž nekvalitního. Z tohoto důvodu vzešla v roce 1998 iniciativa OSI (The Open Source Initiative). Tato iniciativa se snaží svobodnému softwaru nadělit marketingový kabátek, který by posílil důvěru v něj. Jako jeden z nástrojů ke splnění tohoto cíle byl zvolen název „open-source“ místo „free software“, čímž se zabránilo chybné interpretaci, aniž by se dotkla samotné podstaty myšlenky.[4]

1. 2. Licence

Kromě již zmíněné licence GPL pro open-source existuje ohromující množství dalších licencí. V této kapitole se zaměříme na názvosloví a další běžné licence, přičemž tato část textu čerpá zejména z výkladu na stránce root.cz ze série článků věnovaných licencím v oblasti svobodného softwaru[8].

- Public domain – předpoklad volného užití pro všechny, je vyňato z ochrany autorských práv.
- Copyleft – slouží k zachování svobodné licence.
- Freeware – bezúplatně použitelný software bez časového omezení podléhající autorským právům, není free software ani open-source
- Shareware – umožňuje za podmínek uvedených v licenčním ustanovení další šíření, zpravidla je dodáván pouze jako spustitelný soubor. Příklad takového software je bezplatná znehodnocená verze placeného software.
- Proprietární software – software, u něhož není dostupný kód, je zakázáno modifikovat jej či používat reverzní inženýrství k zjištění fungování.

Následující přehled nabízí zjednodušený pohled na práva, povinnosti, stejně jako na aspekty, které daná licence nezaručuje či nepovoluje. Uvedený souhrn je sestaven na základě několika odborných zdrojů. Nicméně, je třeba zdůraznit, že autor tohoto textu není držitelem právnického vzdělání. V důsledku toho tato tabulka nemůže být považována za úplný, nezměnitelný právní dokument. Je to spíše srozumitelný přehled, který má za cíl poskytnout čtenáři základní orientaci v problematice licencí. Vždy je třeba konzultovat s kvalifikovaným právníkem nebo se obrátit na relevantní instituci pro získání přesných a závazných informací.[8][9][10][11]

Licence	Zkratka	Povoluje	Nařizuje	Nezaručuje / Nepovoluje
GNU General Public License version 2	GPL v2	Komerční využití Distribuci Modifikace	Zveřejnění zdrojového kódu Udržení stejné licence	Zodpovědnost Záruky
GNU General Public License version 3	GPL v3	Komerční využití Distribuci Modifikace	Zveřejnění zdrojového kódu Udržení stejné licence Patentovou klauzuli	Zodpovědnost Záruky
Mozilla Public License 2.0	MPL 2.0	Komerční využití Distribuci Modifikace	Zveřejnění zdrojového kódu Udržení stejné licence na modifikovaný kód	Zodpovědnost Záruky
MIT License	MIT	Komerční využití Distribuci Modifikace	Zahrnutí kopie licence	Zodpovědnost Záruky Užití ochranné známky
The 2-Clause BSD License	BSD 2-Clause	Komerční využití Distribuci Modifikace	Zahrnutí kopie licence	Zodpovědnost Záruky Užití ochranné známky
The 3-Clause BSD License	BSD 3-Clause	Komerční využití Distribuci Modifikace	Zahrnutí kopie licence Nesmí být použito jméno přispěvatelů k propagaci odvozených produktů bez jejich souhlasu	Zodpovědnost Záruky Užití ochranné známky
Apache License, 2.0	Apache 2.0	Komerční využití Distribuci Modifikace	Zahrnutí kopie licence Označení změn	Zodpovědnost Záruky

Tabulka 1. Stručný přehled práv licencí

1. 3. Prezentace

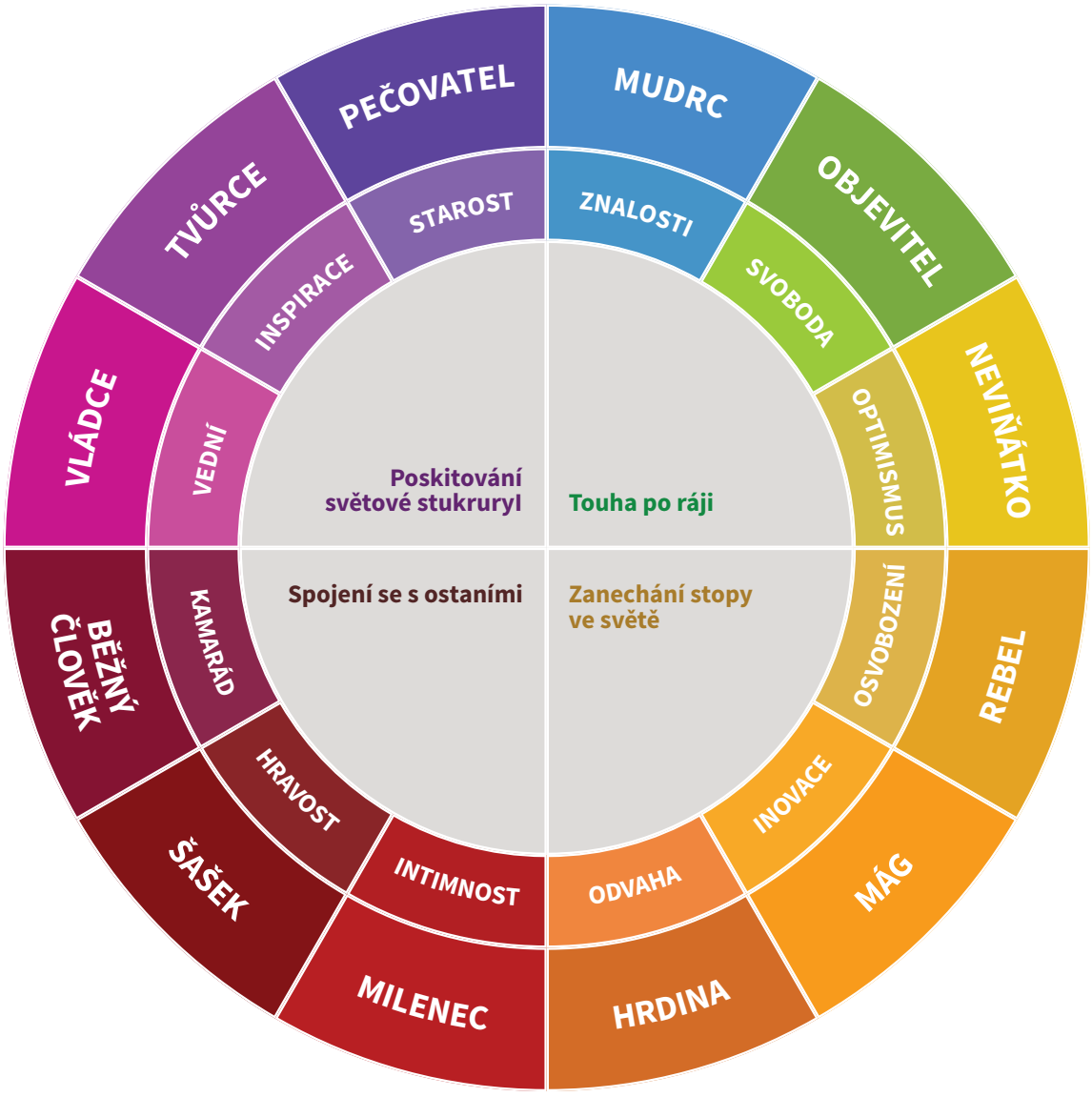
V době digitálního věku je reprezentace jakéhokoli produktu klíčovým elementem při dosahování úspěchu. Tento princip operuje jak v rámci proprietárního softwaru, tak i u softwaru svobodného. Proces přitahování nových uživatelů se stává stále náročnějším úkolem, a proto je nezbytné budovat značku jako takovou. Proprietární software má za úkol vybudovat svůj brand na takovou úroveň, aby dokázal přesvědčit zákazníky k investici svých finančních prostředků. Naopak, softwaru open-source se tato úloha jeví jako mnohem obtížnější, jelikož je nutné uživatele přesvědčit o bezpečnosti používání a o tom, že je hodnotným využitím jejich času.

Konstrukce brandu není jednoduchou a krátkodobou záležitostí, nicméně, existuje metoda, jak tento proces usnadnit. Napříč kulturami se objevují jisté archetypy, jež popsal Carl Gustav Jung. Tyto archetypy byly později upraveny pro potřeby marketingu Carol Pearsonovou, která identifikuje 12 základních typů. Tyto archetypy umožňují značce komunikovat se zákazníkem konzistentním způsobem.[12][13]

1. 3. 1. Popis archetypů

Stejně jako psychologové, i odborníci na branding a marketing zkoumají lidskou povahu a chování. Cílem každé organizace by měla být konzistence vystupování, a proto je nezbytné mít soubor pravidel, kterými se firma prezentuje. K tomuto účelu slouží 12 archetypů, které mají jasně vymezené chování. Jelikož se jedná o archetypy, které jsou všeobecně známy z různých příběhů, které si vyprávíme po mnoho století, není pro nikoho problém rozlišit mezi šaškem a mudrcem.[14][15]

Následující obrázek vyobrazuje ve vnějším kruhu název archetypů. Prstenec další úrovně zobrazuje výrazné vlastnosti daného archetypu a nakonec uprostřed se nachází souhrná charakteristika více archetypů.



Obrázek 1. Kruh archetypů vhodných pro marketing

I přesto, že proces tvorby a udržování značky v souladu s archetypem je fascinující a otevírá široké spektrum tématických oblastí, bude náš další pohled na tuto problematiku pouze povrchní. Důvodem je, že rozsah a specifika témat značně překračují rámec této práce.

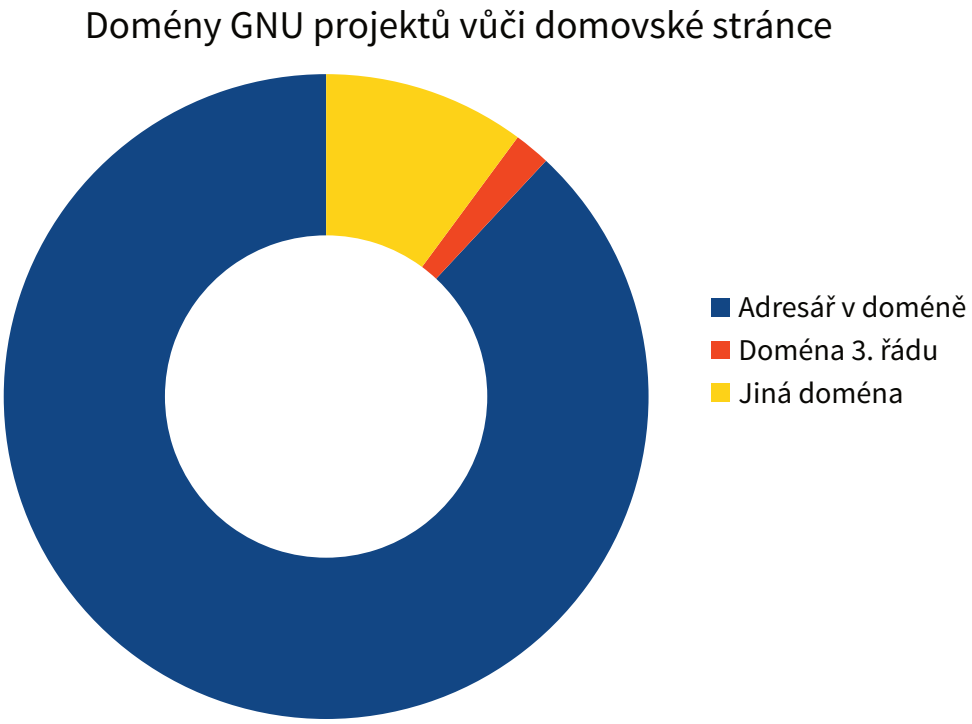
1. 3. 2. Název

Prvotním kontaktem potenciálního uživatele se softwarem je název spolu s krátkou charakteristikou jeho funkcí. Tento název by měl být stručný, dobře vyslovitelný v angličtině, jež je lingua franca celosvětového internetu.

Pokud je potenciální uživatel vůči danému softwaru zaujat a poznal už název a stručný popis z jiného zdroje, bude pravděpodobně toužit dozvědět se více. Logickým krokem je návštěva oficiálního webu projektu. URL takového projektu by měla mít název projektu a reflektovat jej jako doménu druhého řádu a použít „org“ či „io“ jako doménu prvního řádu. Tyto domény prvního řádu ve znalejších uživatelných evokují projekt s otevřeným kódem.

Pokud se jedná o firmu s více open-source projekty v portfoliu, je možné uplatnit dřívější postup a pro každý projekt vytvořit novou doménu. Další možností je tvorba domény třetího řádu, přičemž doména druhého řádu je náš firemní web. Třetí variantou je umístění názvu projektu jako adresář na webu, nicméně tato metoda je nejméně vhodná, protože je to nejméně viditelný způsob oddělení.

Jak projekt GNU řeší tuto problematiku webových stránek pro projekty, ukazuje následující obrázek.[16]



Obrázek 2. Typ domény GNU projektů

Tato statistika byla získána pomocí jednoduchého python skriptu, jenž je zde vypsán.

```
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse

url = 'https://www.gnu.org/software/software.html'

response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

div = soup.find('div', {'class': ['package-list', 'emph-box']})

links = div.find_all('a')

custom_domain_urls = []
third_level_gnu_urls = []
regular_gnu_urls = []

for link in links:
    # Add schema and domain to relative links
    full_url = urljoin(url, link['href'])
    redirected = requests.get(full_url)
    final_url = redirected.url

    # Division of URLs into three categories
    parsed_url = urlparse(final_url)
    domain_parts = parsed_url.netloc.split('.')

    # We find out if the URL belongs to gnu.org, ignoring www
    if domain_parts[-2:] == ['gnu', 'org']:
        # Remove www if it exists
        domain_parts = [part for part in domain_parts if part != 'www']

        if len(domain_parts) > 2:
            third_level_gnu_urls.append(final_url)
        else:
            regular_gnu_urls.append(final_url)
    else:
        custom_domain_urls.append(final_url)
```

Jak je patrné, GNU řeší tuto problematiku většinou nejméně vhodnou variantou, což je pravděpodobně způsobeno dlouholetou historií celého projektu GNU.

1. 3. 3. Web

Webová stránka softwaru hraje kardinální roli při prezentaci a propagaci jakéhokoli open-source programu. Účel, design a uživatelská přívětivost tohoto internetového prostoru mohou dramaticky ovlivnit uživatelovu perspektivu produktu, a tím i jeho úspěch na trhu.

Základním předpokladem je, že stránka by měla odrážet aktuální trendy a konvence. To znamená, že by měla být koncipována a upravována tak, aby byla pro uživatele atraktivní a přehledná, včetně responzivnosti, vizuálního stylu a snadné navigace.

Responzivita je klíčovým prvkem, protože v dnešní době je internet především doménou mobilních zařízení. Design webové stránky by tedy měl být optimalizován pro různé velikosti obrazovek, od chytrých telefonů až po stolní počítače. Stránka by měla být intuitivní a přístupná na jakékoliv platformě a zařízení, a tak působit dobře jak na mobilních zařízeních, tak na stacionárních počítačích.[17]

Taktéž bychom neměli opomíjet vizuální styl. Ten by měl odrážet hodnoty a cíle daného

open-source projektu. K tomu by měl být v souladu s celkovým brandem a případně s archetypem, který projekt využívá. To může napomoci vytvoření jednotného a srozumitelného obrazu produktu, který je pro uživatele atraktivní a zároveň poskytuje informace o jeho klíčových vlastnostech a výhodách. K dispozici musí být snadný přístup k instalačním balíčkům. Další podstatnou funkcí takové stránky je možnost okamžitého finančního příspěvku, tlačítko pro příspěvek by mělo být dobře viditelné a poskytnout přispěvateli více možností, zároveň je vhodné pravidelné finanční přispěvatele zveřejnit. Licence, uživatelská dokumentace a část pro vývojáře by měly být také snadno dostupné.

Jako poslední bod stojí za zmínku, že web by měl být nejen vizuálně atraktivní a funkční, ale také obsahovat důležité informace o produktu, jeho vývoji, komunitě a možnostech zapojení. Transparentnost a otevřenost informací jsou v open-source prostředí zcela nezbytné a měly by být prezentovány přímo na webových stránkách.

Všechny tyto faktory dohromady vytvářejí efektivní a úspěšnou webovou prezentaci, která může přispět k lepšímu šíření a popularizaci mezi potenciálními uživateli a vývojáři.

1. 4. Dokumentace

Pokud chceme, aby náš software byl používaný a současně se vyvíjel, je naprosto nezbytné zajistit kvalitní dokumentaci. Pro uživatelsky orientovanou aplikaci je základem přítomnost jak dokumentace pro koncového uživatele, tak pro vývojáře.

1. 4. 1. Uživatelská dokumentace

Uživatelská dokumentace je instrumentální nástroj pro naučení uživatelů, jak efektivně ovládat náš software. V případě mnoha programů může být tato dokumentace snadno přehledná, zejména pokud dodržíme zaběhlá pravidla týkající se designu aplikací. Pokud je software dostatečně intuitivní a jeho funkce jsou zřetelně popsány přímo v aplikaci, uživatelé budou zřídka kdy muset konzumovat uživatelskou dokumentaci pro většinu interakcí s aplikací. Tento přístup výrazně zvyšuje uživatelskou přívětivost aplikace a především v počáteční fázi projektu nám umožňuje do značné míry obejít se bez detailní uživatelské dokumentace.

1. 4. 2. Vývojářská dokumentace

Pro vývojářskou dokumentaci je nezbytné dosáhnout správné rovnováhy mezi samo-dokumentujícím se kódem a doplňkovou dokumentací. Důvodem je skutečnost, že k dokumentaci kódu se často nepřistupuje s dostatečnou péčí a mnozí vývojáři ji nepovažují za svou oblíbenou disciplínu. Ideální dokumentace by měla podrobně popisovat všechny funkce, jejich závislosti na jiných funkcích, využití těchto funkcí v rámci projektu a celkovou strukturu projektu. Cílem je, aby vývojář, který si pročte tuto dokumentaci, byl schopen přispívat do projektu bez nejasností ohledně fungování jakékoli části kódu.

Nicméně vytvoření takto komplexní dokumentace může být velmi časově náročné a prakticky nemožné. Proto by se vývojáři měli snažit psát kód tak, aby byl co nejvíce srozumitelný, a nesmějí opomíjet užití komentářů, které mají v komplexních funkcích neocenitelnou hodnotu pro každého, kdo s funkcí není obeznámen. Když mluvíme o „funkcích“ v tomto kontextu, máme na mysli všechny části kódu, které mají svou strukturu, jako jsou třídy, metody, funkce a struktury.

1. 5. Komunita a dobrovolníci

Úspěch open-source projektu závisí na silné a zapojené komunitě, a to jak na komunitě uživatelů, tak vývojářů. Efektivní fungování takové komunity vyžaduje charismatického vůdce nebo takovou atmosféru, která mezi členy vytváří pocit sounáležitosti a touhu společně vést projekt k prospěchu všech jeho uživatelů. Tyto dvě komunity – uživatelů a vývojářů – jsou životně důležité pro celý projekt.[4]

Nicméně, je důležité si uvědomit, že hranice mezi těmito dvěma skupinami nejsou pevné. Vývojář může být zároveň uživatelem produktu a naopak, běžný uživatel se může zapojit do vývoje. Přestože pro potřeby naší další diskuze budeme rozlišovat mezi těmito dvěma skupinami, nesmíme zapomenout, že jeden dobrovolník může hrát různé role v rámci komunity.[4]

1. 5. 1. Běžný uživatel

Většina uživatelů open-source softwaru není schopna přímo přispět do kódu, což je z velké části závislé na cílové skupině uživatelů. Nicméně to neznamená, že by byl běžný uživatel méně důležitý než vývojáři. Vezměme si například Blender, open-source 3D grafický software. Podle dat z roku 2021 na oficiálních stránkách Blenderu, návštěvnost webu stále prudce stoupá.[18]

Tato rostoucí popularita přináší řadu přínosů. Běžní uživatelé vytváří návody, sdílejí své výtvory na sociálních sítích a rozšiřují povědomí o Blenderu. Taková popularita přitahuje velké sponzory, což umožňuje Blender Foundation, organizaci stojící za Blenderem, zaměstnat kvalifikované programátory. Rovněž to láká vývojáře, kteří chtějí být součástí tak velkého projektu.

Tyto faktory dohromady urychlují vývoj softwaru a tlačí konkurenci, jako je Autodesk, k inovacím. Ačkoli se to může zdát je odbočení od tématu „běžného uživatele“, opak je pravdou. Bez aktivní uživatelské komunity by tento růst a pokrok nebyl možný.

2. MOŽNOSTI ZABEZPEČENÍ DAT

Lidstvo jako takové v zápisu a úschově dat setrvává již několik tisíc let. První písmo se objevuje 4000–3000 let př. n. l. v Egyptě. Toto schéma později následují další kultury. [19][20]

Pro současného běžného člověka jsou tyto data nečitelné. Ovšem jediný důvod proč tyto data nepřečteme, je jen protože neznáme jazyk, v němž jsou zapsány. I když se na první pohled zdá, že data zapsaná v mrtvém jazyce jsou bezpečně uložena, v případě že se však objeví obdoba Rosettské desky, tato data dokáže přečíst každý dostatečně jazykově nadaný jedinec.[19]

Vzhledem k tomu, že uchování dat v jiném jazyce nemá vliv na jejich zabezpečení soukromí, je nezbytné zvolit alternativní metodu. Nejsnadnější způsob jak soukromá data ochránit je jejich uschování na místo kde o nich ví pouze pověřené osoby. Takové místo uložení dat však může najít i osoba nepověřená. Zde jsou možné tři varianty jak data dostatečně uchránit.

1. Místo uložení dat hlídat.
2. Data rozdělit tak, že z jedné části nelze jednoznačně určit obsah dat.
3. Data upravit pomocí algoritmu a tajného klíče – šifrovat data.

2. 1. Známé přístupy zabezpečení dat v kyberprostoru

V úvodní části této kapitoly bylo naznačeno, jak je možné bezpečně uložit data v reálném světě. Tato kapitola se bude zabývat převedením těchto technik do kyberprostoru.

2. 1. 1. Ukrývání dat

Stejně jako v reálném světě, i v digitálním je možné ukrýt data. První myšlenka, která nás může napadnout, spočívá v ukrytí dat na webu, kde se k nim dostane pouze člověk, který zná přesný odkaz. Avšak problém tohoto přístupu zůstává neměnný, neboť na tento odkaz se může dostat i osoba nepovolaná.

Dalším způsobem skrytí dat, je smísiti data s jinými, tak aby nezasvěcený uživatel nevěděl, že jde o směs dvou informačních pramenů. V reálném světě lze najít příklad takovéto úschovy dat v situaci, kdy jsou tajná poselství skryta ve formě neviditelného inkoustu na jiné listině.[21][22]

Jeden z největších neduhů tohoto přístupu je skutečnost, že nejsme schopni uschovat větší množství informací. Jelikož musíme zachovat data formátu, ve kterém svá data ukrýváme, a tak jsme schopni uložit pouze zlomek velikosti souboru (dat), do něhož informace ukrýváme.

Typickým příkladem takovéto úschovy dat může být zapsání informací do obrázku, což se provádí například změnou nejméně důležitého bitu v datech o barvách. Pokud budeme obrázek považovat za pole, které obsahuje další pole reprezentující řádek obrázku a v každém řádku pole tři hodnoty, jež reprezentují jednotlivé kanály RGB s bitovou hloubkou 8 bitů na kanál, potom nám obrázek o rozlišení FullHD reprezentuje data o velikosti 6 220 800 bytů. Zaznamenáme-li do každého byte jeden bit informace z jiného pramene, můžeme do tohoto obrázku zapsat 6 220 800 bitů dat, což je 777 600 bytů, tedy jedna osmina na čistá obrazová data.

Toto mísení dvou pramenů informací se nazývá steganografie. Pro účely bezpečné úschovy většího množství dat je tento způsob nevhodný, jelikož by bylo nutné vytvářet druhý pramen dat, do kterého bychom ukládali svá tajná data. I v případě, že bychom neměli problém se získáním vhodného pramenu dat pro steganografii. Je tu velké riziko narušení integrity dat.

2. 1. 2. Omezení přístupu

Pokud jsou data zapsána na fyzické médium jako je papír, je možné toto médium uzavřít například do trezoru. Princip trezoru je snadný, jedná se o úložný prostor, jenž není možné otevřít v daném čase jinak než znalostí přístupového kódu.

V kyberprostoru je tento princip obdobný, avšak mnohem více náchylný k úniku dat. Jako příklad omezení přístupu může sloužit Linux, v tomto operačním systému máme jméno uživatele a skupiny. Každý uživatel musí být přiřazen do minimálně jedné skupiny. Každý soubor (soubor je i složka) v souborovém systému má přiřazeného majitele a skupinu. Operační systém potom přiřazuje možnosti, které lze nad souborem provést na základě oprávněních, které má soubor přiřazen. Přiřazené oprávnění se skládá ze tří úrovní – oprávnění majitele souboru, oprávnění skupiny souboru a oprávnění ostatních. [23][24][25]

Pozn. Tato data byli vypočítaná na základě vzorce: maxDat = (Wpx × Hpx × Ch × E) ÷ 8. MaxDat je maximální počet dat, které lze uložit Wpx je šířka obrázku v pixelech, Hpx je výško obrazu v pixelech, Ch je počet kanálů pro pixel, E je počet upravitelných pixelů a 8 reprezentuje počet bitů v 1 byte. Tedy pro uvedený příklad to znamená: (1 920 × 1 080 × 3 × 1) ÷ 8 = 777 600 byte.

Následující scénář slouží k nastínění nedostatkům problematiky omezení přístupu: Mějme počítač ve veřejném prostoru s operačním systémem Windows, jenž nevyužívá šifrování dat na disku. Běžné využití takového počítače spočívá v možnosti využití připojení se na internet, vytištění dokumentů za poplatek apod. Takový počítač využívá omezení přístupu v rámci operačního systému na základě rolí, a tak se zdá, že systém může ovlivňovat pouze ten kdo má přístup k roli správce. Dále mějme útočníka, jenž si přinese live USB s Kali. Útočník zapojí USB do našeho počítače, resetuje jej aby spustil vlastní systém. Jelikož používá jiný operační systém, netýkají se ho nastavená omezení a tak si může se systémem provést co se mu zlíbí.[26]

2. 1. 3. Šifrování dat

Jak již bylo naznačeno v předchozích částech, je tu možnost data šifrovat. Od časů, kdy lidstvo stvořilo písmo, usilovalo o znesnadnění čtení těm, jenž nebyli povoláni. Asyřané, Babyloňané, Egypťané i Hebrejci vytvořili protokryptografické systémy. Přestože obecně panuje představa, že první šifrovanou korespondenci zasílal Julius Cesar, první záznam o šifrované korespondenci pochází od Aenease Tacticuse. Tento muž popsal používání transpoziční šifry Sparťany již ve 4. stol. př. n. l., když jako první v záři historie vykročil na pole kryptografie.[22][27]

Pokud si přejeme digitální data zašifrovat, musíme procesy šifrování a dešifrování přetvořit na matematické funkce. Na vstupu šifrovací funkce máme čitelná data a tajemný klíč, zatímco na výstupu získáváme data, která nelze bez klíče či prolomení šifry opětovně přeměnit na čitelnou podobu. Na vstupu dešifrovací funkce nalezneme zašifrovaná data a klíč, a na výstupu jsou pak obdržena data nezašifrovaná. [28]

Abychom zjednodušili pochopení, představme si data ve formě řetězce obsahujícího 26 znaků anglické abecedy. Každý znak lze reprezentovat číslem v rozmezí 0 až 25. Převedeme-li text „ahoj“ na pole čísel v uvedeném rozmezí, získáme pole {0, 7, 14, 9}. Pro názornost zvolme klíč s hodnotou 3. Upravíme-li každý prvek v dříve definovaném poli dle následujícího vzorce:

```
<nový prvek> = (<původní prvek> + <klíč>) mod <délka abecedy>
```

obdržíme pole {3, 10, 17, 12}. Převedeme-li číselnou reprezentaci písmen zpět na písmena, zašifrovaný řetězec zní: dkrm. Touto cestou jsme přetvořili šifrování Caesarovy šifry do matematického vzorce.

2. 1. 4. Zhodnocení jednotlivých přístupů

Steganografii lze nejlépe využít, když je třeba propašovat tajná data skrze kontrolovaný prostor. Pokud steganografii v kombinaci se šifrováním uplatníme, získáme poměrně bezpečný přenosový kanál, jenž odolává nepovolaným pohledům.

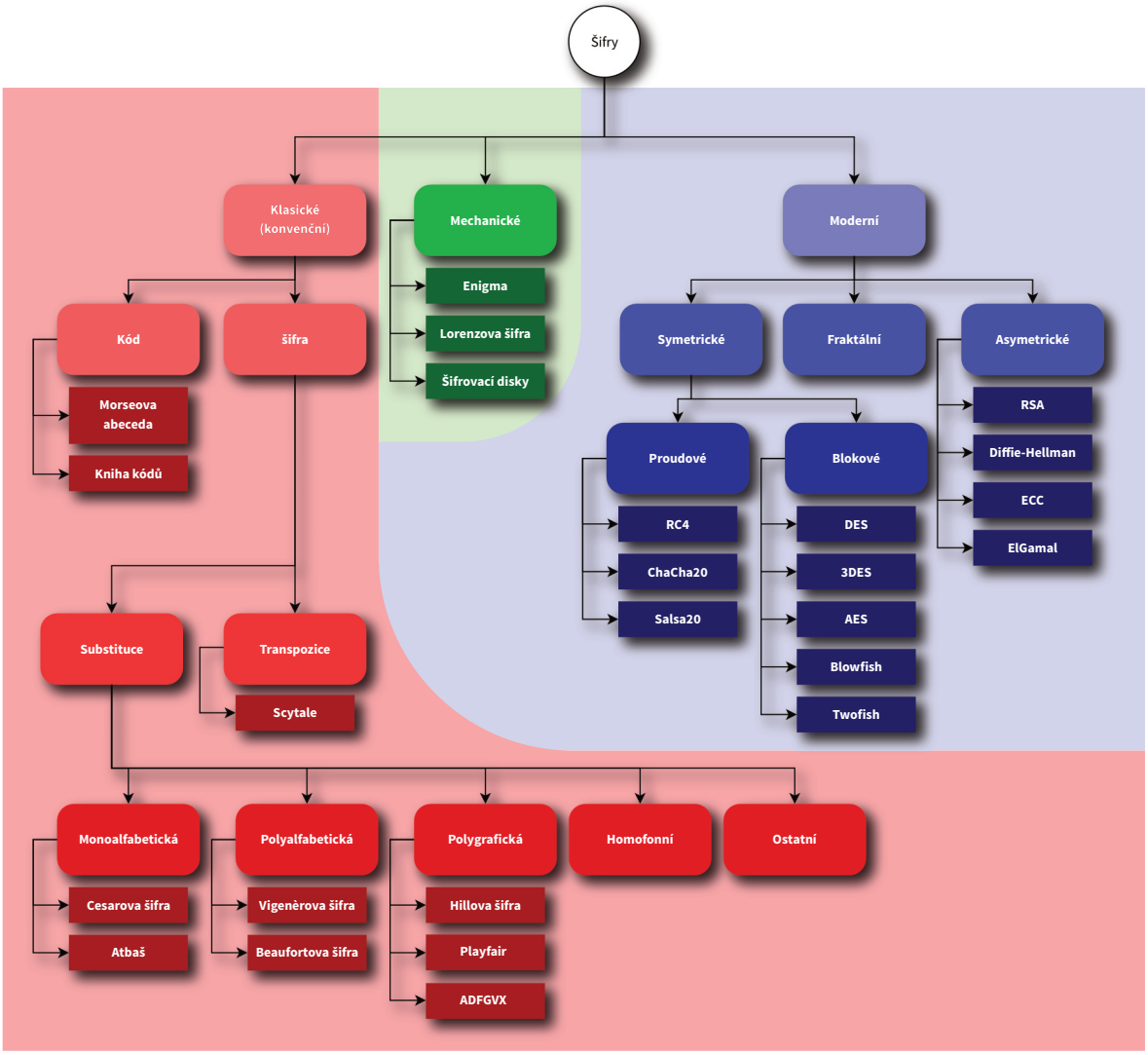
Pro efektivní zabezpečení dat prostřednictvím omezení přístupu jsme nuceni spoléhat na rozsáhlejší systémy. Typickým příkladem takového systému může být operační systém či webová aplikace. Je však důležité si uvědomit, že samotné omezení přístupu nezaručuje ochranu dat mimo daný systém. Pro odolnost vůči čtení, zápisu a spuštění je třeba využít šifrování, jež představuje další vrstvu ochrany.

A právě to nás přivádí k poslední části zabezpečení dat – šifrování. Šifrování dat představuje efektivní způsob, jak zabránit čtení, zápisu i spouštění neoprávněným osobám, a tak uchránit naše drahocenné informace. Existuje velké množství šifer, které se liší svou bezpečností, a proto je klíčové pečlivě zvolit vhodnou šifru pro zabezpečení našich tajemství, neboť od tohoto rozhodnutí závisí účinnost ochrany.

2. 2. Bezpečnost šifrovacích algoritmů

V této části se zaměříme konkrétně na bezpečnost a princip funkčnosti různých šifrovacích algoritmů, jenž hrají zásadní roli v ochraně dat a jejich přenosu v různých aplikacích. Představíme si, jak se šifry vyvíjeli v průběhu času, od klasických přes mechanické až po moderní šifrovací metody, a jak se jejich bezpečnost zvyšovala v reakci na zvyšující se bezpečnostní hrozby.

Následující info grafika poskytuje strukturovaný přehled šifer a šifrovacích algoritmů podle jejich kategorií a podkategorií, které budeme v této kapitole probírat. Pomocí tohoto stromového schématu si můžeme snadno udělat představu o vztazích mezi jednotlivými šiframi a jejich vývojem.



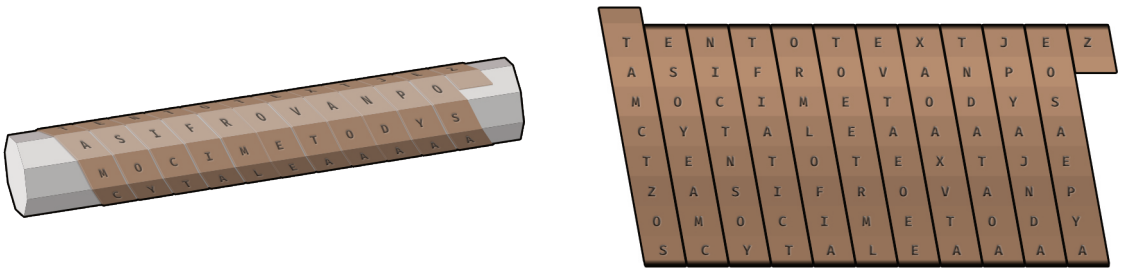
Obrázek 3. Info grafika, strom šifer, jejich kategoriích a podkategoriích

V rámci kapitoly se zaměříme na analýzu bezpečnosti těchto šifrovacích algoritmů, jejich odolnost vůči různým útokům a faktory, které ovlivňují jejich praktickou použitelnost. Diskutovat budeme také o potenciálních slabých místech a doporučeních pro zajištění bezpečnosti při jejich použití.

Pozn. Je podstatné zmínit, že zastoupení kódů, jako je kniha kódů či Morseova abeceda, v této info grafice může být považováno za diskutabilní. Avšak, je nezbytné si uvědomit, že kód je ve své podstatě substituční šifrou, která disponuje pevným klíčem, jež je všeobecně znám a rozšířen.

2. 2. 1. Scytale

Tuto šifru lze datovat až do roku 400 př. n. l. a jedná se o jednoduchý princip založený na namotaní pásku kolem tyče. Proces šifrování spočívá v tom, že na válcovitý předmět, například hranol, spirálovitě navineme pásek zapisovatelného média (jako je proužek papyru). Poté zapisujeme jednotlivá písmena po jednotlivých otočkách. Následně se zpráva odesílá bez předmětu, na který se pásek namotává. Přečíst zprávu je možné, když pásek navineme na předmět se stejným poloměrem, jaký byl použit při šifrování. Pro dosažení účinnosti šifry je samozřejmě nutné při šifrování zapsat zprávu po celém obvodu jako je naznačeno na následujícím obrázku. [27][19]



Obrázek 4. Princip scytale

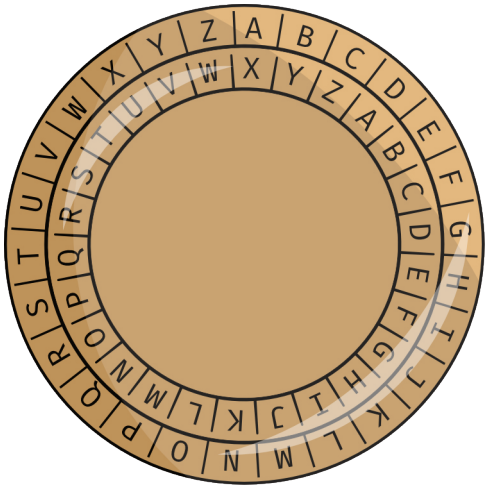
Kdybychom chtěli tuto šifru převést do virtuální podoby, vezmeme data, které narovnáme po řádcích do matice, rozměry této matice nám definují klíč (stačí pouze jeden rozměr). Matici transponujeme a řádky matice postupně vypíšeme zpět jako souvislá data. Poté v závislosti na implementaci se bude či nebude lišit klíč pro šifrování a klíč pro dešifrování. Případ který vidíme na obrázku výše není v této implementaci podporován. Podpora použití by vyžadovala vlastní implementaci pseoudomatice (pole polí o rozdílné velikosti) a její transpozici.

Bezpečnost takové šifry není však nijak valná. Pomocí algoritmu zkoušející různé rozměry matice a hledajícího výrazy či datové struktury podle očekávaných zašifrovaných dat.

2. 2. 2. Monoalfabetické substituční šifry

Princip monoalfabetických substitučních šifer spočívá v tom, že abeceda původního textu (jazyka) je nahrazena jinou abecedou. Tato náhradní abeceda může být vytvořena rotací původní abecedy, přepisem abecedy pozpátku, kombinací těchto technik či vytvořením vlastního druhu abecedy. Mezi nejznámější historické šifry tohoto druhu patří Caesarova šifra. Její původní verze měla pevně stanovený posun (rotaci) o 3 písmena v abecedě. Později se název Caesarovy šifry začal používat pro jakoukoli šifru tohoto typu s libovolně vybraným posunem (rotací). Správní název je však afinní šifra, jelikož se zde využívají afinní transformace.[19][29][30]

Pro afinní šifru existuje jednoduché zařízení, které pomáhá se šifrováním a dešifrováním. Jedná se o mechanicky pohyblivé zařízení jenž obsahuje dva soustředné prstence, jeden prstenec s původní abecedou textu a druhý prstenec s nahrazující abecedou. Tomuto zařízení se jinak taky říká šifrovací disky. Pootočením jednoho prstence získáme posun (rotaci) abecedy. K jasnější představě následující obrázek, jenž ukazuje nastavení pro Cesarovu šifru.



Obrázek 5. Šifrovací disky pro šifrování a dešifrování afinních šifer v nastavení pro Cesarovu šifru

Kromě afinních šifer do této kategorie patří šifra Atbaš, stejně jako u Cesarovi šifry, jedná se o konkrétní případ šifry. Pro atbaš platí jednoduché pravidlo, originální abeceda je nahrazena stejnou abecedou, která je však zapsáná pozpátku. Tato šifra je zastoupena v bibli.[19][31]

Provedení tohoto typu šifer do elektronické podoby není problém, Stačí nám původní a výstupní abeceda, číslo reprezentující rotaci a text na zašifrování. Samotnou implementaci můžeme provést tak, že vstupní text převedeme na čísla pomocí indexů původní abecedy, text převedený na pole čísel postupně iterujeme s následujícím vzorcem:

```
<nové písmeno> = (<původní písmeno> + <rotace>) mod <délka abecedy>
```

Takto upravené pole převedeme zpět na text pomocí indexů výstupní abecedy (bez rotace).

Dalším způsobem jak tuto šifru implementovat je zadání původní a výstupní abecedy, číslo reprezentující rotaci a text na zašifrování. Tentokrát však použijeme datovou strukturu slovník. Původní abecedu využijeme jako klíč slovníku a jako hodnotu využijeme výstupní abecedu kterou ještě rotujeme podle výše zmíněného vzorce. Teď pouze stačí iterovat textem pro šifrování jako polem klíčů a na výstup zapisovat hodnoty slovníku.

Známe-li původní a výstupní abecedu, je poměrně snadné získat původní text hrubou silou, jelikož má šifra velice omezený počet klíčů. V případě že neznáme abecedy, lze využít frekvenční analýzu zašifrovaného textu, k tomu nám stačí znát jazyk původní zprávy a statistiku výskytu jednotlivých písmen v daném jazyku.

2. 2. 3. Polyalfabetické substituční šifry

Polyalfabetické substituční šifry představují vývojový krok oproti monoalfabetickým šifrám, který ztěžuje frekvenční analýzu šifrovaného textu. Tyto šifry spočívají v použití více než jedné abecedy pro šifrování textu, čímž mění četnost písmen v zašifrovaném textu. Tímto způsobem se zvyšuje bezpečnost šifry, protože útočník nemůže jednoduše zjistit původní text pomocí frekvenční analýzy. Mezi nejznámější polyalfabetické substituční šifry patří Vigenèrova a Beaufortova šifra.

Vigenèrova šifra je polyalfabetická substituční šifra, která používá klíč složený z písmen abecedy. Pro šifrování se vytvoří Vigenèrova tabulka, kde je každému písmenu přiřazena jedna abeceda posunutá o určité množství písmen. Klíčem šifry je slovo či fráze, která určuje, jakou abecedou se dané písmeno zprávy šifruje. Vigenèrova šifra zajišťuje vyšší zabezpečení než monoalfabetické šifry, avšak i tuto šifru je možné prolomit pomocí frekvenční analýzy písmen či kasiského testu. Pokud ovšem splníme kritéria, jako je délka klíče stejně dlouhá jako text a klíč dokonale náhodný, můžeme Vigenèrovu šifru považovat za neprolomitelnou, v takovém případě se bavíme o šifře s jednorázovou tabulkou. Takový přístup je však velice náročný na správu klíčů.[19][32][33][34][35][36]

Beaufortova šifra je varianta Vigenèrovy šifry, která používá zrcadlovou Vigenèrovu tabulku. Beaufortova šifra je tedy reciproční, což znamená, že šifrování a dešifrování probíhá stejným způsobem. Stejně jako u Vigenèrovy šifry se používá klíč složený z písmen abecedy a zrcadlová Vigenèrova tabulka pro určení, jakou abecedou se dané písmeno zprávy šifruje. Beaufortova šifra poskytuje podobnou úroveň zabezpečení jako Vigenèrova šifra.

Implementace polyalfabetických šifer do elektronické podoby je rovněž možná. Pro Vigenèrovu a Beaufortovu šifru bychom vytvořili tabulku s posunutými abecedami a šifrovali text podle klíče a tabulky.

I když polyalfabetické šifry poskytují vyšší zabezpečení než monoalfabetické šifry, dnes již nejsou považovány za dostatečně bezpečné pro šifrování citlivých informací, či jsou nevhodné v náročnosti na správu klíčů Moderní kryptografické metody, jako například symetrické šifry nebo asymetrické šifry, poskytují výrazně vyšší úroveň zabezpečení.

2. 2. 4. Polygrafické substituční šifry

Polygrafické substituční šifry představují další vývojový krok šifer, který zlepšuje zabezpečení oproti polyalfabetickým šifrám. Tyto šifry jsou specifické v nahrazení skupin (n-gramů) písmen původního textu skupinami písmen šifrovaného textu, místo toho, aby nahrazovaly jednotlivá písmena, jak je tomu u monoalfabetických a polyalfabetických šifer. Díky tomuto způsobu šifrování se ztěžuje frekvenční analýza šifrovaného textu, což zvyšuje bezpečnost šifry. Mezi polygrafické šifry patří Playfair, ADFGVX a Hillova šifra.

Pozn. Playfair a ADFGVX se někdy řadí do polyalfabetických šifer a jindy do polygrafických šifer. V této práci jsou zařazeny do polygrafických šifer, jelikož pro šifrování a dešifrování Playfair a dešifrování ADFGVX používáme vždy dvě písmena.

Playfaire šifra je polygrafická substituční šifra, která šifruje text pomocí bigramů, tedy dvojic písmen. Pro šifrování se používá tabulka 5 × 5, která obsahuje abecedu bez jednoho písmene (nejčastěji se vynechává „J“ nebo „Q“, poté „J“ = „I“ nebo „Q“ = „K“). Pokud bigram obsahuje dvě stejná písmena, musí se rozdělit na dva bigramy s původním písmenem a jedním zástupným (obvykle statisticky nejméně četným). Taktéž pokud není délka textu sudá, použijeme doplňující znak k doplnění délky. Klíčem šifry je slovo či fráze, která se použije pro sestavení tabulky, tedy pro vytvoření pořadí písmen v ní (již použitá písmena vynecháváme). Při šifrování se dvě písmena zprávy spojí do bigramu, jak již bylo popsáno a podle tabulky se nahradí novým bigramem. Tato změna písmen probíhá podle následných pravidel:

1. Při přepisu písmen dodržujeme pořadí původních písmen
2. Pokud bigram v tabulce vytvoří čtverec, přepíší se písmena v opačné diagonále na stejném řádku jako původní písmeno z bigramu.
3. Pokud je bigram v tabulce na jednom řádku, písmena se posunou o jedno písmeno doprava. Pokud je písmeno v posledním sloupci, přepíše se písmeno stejného řádku z prvního sloupce.
4. Pokut je bigram v tabulce v jednom sloupci, písmena se posunou o jedno písmeno dolů. Pokud je písmeno v posledním řádku, přepíše se písmeno z prvního řádku ve stejném sloupci.

Playfair šifra zajišťuje vyšší zabezpečení než monoalfabetické šifry, avšak i tuto šifru je možné prolomit pomocí frekvenční analýzy bigramů.[37][38]

Šifra ADFGVX, někdy také ve formě ADFGX, je podobná šifře Playfair. Její název vyplývá z výstupu šifry, který se reprezentuje bigramy těchto písmen. Výběr těchto písmen je spojen s jejím využitím – šifra se používala k odesílání zpráv v Morseově abecedě a zvolená písmena jsou velmi odlišná. K šifrování algoritmem vytváříme tabulku 6 × 6 nebo 5 × 5, v závislosti na variantě šifry. Každý řádek a sloupec označíme písmeny „A“, „D“, „F“, „G“, „V“ (pouze v případě tabulky 6 × 6 použijeme „X“), které slouží jako souřadnice tabulky. Do této tabulky pak umisťujeme šifrovací klíč, podobně jako v šifře Playfair. Varianta 6 × 6 nám navíc umožňuje zakomponovat do tabulky i čísla nebo jiné znaky.[39]

Hillova šifra je založena na lineární algebře a používá maticové operace pro šifrování a dešifrování textu. Klíčem šifry je invertibilní čtvercová matice s rozměry n × n, která určuje, jak budou skupiny písmen vstupního textu transformovány. Pro šifrování se vstupní text rozdělí na skupiny o velikosti n a každá skupina se následně vynásobí klíčovou maticí. Výsledný zašifrovaný text se získá převedením výsledných vektorů zpět na písmena. [40]

Implementace Hillovy šifry do elektronické podoby zahrnuje několik kroků. Nejprve je třeba zvolit klíčovou matici s rozměry n × n, která musí být invertibilní v modulo délky abecedy. Při šifrování se vstupní text rozdělí na skupiny o velikosti n a každé písmeno v těchto skupinách se převede na číselnou hodnotu dle indexu v abecedě. Následně se skupiny písmen převedených na čísla reprezentují jako vektory a vynásobí se klíčovou maticí v modulu délky abecedy. Výsledné zašifrované vektory se poté převedou zpět na písmena, čímž vznikne zašifrovaný text.

Playfair a ADFGVX si jsou implementačně z velké části podobné, vytvoříme matici do níž zapíšeme klíč podle výše popsaného pravidla. U Playfair poté uplatňujeme pravidla změn písmen v rámci tabulky, u ADFGVX získáváme pozici příslušného písmene v tabulce a tuto pozici převedeme na příslušná písmena.

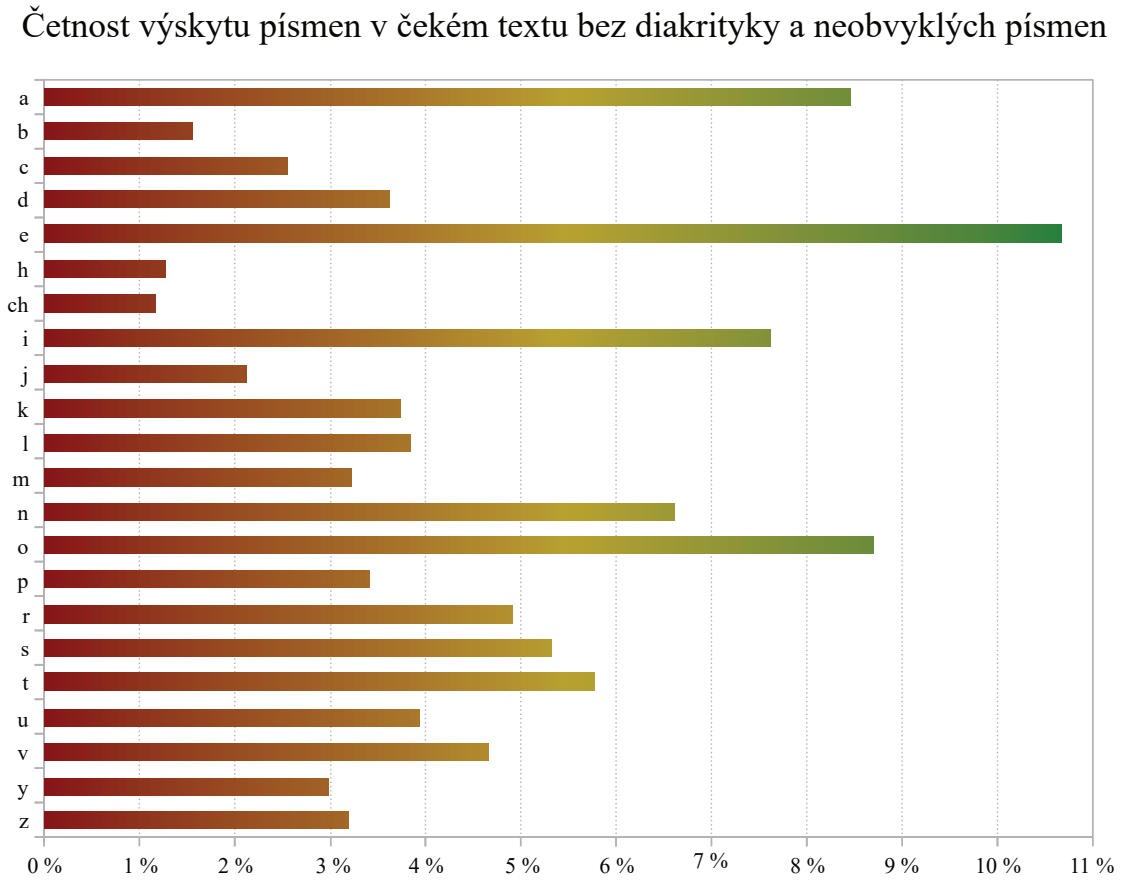
Největší slabinou Hillovy šifry je její zranitelnost vůči známým textovým útokům. Známý textový útok je metoda kryptoanalýzy, při které útočník má přístup k párovým datům otevřeného a zašifrovaného textu. Pokud útočník zná dostatečnou část otevřeného textu a odpovídající zašifrovaný text, může použít tuto informaci k odhalení šifrovací matice. Tato slabina spočívá v lineární povaze Hillovy šifry, která je založena na maticových operacích. Vzhledem k tomu, že šifrování a dešifrování jsou prováděny prostřednictvím maticového násobení, může útočník použít známý otevřený text a zašifrovaný text k vytvoření soustavy lineárních rovnic. Poté může řešit soustavu rovnic, aby získal šifrovací matici a její inverzní matici.[41][42]

2. 2. 5. Homofonní substituční šifra

Homofonní substituční šifry představují další krok ve vývoji šifer, který zlepšuje zabezpečení oproti jednoduchým monoalfabetickým a polyalfabetickým šifrám. Tato šifra spočívá v nahrazení jednotlivých písmen původního textu více různými symboly či znaky, což ztěžuje frekvenční analýzu šifrovaného textu. Tímto způsobem je každé písmeno zprávy zašifrováno několika různými symboly, což ztěžuje kryptoanalýzu. [19]

Prakticky to znamená, že vypočítáme procentuální zastoupení jednotlivých písmen v textu určeného pro šifrování. V případě znaků, jenž mají příliš nízké zastoupení, můžeme uvažovat o zástupných znacích jako v šifře Playfair, kde se nahrazovalo písmeno „J“ písmenem „I“ nebo písmeno „Q“ písmenem „K“. Když máme sestavenou tabulku procentuálního zastoupení jednotlivých písmen, musíme zvolit dostatečně jemný rozptyl, aby nejméně zastoupené písmeno bylo nahrazeno minimálně jedním znakem a každé další písmeno bylo nahrazeno násobkem počtu znaků nejméně zastoupeného písmena. Podmínka pro fungování šifry je, že žádný zástupný symbol nebudeme opakovat ve vznikající tabulce. Jakmile máme sestrojenou celou tabulku, můžeme šifrovat text. To probíhá tak, že pro každé písmeno v prostém textu vyhledáme v tabulce a nahradíme náhodným zástupným znakem písmene. Klíč pro šifrování a dešifrování dat je tedy tabulka písmen a zástupných znaků.

Pokud tedy vezmeme běžnou reprezentaci českého textu bez cizích slov, odstraníme diakritiku, získáme následující graf.



Obrázek 6. Graf ukazující četnost písmen českých písmen, upraveno [43]

Na základě těchto dat můžeme zvolit rozptyl zástupných znaků. Jako nejvhodnější se jeví hodnoty zaokrouhlovat na půl procenta, ovšem pro zjednodušení této ukázky se zvolí zaokrouhlení na celá procenta. Zaokrouhleně zde máme 101% proto potřebujeme vytvořit sto jedna hodnot jenž náhodně rozmístíme ke každému procentu k jednotlivým písmenům. Výsledná tabulka může vypadat následovně:

a	b	c	d	e	h	ch	i	j	k	l	m	n	o	p	r	s	t	u	v	y	z
88	21	32	75	23	26	59	77	68	97	36	41	67	72	22	18	12	80	78	34	87	29
30	90	25	33	27			53	94	92	62	95	79	7	69	6	61	14	19	58	50	16
84		73	93	48			83		44	17	3	43	9	96	0	57	52	15	89	31	28
5			71	99			20		81	82		74	100		4	60	49	86	54		
98				76			66					64	24		56	70	63		13		
2				35			91					8	40				51				
46				85			38					11	45								
1				10			39						47								
				65									55								
				42																	
				37																	

Obrázek 7. Substituční tabulka pro šifrování a dešifrování

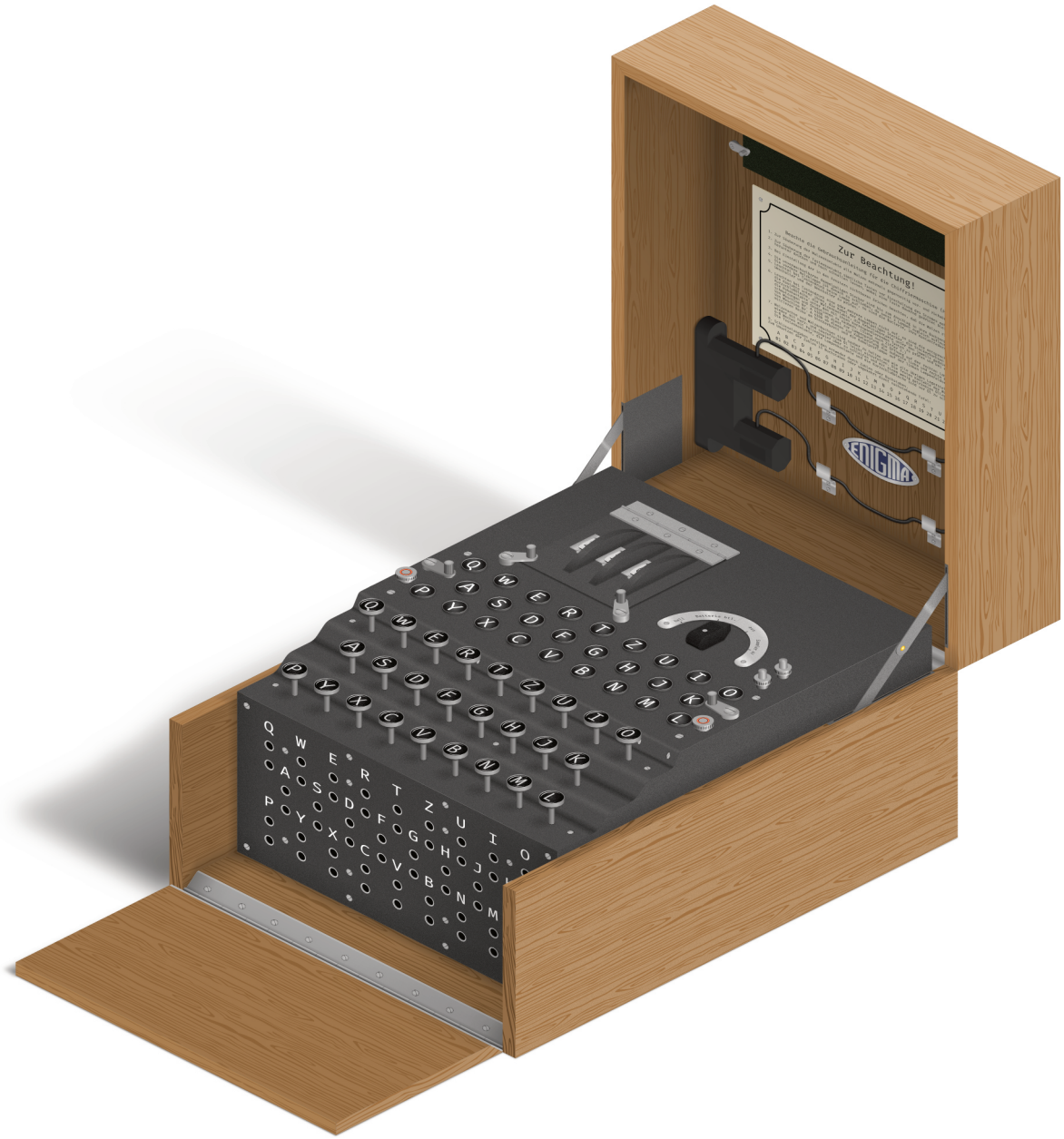
Jak můžeme vidět, pro náhradní hodnoty byli vybrány čísla od 0–100. Každé písmenu v prostém textu převedeme na šifrovaný text tak, že z tabulky vybereme náhodně číslo určené pro dané písmeno.

Implementace homofonní šifry může být provedena například pomocí datové struktury slovník. Postup pro tuto implementaci může být následující: U prostého textu provedeme frekvenční analýzu. Na základě této analýzy zjistíme procentuální zastoupení symbolů a následně určíme vhodný rozptyl pro zástupné symboly. Z těchto dat určíme, kolik je potřeba vygenerovat zástupných symbolů. Pole zástupných znaků náhodně promícháme a začneme je skládat do struktury slovníku ve formátu ’(klíč)<původní symbol>: (hodnoty) <pole zástupných symbolů>’. Tímto způsobem jsme dokázali vytvořit substituční tabulku, jenž představuje klíč. Samotné šifrování proběhne tak, že vstupní text budeme iterovat jako pole slovníkových klíčů, pro každý slovníkový klíč se náhodně vybere hodnota z pole hodnot a zapíše na výstup.

Homofonní šifra je poměrně bezpečná šifra, ovšem je náchylná na frekvenční analýzu. Útočník sice nezíská klíč šifry tak snadno jako u afinních šifer, protože každý zástupný znak by měl být v ideálním rozložení náhodných výběrů zástupných symbolů zastoupen zhruba ve stejném množství, kde ale tento princip selhává je u analýzy bigramů a trigramů.

2. 2. 6. Enigma

Na mechanickou šifru jsme již narazili v dřívější kapitole o afinních šifrách v podobě šifrovacích disků. Dalším krokem této mechanické šifry je nechvalně známá Enigma z 2. světové války. Pomocí tohoto elektromechanického stroje byly šifrovány německé zprávy. Zařízení se skládá z klávesnice, tří a více rotačních disků, propojovací desky a zobrazovač šifrovaných, nebo dešifrovaných písmen pomocí rozsvícení žárovky s dotyčným písmenem. Následující obrázek ukazuje přístroj Enigmy vytvořený na základě 3D modelu na webové stránce Sketchfab.[44][45]



Obrázek 8. Zobrazení přístroje Enigma na základě 3D modelu [45]

Poté co uživatel stiskne klávesu, stanou se dvě věci – propojí se elektrický obvod a aktivuje se mechanizmus pro otočení jednoho čí více rotačních disků. Otáčení disků je realizováno pomocí houpačkové konstrukce, jenž uživatel převrátí pomocí zmáčknutí tlačítka, v pozici rotačních disků jsou dva typy interakce s každým diskem.

První z nich je nástroj na otočení disku, to je prováděno pomocí západkového mechanismu. Z pohledu uživatele disk úplně vpravo se hýbe po každém stisku klávesy, prostřední disk se pohybuje podle výřezů na pravém disku, které umožňují zapadnutí západky do západkového kola a pootočit s ním, stejný scénář potom platí pro každý další disk. Druhý mechanizmus, jenž interaguje s disky, je západka, jenž má za úkol, aby se disk zastavoval v přesné pozici.

Nyní když víme, jak se disky hýbou, můžeme se podívat na to jaký tyto disky mají účel při šifrování. Každý disk má z jedné strany plošky a z druhé piny se kterými propojuje zbytek elektrického obvodu. Těchto plošek a pinů disk obsahuje 52, přičemž každé písmeno má svůj vstup a výstup. Princip disku spočívá v tom, že výstup písmene má jinou pozici vstup.

Další část šifrování se týká propojovací desky, pokud do dírek nic nezapojíme, písmena se nezmění, pokud však použijeme propojovací kabel, který zaplní obě pozice pod písmenem, má za úkol přepojit signál z jednoho písmene na druhé.

Pro pochopení celého šifrování si projdeme cestu elektrického proudu od baterie k žárovce. Baterie je napojena na plíšky pod klávesnicí, pokud uživatel zmáčkne tlačítko, propojí obvod, kde proud protéká směrem propojovací desce. V propojovací desce se v závislosti na tom, jestli je k danému písmenu připojený kabel, změní na jiné písmeno nebo zůstane stejné. Z propojovací desky zamíří směrem k rotačním diskům, kde projde skrze všechny disky, kde dojde k několika posunům v závislosti na počtu disků. Poslední disk je vyveden do reflektoru, který proud vrátí jinou cestou do rotoru zcela vlevo. Následně proud protéká jinou cestou zase skrz rotační disky zpět na propojovací desku, dochází zde ke stejnému chování jako na začátku této cesty. Z propojovací desky se proud vrací do klávesnice, kde klávesy, které nejsou propojeny s baterií, jsou propojeny s odpovídajícími žárovkami. Rozsvícení písmena značí uživateli, jaké písmeno odpovídá stisknuté klávese. Okruh se nakonec uzavírá a vrací se zpět do baterie.

Ačkoliv je obecně známá věc, že se angličanům povedlo prolomit Enigmu, nebyli první komu se to povedlo, za prvního kryptoanalytika, jenž měl úspěch s prolomením vojenské Enigmy je považován Martin Rejewski, který byl schopný luštit Enigmu od roku 1932 až do roku 1939. Později již nebyl schopný držet tempo s vývojem tohoto přístroje.[46]

Polští kryptoanalytici se přesunuli do Francie odkud spolupracovali na lámání šifer Enigmy spolu s Brity z Bletchey Parku. I když Němci Enigmu vylepšili o další dva rotory, dařilo se jim šifru lámat z důvodu mnoha chyb jenž se dopouštělo jak letectvo tak pozemní jednotky. Problém bylo námořnictvo kde k chybám docházelo daleko méně. Zde pomohl Alan Turing, který pomocí svého stroje a hledání indíciích v podobě častých slov lámal pěti diskovou Enigmu. Kvůli zpětné kompatibilitě s Enigmou s 5 rotory, byli prolamovány i šifry Enigmy s 8 rotory. Podstatnou změnou v množství rozšifrovaných zpráv udělalo až ukořistění námořní knihy s klíči.

2. 2. 7. Data Encryption Standard (DES)

Data Encryption Standard (DES) byl symetrický blokový šifrovací algoritmus vyvinutý v 70. letech 20. století. Byl to vůbec první šifrovací standard schválený pro veřejné použití americkým Národním institutem pro standardy a technologie (NIST).

Proces šifrování pomocí DES zahrnuje několik kroků. DES provádí 16 „kol“ transformací na každý 64bitový blok dat. Každé kolo zahrnuje čtyři operace: rozšíření, substituci, permutaci a kombinaci s klíčem.

1. Rozšíření: Tato operace zahrnuje rozšíření 32bitového bloku dat na 48 bitů pomocí permutace a duplikace některých bitů.
2. 2.Substituce: Tato operace zahrnuje nahrazení 48bitového bloku dat šesti 4bitovými bloky pomocí předdefinovaných S-boxů. Každý S-box má 4 vstupní a 16 výstupních bitů, které jsou použity k nahrazení 6bitového vstupu 4bitovým výstupem.
3. Permutace: Tato operace zahrnuje přeuspořádání 32bitového bloku dat podle předdefinované permutační tabulky.

4. Kombinace s klíčem: Tato operace zahrnuje aplikaci podklíče na data pomocí operace XOR. Pro každé kolo je generován unikátní 48bitový podklíč.

Tento proces je opakován v každém kole, přičemž poslední kolo zahrnuje pouze operace substituce, permutace a kombinace s klíčem. Po 16 kolech je výsledný šifrovaný blok dat převeden zpět na 64bitový blok pomocí závěrečné permutace. [49]

I když byl DES považován za bezpečný v době svého vzniku, rychlý technologický pokrok znamenal, že v 90. letech se stával stále více zranitelným. DES byl napaden v roce 1998 skupinou počítačů, které dokázaly prolomit šifru za pouhých 22 hodin. To vedlo k vytvoření nových standardů, jako je 3DES a později AES.[50]

2. 2. 8. Triple Data Encryption Standard (3DES)

Triple DES, nebo 3DES, byl vyvinut jako zlepšená a bezpečnější varianta DES. Jak název napovídá, 3DES aplikuje DES třikrát na každý datový blok. 3DES funguje tak, že aplikuje DES třikrát na každý blok dat, což zvyšuje bezpečnost šifrování. Proces šifrování pomocí 3DES je následující:

1. V první fázi je DES použit pro šifrování dat pomocí prvního klíče. Tento proces je stejný jako u standardního DES, kde jsou data rozdělena do 64bitových bloků a na každý blok je aplikováno 16 kol transformací.
2. Ve druhé fázi je DES použit opět, ale tentokrát pro dešifrování výsledku první fáze pomocí druhého klíče. Tento proces je opakem procesu šifrování DES.
3. Nakonec je v třetí fázi DES použit pro šifrování výsledku druhé fáze pomocí třetího klíče. Tento proces je stejný jako v první fázi.

3DES může používat tři různé klíče, každý o velikosti 56 bitů, což znamená, že celková délka klíče může dosáhnout až 168 bitů. Nicméně, z důvodu tzv. meet-in-the-middle útoku je efektivní bezpečnost 3DES jen kolem 112 bitů. Meet-in-the-middle útok je typ útoku, kde útočník snaží se najít dva různé klíče, které by mohly být použity k šifrování a dešifrování stejných dat, což vede k snížení efektivní bezpečnosti šifrování. [51]

Ačkoliv je 3DES mnohem bezpečnější než DES, je také poměrně pomalý, což vedlo k vývoji nových a efektivnějších šifrovacích standardů, jako je AES. Navzdory tomu je 3DES stále používán v některých aplikacích, kde je rychlost šifrování méně důležitá než bezpečnost.[51]

2. 2. 9. Advanced Encryption Standard (AES)

Advanced Encryption Standard, neboli AES, je široce používaný symetrický šifrovací algoritmus, který byl v roce 2001 schválen jako standard pro šifrování dat americkým Národním institutem pro standardy a technologie (NIST). AES nahradil svého předchůdce, Data Encryption Standard (DES), který byl kritizován pro svou relativně nízkou úroveň zabezpečení.

AES je založený na algoritmu Rijndael a nabízí tři délky klíčů - 128, 192 a 256 bitů. Má blokovou strukturu, což znamená, že data jsou šifrována (nebo dešifrována) v pevně stanovených blokoch, v tomto případě 128 bitů.

Proces šifrování pomocí AES je poměrně komplexní a zahrnuje několik kroků. Nejprve jsou data rozdělena do 16-bajtových bloků. Každý blok je pak zpracován v několika „kolech“, jejichž počet závisí na délce klíče - 10 kol pro 128-bitový klíč, 12 kol pro 192-bitový klíč a 14 kol pro 256-bitový klíč. Každé kolo zahrnuje čtyři operace: SubBytes, ShiftRows, MixColumns a AddRoundKey.

1. SubBytes: Tato operace zahrnuje nahrazení každého bajtu dat bajtem z předdefinované tabulky, známé jako S-box.
2. ShiftRows: Tato operace zahrnuje posun bajtů v rámci každého bloku.
3. MixColumns: Tato operace zahrnuje kombinaci bajtů pomocí operací v tělese Galois.
4. AddRoundKey: Tato operace zahrnuje aplikaci klíče na data pomocí operace XOR.

Tento proces je opakován v každém kole, přičemž poslední kolo zahrnuje pouze operace SubBytes, ShiftRows a AddRoundKey.

Jedním z klíčových aspektů AES je jeho odolnost vůči většině známých útoků. AES byl podrobený intenzivnímu testování a analýze od svého uvedení a zatím nebyla nalezena žádná praktická metoda, která by umožnila prolomit jeho zabezpečení. Je to především díky jeho robustní struktuře a vysoké úrovni komplexity, která umožňuje AES poskytnout velmi silnou ochranu dat.

Nicméně, i přes tuto vysokou úroveň zabezpečení, je třeba mít na paměti, že žádný šifrovací algoritmus není 100% bezpečný. AES, stejně jako jakýkoli jiný šifrovací algoritmus, může být zranitelný vůči útokům, pokud je nesprávně používán, nebo pokud jsou klíče špatně spravovány. Proto je důležité při používání AES (nebo jakéhokoli jiného šifrovacího algoritmu) dodržovat nejlepší bezpečnostní praxe a provádět pravidelné bezpečnostní audity.[52]

2. 2. 10. Blowfish

Blowfish, symetrický blokový šifrovací algoritmus, byl vytvořen Bruce Schneierem v roce 1993 jako alternativa k existujícím šifrovacím standardům. K jeho popularitě přispělo zejména vynikající zabezpečení a rychlost.

Pracuje s bloky dat o velikosti 64 bitů a s klíčem o variabilní délce do 448 bitů. Algoritmus je založen na Feistelově síti, která rozděluje blok dat na dvě poloviny a provádí řadu transformací.

Proces šifrování pomocí Blowfish zahrnuje několik kroků. Každý 64bitový blok dat prochází šestnácti koly šifrování, kde je aplikována substituce a permutace.

1. Rozdělení bloku: Na začátku je 64bitový blok dat rozdělen na dvě 32bitové poloviny.
2. Substituce: V každém kole je levá polovina bloku XORována s podklíčem generovaným z klíče a poté je na ni aplikována funkce F, která zahrnuje několik operací, včetně substituce pomocí S-boxů.
3. Permutace: Po aplikaci funkce F je výsledek XORován s pravou polovinou bloku a poloviny jsou prohozeny.

Tento proces je opakován v každém kole, přičemž poslední kolo zahrnuje pouze operace substituce a permutace, ale bez prohození polovin. Po 16 kolech jsou obě poloviny bloku opět spojeny do jednoho 64bitového bloku.

Blowfish je známý svou rychlostí a efektivitou, zejména v systémech s omezenými zdroji. Jeho bezpečnost je zajištěna použitím velkého klíče až do 448 bitů a komplexními transformacemi v každém kole šifrování. [53]

Z hlediska zabezpečení je Blowfish stále považován za silnou šifru. Avšak jeho nástupce, Twofish, který je založen na podobném principu, ale pracuje s většími bloky dat a nabízí více kol šifrování, přináší vyšší bezpečnost. Díky své efektivitě a robustnosti je Blowfish často používán v situacích, kde není nutná nejvyšší možná úroveň šifrování, jako je například ochrana lokálních dat.

2. 2. 11. Twofish

Twofish je symetrický blokový šifrovací algoritmus, který zpracovává data ve 128bitových blocích a podporuje klíče o délce až 256 bitů. Stejně jako Blowfish, Twofish je založen na Feistelově síti a používá komplexní systém substituce a permutace v šestnácti kolech šifrování.

Proces šifrování pomocí Twofish je komplexní a zahrnuje několik kroků. Každý 128bitový blok dat prochází šestnácti koly šifrování, kde je aplikována substituce, permutace a lineární transformace.

1. Rozdělení bloku: Na začátku je 128bitový blok dat rozdělen na dvě 64bitové poloviny.
2. Substituce: V každém kole je jedna polovina bloku zpracována pomocí funkce g, která zahrnuje substituci pomocí S-boxů a lineární transformaci. Výsledek je poté XORován s druhou polovinou bloku.
3. Permutace: Po aplikaci funkce g a operace XOR jsou poloviny bloku prohozeny.

Tento proces je opakován v každém kole, přičemž poslední kolo zahrnuje pouze operace substituce, lineární transformace a permutace, ale bez prohození polovin. Po 16 kolech jsou obě poloviny bloku opět spojeny do jednoho 128bitového bloku.

Twofish je známý svou vysokou bezpečností a flexibilitou. Jeho bezpečnost je zajištěna použitím velkého klíče až do 256 bitů a komplexními transformacemi v každém kole šifrování. Flexibilita Twofish spočívá v jeho schopnosti efektivně zpracovávat různé velikosti klíčů a bloků dat.[54]

2. 2. 12. RC4

RC4, nebo také ARC4, je proudový šifrovací algoritmus, který byl vytvořený v roce 1987 Ronaldem Rivestem pro společnost RSA Security. Na rozdíl od blokových šifrovacích algoritmů, jako jsou DES, 3DES, Blowfish a Twofish, RC4 šifruje jednotlivé bajty dat místo bloků dat. Proces šifrování pomocí RC4 je následující:

1. Inicializace: Nejprve je vytvořena inicializační permutace bajtů (tzv. S-box) pomocí klíče, který může být v délce od 40 do 2048 bitů.
2. Generování pseudonáhodného proudu klíčů: Poté je generován pseudonáhodný proud klíčů pomocí permutace S-box. Tento proces zahrnuje permutaci S-boxu a výběr bajtů z permutace, které tvoří proud klíčů.
3. Šifrování dat: Nakonec je každý bajt dat kombinován s bajtem z proudu klíčů pomocí operace XOR, což vede k šifrovanému výstupu.

Ačkoli byl RC4 populární pro jeho rychlost a jednoduchost, bylo zjištěno, že trpí několika slabostmi, které mohou být využity k prolomení šifrování. Mezi tyto slabosti patří například nedostatečná náhodnost výstupního proudu, který může být využit k útoku. Přestože může být RC4 stále bezpečný při správném použití, většina moderních systémů se k němu již neobrací a preferuje algoritmy s vyšší úrovní zabezpečení, jako je AES nebo ChaCha20.[55]

2. 2. 13. ChaCha20

ChaCha20 je moderní proudový šifrovací algoritmus, který byl vytvořený v roce 2008 kryptografem Danielem J. Bernsteinem jako alternativa k RC4. ChaCha20 generuje pseudonáhodný proud klíčů, který se poté kombinuje s daty pomocí operace XOR, podobně jako RC4. Proces šifrování pomocí ChaCha20 je následující:

1. Inicializace: Na začátku je vytvořen 512bitový blok, který se skládá z konstanty, 256bitového klíče, 96bitového nonce (čísla použitého pouze jednou) a 32bitového čítače bloků.
2. Generování pseudonáhodného proudu klíčů: Poté je na blok aplikováno 20 kol transformací, které zahrnují operace jako je bitové rotace, sčítání a operace XOR. Výsledkem je pseudonáhodný proud klíčů.
3. Šifrování dat: Nakonec je každý bajt dat kombinován s bajtem z proudu klíčů pomocí operace XOR, což vede k šifrovanému výstupu.

ChaCha20 je považován za jeden z nejbezpečnějších šifrovacích algoritmů dneška. Jeho design je odolný proti útokům, které využívají slabosti v RC4. Díky své efektivitě a bezpečnosti je ChaCha20 široce používán v mnoha moderních aplikacích a protokolech, včetně TLS a IPsec. [56]

2. 2. 14. Salsa20

Salsa20 je proudový šifrovací algoritmus, který byl vytvořený Daniel J. Bernsteinem a poprvé prezentován veřejnosti v roce 2005. Salsa20 generuje pseudonáhodný proud klíčů a kombinuje ho s daty pomocí operace XOR. Proces šifrování pomocí Salsa20 je následující:

1. Inicializace: Na začátku je vytvořen 512bitový blok, který se skládá z konstanty, 256bitového klíče, 64bitového nonce (čísla použitého pouze jednou) a 64bitového čítače bloků.
2. Generování pseudonáhodného proudu klíčů: Poté je na blok aplikováno 20 kol transformací, které zahrnují operace jako je bitové rotace, sčítání a operace XOR. Výsledkem je pseudonáhodný proud klíčů.
3. Šifrování dat: Nakonec je každý bajt dat kombinován s bajtem z proudu klíčů pomocí operace XOR, což vede k šifrovanému výstupu.

Salsa20 je považován za bezpečný šifrovací algoritmus. Byl podroben mnoha kryptografickým analýzám, které potvrdily jeho odolnost proti známým útokům. I přes svou odolnost a efektivitu, Bernstein navrhl ChaCha20 jako jeho nástupce, který nabízí ještě vyšší úroveň zabezpečení. Salsa20 je stále používán v některých aplikacích, ale většina systémů dnes upřednostňuje ChaCha20.[57]

2. 2. 15. RSA

RSA je asymetrický kryptografický algoritmus, který byl vytvořen v roce 1977 Ronaldem Rivestem, Adi Shamirem a Leonardem Adlemanem. RSA funguje na základě matematické složitosti faktorizace velkých prvočísel. RSA bylo vytvořeno pro elektronický podpis a šifrování. Proces šifrování a dešifrování pomocí RSA je následující:

- Generování klíčů: Nejprve jsou vygenerovány dva velké prvočísla. Tyto prvočísla jsou poté vynásobeny dohromady a výsledek tvoří veřejný klíč. Soukromý klíč je poté generován pomocí dalších matematických operací, které zahrnují tyto prvočísla.
- Šifrování dat: Data jsou šifrována pomocí veřejného klíče. To zahrnuje převedení dat na číslo a poté výpočet jeho mocniny modulo veřejný klíč.
- Dešifrování dat: Data jsou dešifrována pomocí soukromého klíče. To zahrnuje výpočet mocniny šifrovaných dat modulo soukromý klíč.

RSA je považován za bezpečný, pokud je použit správně. Bezpečnost je zajištěna složitostí faktorizace velkých prvočísel. Avšak s narůstající výpočetní silou a vývojem kvantových počítačů může být bezpečnost RSA v budoucnosti ohrožena. Přesto se RSA stále hojně používá v mnoha bezpečnostních protokolech a systémech, včetně SSL/TLS. [58]

2. 2. 16. Kryptografie nad eliptickými křivkami (ECC)

Kryptografie nad eliptickými křivkami je typ asymetrické kryptografie, který je založený na algebraické struktuře eliptických křivek nad konečnými tělesy. ECC nabízí silné šifrování s kratšími klíči, což činí ECC efektivnější než tradiční metody, jako je RSA. Proces šifrování a dešifrování pomocí ECC je následující:

- Generování klíčů: Nejprve je vybrána eliptická křivka a bod na této křivce. Soukromý klíč je náhodné číslo a veřejný klíč je bod na křivce vynásobený soukromým klíčem.
- Šifrování dat: Data jsou šifrována pomocí veřejného klíče. To zahrnuje převedení dat na bod na eliptické křivce a poté výpočet jeho násobku pomocí veřejného klíče.
- Dešifrování dat: Data jsou dešifrována pomocí soukromého klíče. To zahrnuje výpočet násobku šifrovaného bodu pomocí soukromého klíče a poté převedení zpět na původní data.

Bezpečnost ECC je založena na problému logaritmu eliptické křivky, což je složité matematické problém. ECC je považována za jednu z nejbezpečnějších metod kryptografie a je široce používána v moderních bezpečnostních protokolech, včetně SSL/TLS a IPsec. Bezpečnost ECC je stále považována za odolnou proti útokům, včetně potenciálních budoucích útoků kvantových počítačů.[59]

2. 2. 17. Diffie-Hellman

Diffie-Hellman je metoda výměny klíčů, která umožňuje dvěma stranám vytvořit sdílený tajný klíč, který mohou použít pro šifrování a dešifrování zpráv. Byl vytvořen Whitfieldem Diffie a Martinem Hellmanem v roce 1976. Proces výměny klíčů pomocí Diffie-Hellman je následující:

1. Generování klíčů: Nejprve jsou vybrány dvě veřejně známé hodnoty: velké prvočíslo a generátor pro tuto grupu. Každá strana poté vygeneruje náhodné soukromé číslo a vypočítá veřejné číslo pomocí generátoru, soukromého čísla a prvočísla.
2. Výměna klíčů: Strany poté vymění svá veřejná čísla.
3. Vytvoření sdíleného tajného klíče: Každá strana poté vypočítá sdílený tajný klíč pomocí přijatého veřejného čísla, svého soukromého čísla a prvočísla.

Bezpečnost Diffie-Hellman je založena na složitosti diskrétního logaritmového problému. Diffie-Hellman je považován za bezpečný a je široce používán v mnoha bezpečnostních protokolech, jako je SSL/TLS. I přes svou bezpečnost, je důležité poznamenat, že Diffie-Hellman poskytuje pouze zabezpečenou výměnu klíčů a musí být použit v kombinaci s jinými kryptografickými algoritmy pro kompletní šifrování zpráv.[60]

2. 2. 18. ElGamal

ElGamal je asymetrický šifrovací algoritmus, který byl vytvořený v roce 1985 Taherem Elgamalem. ElGamal je založen na Diffie-Hellmanově výměnném klíči a poskytuje jak šifrování, tak digitální podpisy. Proces šifrování a dešifrování pomocí ElGamal je

následující:

- Generování klíčů: Nejprve je vybrán velký prvočíselný modul a generátor pro tuto grupu. Soukromý klíč je náhodné číslo a veřejný klíč je generátor umocněný na soukromý klíč modulo prvočíslo.
- Šifrování dat: Data jsou šifrována pomocí veřejného klíče. To zahrnuje generování náhodného čísla, výpočet dvou čísel pomocí veřejného klíče, generátoru a náhodného čísla, a poté kombinování těchto čísel s daty.
- Dešifrování dat: Data jsou dešifrována pomocí soukromého klíče. To zahrnuje výpočet inverzní hodnoty jednoho z čísel modulo prvočíslo, a poté výpočet původních dat pomocí tohoto inverzního čísla a druhého čísla.

Bezpečnost ElGamal je založena na složitosti diskrétního logaritmového problému. ElGamal je považován za bezpečný, pokud je použit správně. I přestože ElGamal není tak široce používán jako RSA nebo ECC, je stále uznáván za silný kryptografický algoritmus a je často používán v protokolech, jako je OpenPGP.[61][62]

2. 2. 19. Fraktální šifrování

Fraktální šifrování je novým a rozvíjejícím se polem v kryptografii, který se zaměřuje na použití fraktálů – komplexních, nekonečně se opakujících vzorů – jako základu pro šifrovací procesy. Jednou z nedávných aplikací fraktálního šifrování, která představuje slibný pokrok v oblasti šifrování obrazu, je systém popsaný v článku s názvem „A Novel Hybrid Secure Image Encryption Based on Julia Set of Fractals and 3D Lorenz Chaotic Map“.

Tento systém šifrování obrazu je založen na unikátní kombinaci fraktálního klíče, vygenerovaného pomocí Juliových množin, a trojrozměrné Lorenzovy chaotické mapy. V prvním kroku se proces zamíchání fraktálního klíče zmatí a zamíchá pixely vstupního obrazu. Následně trojrozměrná Lorenzova chaotická mapa provede difuzní proces, který zkreslí informace obsažené v jednotlivých pixelech, čímž zvýší celkovou bezpečnost šifrovaného obrazu.

Podle článku tento systém úspěšně prošel množstvím bezpečnostních testů a testů nízké výpočetní složitosti, čímž potvrdil svoji bezpečnost, robustnost a schopnost provádět šifrování v reálném čase. Tato zjištění naznačují, že fraktální šifrování, ačkoliv je stále relativně novým oborem, ukazuje slibný potenciál pro šifrování obrazu.

Nicméně je třeba zdůraznit, že se jedná o specifickou aplikaci fraktálního šifrování a celkové pole fraktálního šifrování je stále v rané fázi výzkumu a vývoje. Praktická efektivita a účinnost fraktálního šifrování mohou výrazně kolísat v závislosti na konkrétní aplikaci a implementaci.[63]

3. TECHNOLOGIE IMPLEMENTACE

Když byly osvětleny možnosti zabezpečení, je na čase věnovat se možnostem implementace. Ačkoliv zde nebudou představeny žádné konkrétní informace o implementaci, představíme si klíčové technologie, jež jsou pro tento účel nezbytné a vhodné. Při výběru technologií byl kladen důraz na jejich schopnost zajistit efektivní a bezpečné řešení, jež by bylo snadno použitelné a přizpůsobitelné pro širokou škálu uživatelů.

V následujících částech kapitoly se podrobně zaměříme na každou z těchto technologií, jejich výhody a nevýhody, a jak byly tyto technologie využity při vývoji správce hesel s otevřeným zdrojovým kódem.

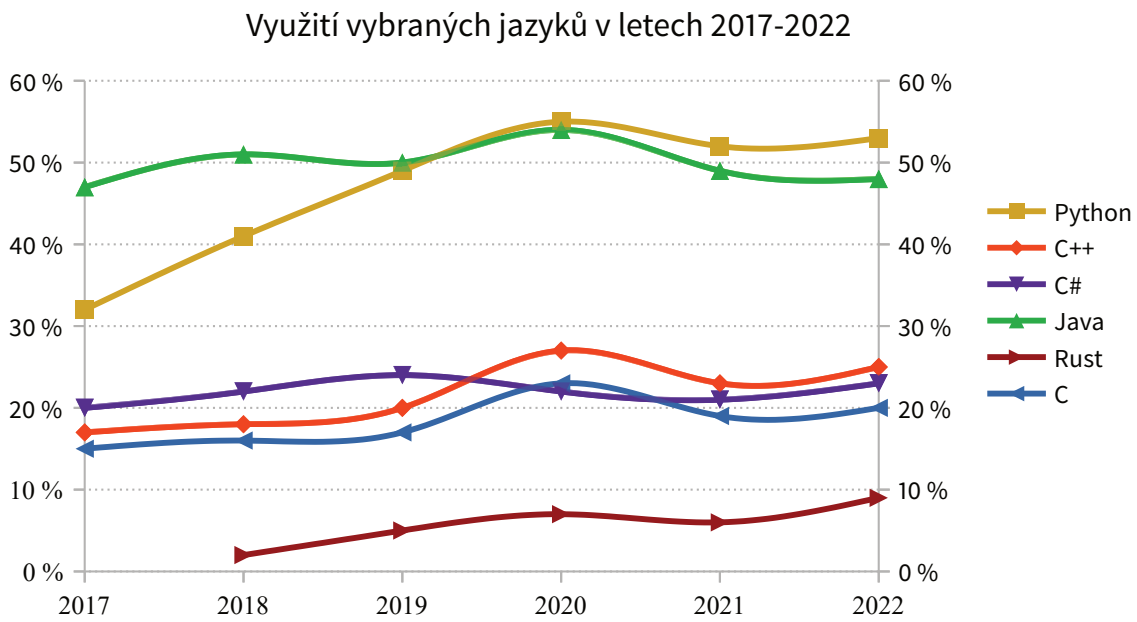
3. 1. Programovací jazyk

Pro volbu jazyka je nutné se zaměřit na účel aplikace. Jelikož se v tomto případě jedná o aplikaci, jenž pracuje s velice citlivými daty, je vhodné vynechat webové technologie.

Dalším kritériem pro volbu jazyka, jsou další plány s touto aplikací. Ačkoliv se tato práce věnuje jen základní implementaci, autor plánuje pokračovat v dalším rozvoji.
A v neposlední řadě, je nutné taky brát v potaz oblíbenost samotného jazyka v komunitě, přeci jenom se jedná o open-source projekt.

Jazyk / rok	2017	2018	2019	2020	2021	2022
Python	32 %	41 %	49 %	55 %	52 %	53 %
C#	20 %	22 %	24 %	22 %	21 %	23 %
Java	47 %	51 %	50 %	54 %	49 %	48 %
Rust	-	2 %	5 %	7 %	6 %	9 %
C++	17 %	18 %	20 %	27 %	23 %	25 %
C	15 %	16 %	17 %	23 %	19 %	20 %

Tabulka 2. Procentuální využití vybraných jazyků v letech 2017-2022 podle JetBrains Developer Ecosystem[67]



Obrázek 9. Graf zobrazující procentuální využití vybraných jazyků v letech 2017-2022 podle JetBrains Developer Ecosystem[67]

V následující tabulce jsou obecně příjemné vlastnosti vybraných jazyků. Tabulka nebere v úvahu specifické případy, a slouží pouze jako pomůcka při volbě jazyka implementace.

pozn. webová technologie musí fungovat v rámci webového prohlížeče, což dává prostor pro různé útoky. Mezi známé útoky na aplikace webové technologie patří například CSRF (Cross-Site Request Forgery)[64][65] a XSS (Cross-Site Scripting) [66]

	Python	C#	Java	Rust	C++	C
Platformní závislost	Nezávislý	Závislý (.NET)	Nezávislý (JVM)	Závislý	Závislý	Závislý
Výkon	Nízký až střední	Střední	Střední	Vysoký	Vysoký	Vysoký
Rychlost	Pomalejší	Střední rychlost	Střední rychlost	Střední rychlost	Velmi rychlý	Velmi rychlý
Hardwarové požadavky	Střední až vysoké	Střední	Střední až vysoké	Nízké až střední	Nízké až střední	Nízké
Přístup k programování	Procedurální OOP FP	Proceduráln OOP	Procedurální OOP	Procedurální OOP FP	Procedurální OOP	Procedurální
Úroveň abstrakce	Vysoká	Středně vysoká	Středně vysoká	Střední	Střední	Nízká
Správa paměti	Garbage collector	Garbage collector	Garbage collector	Vlastnictví a půjčování	Manuální	Manuální
Celikost komunity	Velká	Velká	Velká	Střední	Velká	Velká
Popularita[67]	Rostoucí	Stabilní	Stabilní	Rostoucí	Stabilní	Stabilní
Další vlastnosti	Obtížné propojení s GUI, obtížné tvoření spustitelného souboru	Silná podpora pro Microsoft, nízká podpora pro Linux		Delší čas kompilace, kontrola chyb během kompilace	Pokročilá správa paměti oproti jazyku C	

Tabulka 3. Obecné porovnání populárních jazyků

Z důvodů záměrů a preferencí autora na tuto práci navázat a pokračovat ve vývoji byl zvolen jazyk C++, pro svou obecně známou flexibilitu a výkonnost. Jelikož C++ vychází z jazyka C a je s ním plně kompatibilní může být využit i samotný jazyk C.

Jazyky C a C++ nepatří mezi nejoblíbenější jazyky. Mezi jejich největší výhody patří skutečnost, že poskytují programátorovi plnou kontrolu nad zdroji, Jediné omezení jenž mají, jsou limity operačního systému (popř. limity hardware).

To co je největší výhodou, je zároveň největší nevýhodou. Programátor musí být schopný pracovat se zdroji efektivně a nezapomínat již nepotřebné zdroje uvolňovat.

3. 2. Volba operačního systému

Při výběru vhodného operačního systému pro programování v jazyce C++ je nezbytné vzít v úvahu několik klíčových faktorů. Jednou z populárních možností, která si získala širokou přízeň, je využití Linuxu, konkrétně distribucí odvozených od Debianu. Toto rozhodnutí je motivováno předchozími zkušenostmi autora s těmito distribucemi a přináší několik předností, jež mohou napomoci k úspěšnému a efektivnímu programování.

Linux je proslulý svou stabilitou, spolehlivostí a bezpečností – klíčové vlastnosti pro vývoj aplikací. Distribuce odvozené od Debianu jsou známé svojí robustností a kvalitní správou balíčků, což zjednodušuje instalaci a aktualizaci nástrojů potřebných pro vývoj v C++.

Dalším důvodem je široká podpora vývojových nástrojů a prostředí pro programování v C++ na Linuxu. Existuje mnoho textových editorů, integrovaných vývojových prostředí (IDE) a nástrojů pro ladění, které usnadňují psaní, správu a ladění kódu. Oblíbené volby zahrnují Visual Studio Code, Clion, Emacs a GDB, které nabízejí pokročilé funkce pro vývojáře.

Poslední, avšak neméně důležitým důvodem, je aktivní komunita a podpora. Linuxová komunita je známá svou vášní pro sdílení znalostí a spolupráci. Existuje mnoho diskusních fór, komunitních projektů a zdrojů, které programátorům v C++ na Linuxu poskytují neocenitelnou pomoc a podporu.

3. 3. Výběr integrovaného vývojového prostředí (IDE)

Při vývoji aplikací v jazyce C++ je kromě výběru operačního systému rovněž zásadní volba vhodného integrovaného vývojového prostředí (IDE). IDE značně usnadňuje proces vývoje tím, že poskytuje sadu nástrojů a funkcí, které zjednodušují práci s kódem, jeho ladění a testování.

Poté, co bylo zváženo několik možností, bylo rozhodnuto o využití CLionu, což je výkonné a flexibilní IDE od firmy JetBrains určené pro vývoj v jazyce C++. Volba padla na CLion především z důvodu jeho vyspělých funkcí, které přispívají k efektivitě a produktivitě vývoje.

Jedním z hlavních důvodů pro výběr CLionu je jeho inteligentní našeptávač kódu, který výrazně zjednodušuje psaní kódu a zvyšuje rychlost práce. Tato funkce je schopná poskytnout návrhy kódu na základě kontextu a syntaxe jazyka C++, což usnadňuje orientaci v kódu a snižuje možnost chyb.

Dále CLion zaujal svou snadnou konfigurací, což je klíčové pro rychlé zahájení práce na nových projektech. Tato funkce umožňuje jednoduché nastavení vývojového prostředí podle specifických potřeb daného projektu.

Kromě toho CLion podporuje integraci různých technologií, což umožňuje využití různých nástrojů a knihoven přímo z IDE. To zahrnuje nástroje pro správu verzí, jako je Git, což je důležité pro efektivní správu kódu v týmových projektech.

Navíc, CLion nabízí nástroje pro práci s databázemi, což je nezbytné pro tento projekt. Tyto nástroje poskytují rychlý náhled do databází a ladění programu je podstatně rychlejší když nemusíme přepínat programy.

V neposlední řadě je třeba zmínit, že CLion je podporován aktivní a rychle reagující komunitou uživatelů a vývojářů, což zajišťuje pravidelné aktualizace, nové funkce a rychlou pomoc při řešení problémů.

Výše uvedené vlastnosti CLionu, spolu s jeho uživatelsky přívětivým rozhraním a vysokou stabilitou, činí z tohoto IDE ideální volbu pro vývoj aplikací v jazyce C++.

3. 4. Grafické uživatelské prostřeí (GUI)

Při tvorbě grafického uživatelského rozhraní (GUI) v jazyce C a C++ existuje několik známých knihoven a frameworků, jako jsou QT, GTK, Dear ImGui a wxWidgets. Kromě těchto knihoven a frameworků je také možné přistupovat přímo k hardwaru pomocí aplikačního rozhraní (API), například OpenGL nebo Vulkan. Avšak vývoj GUI založený pouze na aplikační vrstvě je časově náročný a složitý, proto pro prototyp aplikace není ideální se tímto způsobem zabývat.

WxWidgets byly vyřazeny z výběru kvůli rigidnímu přístupu k upravitelnosti designu, což je zřejmé z oficiálních ukázek knihovny prezentujících reálné programy z praxe.[68]

Zbývají tedy QT, GTK a Dear ImGui. Jako další kritérium byla zvolena snadnost přípravy vývojového prostředí pro daný framework nebo knihovnu. První bylo vyřazeno QT, protože pro získání instalačních souborů je nutné vyplňovat formuláře a čelit dalším nepříjemnostem. V souboji mezi GTK a Dear ImGui zvítězilo GTK, jelikož jeho instalace je jednodušší – stačí jeden příkaz v konzoli, který pokryje všechny závislosti, zatímco integrace Dear ImGui do projektu je o něco složitější.

3. 5. Standart programovacího jazyka

Při vývoji aplikací v jazyce C++ je rozhodnutí o výběru konkrétního standardu jazyka jednou z klíčových fází plánování projektu. Standard jazyka C++ se průběžně vyvíjí a každá nová verze přináší s sebou nové funkce, vylepšení a optimalizace. Po důkladném zvážení byl pro tento projekt vybrán standard C++23.

Standard C++23 byl vybrán z několika důvodů. Prvním z nich je podpora pro knihovnu `<filesystem>`, která nabízí pokročilé funkce pro práci se souborovým systémem. Tato knihovna umožňuje snadnou a efektivní manipulaci se soubory a adresáři, což je pro tento projekt důležité.[69]

Druhým důvodem je knihovna `<threads>`, která je nezbytná pro vývoj vícevláknových aplikací. V kontextu vývoje GUI aplikací je schopnost vytvářet a spravovat vlákna nezbytná pro dosažení hladké a reaktivní uživatelské zkušenosti.[70]

Třetím faktorem je příprava na budoucí přepis hlavičkových souborů do modulů. Moduly jsou jednou z nejvýznamnějších novinek v moderním C++, které nabízejí řadu výhod, jako je zlepšená časová efektivita kompilace a lepší izolace kódu. Ve standardu C++23 jsou základní knihovny již přepsány do modulů, což otevírá cestu k jejich širokému využití v budoucích projektech.[71]

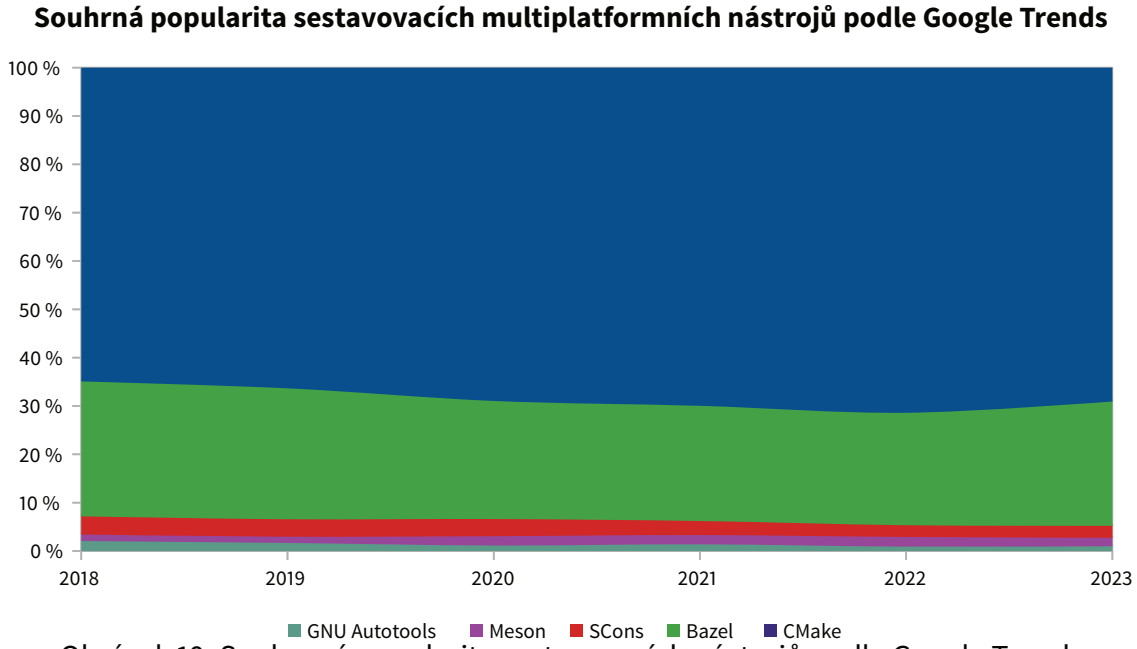
Je třeba poznamenat, že i když moduly přinášejí řadu výhod, jejich podpora v různých kompilátorech je v současné době stále v procesu implementace a může se lišit. Proto je důležité zvážit kompatibilitu s používaným kompilátorem při rozhodování o jejich použití v konkrétním projektu.[72]

3. 6. Výběr nástroje pro sestavování

Při vývoji softwaru je jedním z kritických aspektů volba vhodného nástroje pro sestavování kódu. Cílem je najít nástroj, který podporuje multiplatformní sestavování a je dobře dokumentovaný a podporovaný komunitou. Po pečlivé analýze byl pro tento projekt vybrán nástroj CMake.

CMake je open-source nástroj, který umožňuje jednoduché a efektivní sestavování kódu na různých platformách. CMake generuje nativní sestavovací systémy pro konkrétní platformy a tak podporuje multiplatformní sestavování.

Jedním z faktorů, který vedl k výběru CMake, byla jeho popularita v komunitě vývojářů. Podle statistik z Google Trends má CMake nejvyšší počet hledání v prohlížeči a na youtube, což ukazuje následující graf:[73]



Obrázek 10. Souhrnná popularita sestavovacích nástrojů podle Google Trends

Je třeba zmínit, že byly zvažovány i další nástroje pro sestavování, jako jsou GNU Autotools, Meson, SCons a Bazel. Tyto nástroje také poskytují podporu pro multiplatformní sestavování, ale žádný z nich nedosáhl takové popularity jako Cmake.

Je důležité zmínit, že nástroje jako Makefile a Visual Studio Projekt, ačkoli jsou široce používány, nejsou skutečně multiplatformní.

Takže na základě těchto faktorů byl CMake vybrán jako hlavní nástroj pro sestavování pro tento projekt.

3. 7. Způsobu uložení dat

Při návrhu softwaru je důležité pečlivě zvolit technologii pro ukládání dat. V této práci byla zvolena databáze SQLite z důvodů její multiplatformnosti, výkonnosti a jednoduché implementace. [74]

3. 7. 1. Multiplatformnost

SQLite je systém pro správu relačních databází (RDBMS) obsažený v malé knihovně C, což z něj činí výhodnou volbu pro multiplatformní aplikace. SQLite podporuje širokou škálu operačních systémů, včetně různých verzí Windows, Linuxu a macOS, a to z něj činí flexibilní volbu pro vývoj na různých platformách. [75]

3. 7. 2. Výkonnost a nenáročnost

SQLite je také ceněn pro svou výkonnost a nenáročnost na systémové zdroje. I přes svou kompaktní velikost SQLite poskytuje plně funkční relační databázi s bohatou sadou funkcí. SQLite nabízí vše, co je potřeba pro efektivní manipulaci s daty, aniž by bylo nutné vynakládat značné množství systémových zdrojů.

3. 7. 3. Jednoduchá implementace

SQLite je navržen tak, aby byl snadno implementovatelný. Nevyžaduje instalaci

databázového serveru ani konfiguraci, což výrazně zjednodušuje proces nasazení. Data jsou uložena v jediném souboru na disku, který je snadno přenosný mezi různými systémy. To znamená, že při vývoji nebo nasazení aplikace nejsou nutné žádné dodatečné kroky pro správu databáze. [75]

Ve výsledku SQLite nabízí robustní, efektivní a flexibilní řešení pro ukládání dat, které se hodí pro široké spektrum aplikací. Jeho jednoduchost a schopnost fungovat na různých platformách ho činí ideálním pro tento projekt.

3. 8. Přenositelnost dat

Při implementaci softwarového řešení této práce bylo rozhodnuto pro použití SQLite jako primárního úložiště pro data. Využili jsme jeho potenciálu jako kompaktního, serverless, self-contained systému, který je schopen efektivně manipulovat s daty v rámci naší aplikace.

Náš systém se skládá ze dvou hlavních druhů databází. Prvním je centrální SQLite databáze, jejímž hlavním úkolem je autentifikace uživatele. Tato databáze obsahuje pouze hash hodnoty, která je vygenerována na základě specifické kombinace jména a hesla uživatele. Tato metoda poskytuje vysokou úroveň zabezpečení, jak je podrobněji vysvětleno v kapitole 4.

Druhým druhem databáze, který naše aplikace využívá, je individuální SQLite databáze pro každého uživatele. Tato databáze slouží k ukládání šifrovaných dat uživatele. Jediný prvek, který se nešifruje, je ID dat. Toto rozhodnutí bylo učiněno z důvodu zachování základní úrovně přístupnosti dat, zatímco zároveň byla udržena maximální možná úroveň zabezpečení. O této problematice se více dozvíte v níže.

II. PRAKTICKÁ ČÁST

4. POPIS IMPLEMENTACE

V praktické části jsou reprodukovány kódy aplikace. Tyto kódy jsou rozšířeny o větší množství komentářů než v samotné aplikaci. Toto rozhodnutí bylo učiněno tak, aby kódu mohl porozumět i čtenář, který není obeznámen s jazykem C++. Komentáře jsou psány v angličtině, což odpovídá konvenci popisování kódu pouze pomocí ASCII znaků a obecné konvenci užívání anglického jazyka v kódu.

Z důvodu lepší správy verzí, zlepšení udržitelnosti a také kvůli rozdílnému licencování bylo rozhodnuto, že aplikace je implementována jako šifrovací jádro a zbytek programu s uživatelským rozhraním. Vzhledem k tomu, že pro grafické rozhraní využíváme knihovnu GTK, která je pod licencí GPL, musíme tuto licenci dodržet i pro odvozenou aplikaci. Nicméně jádro je pod licencí MIT, což umožňuje jeho využití i v komerčních projektech.

4. 1. Jádro aplikace

Pro jádro aplikace byla vytvořena následující adresářová struktura:

RainTextCore – Kořenový adresář jádra.

- **include** – adresář pro hlavičkové soubory.
  - **core** – adresář pro hlavičkové soubory, jenž **budou** používány externě.
  - **utils** – adresář pro hlavičkové soubory, jenž **nebudou** používány externě.
    - **cypher** – adresář pro hlavičkové soubory šifrovacích algoritmů.
- **lib** – adresář pro externí knihovny.
  - **argon2** – adresář knihovny argon2, upraveno pro CMake.[76]
  - **scrypt** – adresář knihovny scrypt, upraveno pro CMake.[77]
- **src** – adresář pro zdrojové soubory.
  - **core** – adresář pro zdrojové soubory, jenž **budou** používány externě.
  - **utils** – adresář pro zdrojové soubory, jenž **nebudou** používány externě.
    - **cypher** – adresář pro zdrojové soubory šifrovacích algoritmů.
- **tests** – adresář pro testování projektu.
  - **lib** – adresář pro knihovnu google testů.

Jak již bylo uvedeno v části nástroj pro sestavování, v kapitole **3. Technologie implementace**, pro projekt je využíván nástroj CMake. Proto bylo nutné vytvořit soubory CMakeList.txt v následujících cestách:

1. `./RainTextCore/CMakeList.txt` – Hlavní CMake soubor pro projekt jádra.
   - Požaduje verzi CMake 3.22 a vyšší.
   - Po kompilátoru vyžaduje dodržení standardu C++ 23.
   - Do kódu zahrnuje `./include`, kde jsou uloženy všechny hlavičkové soubory.
   - Přidává podsložky `./lib`, `./src` a `./tests`.
   - Kód pro své fungování vyžaduje najití knihovny crypto++ a threads.[78]
2. `./lib/CMakeList.txt` – Soubor pro připojení knihoven (pro převod Makefile knihoven na CMake byl využit chatGPT)
   - Do kódu zahrnuje./argon2 a./scrypt.
   - Určuje, kde najít hlavičkové soubory pro projekty „scrypt“ a „argon2“.
3. `./src/CMakeList.txt` – Soubor pro řízení sestavení zdrojových souborů.
   - Přidává soubory zdrojové soubory jádra do sestavení jako statickou knihovnu.
   - Nastavuje cestu pro vytvořené spustitelné soubory.

4. `./tests/CMakeList.txt` – Soubor pro řízení testů.
   - Přidání zdrojových kódů testů.
   - Nastavení cesty spustitelného souboru.
   - Nastavení cesty hlavičkových souborů.
   - Připojení Google testů.
   - Nastavení projektu jako knihovny.
5. `./tests/lib/CMakeList.txt` – Soubor pro získání Google testů z GitHubu. [78]

Každý projekt by měl mít jednotné formátování kódu, pro tyto účely byla zvolena technologie clang-format s nastaveným formátem pro Google. Toto nestavení je uloženo v kořenovém adresáři v souboru.clang-format.[80]

4. 1. 1. Implementace jádra

Jádro aplikace obsahuje jedinou třídu určenou pro výstup do dalších aplikací, která má pouze čtyři veřejné metody: konstruktor, šifrování, dešifrování a změna vstupních dat.

4. 1. 2. Konstruktor

Konstruktor slouží k přijetí všech potřebných dat, tj. počet iterací pro šifrování, dat a vstupního klíče, který musí být delší než trojnásobek délky klíče nejdelšího klíče ze seznamu implementovaných šifer. Důvodem je, že při inicializaci každé šifry je vložený klíč rozložen na klíče odpovídající délky; náhodně je vybrán klíč pro šifrování, poté je jiný klíč náhodně vybrán jako základ hashe, který slouží jako inicializační vektor, a nakonec je náhodně vybrána i jiná část klíče pro sůl tohoto hashe. Následující kód reprezentuje konstruktor třídy RainTextCore:

```
// Define the namespace rain_text_core
namespace rain_text_core {

// Constructor for RainTextCore, takes in the number of iterations for
// encryption/decryption, key, and text
RainTextCore::RainTextCore(uint16_t iterations, const std::vector<uint8_t>& key,
                           const std::vector<uint8_t>& text)
    : iterations_(iterations), key_(key), text_(text) {
  // Check if the number of iterations is 0. If it is, throw an exception because
  // at least one iteration is necessary
  if (iterations == 0) {
    throw std::invalid_argument(
        "You must enter at least one iteration, we recommend at least 10");
  }

  // Check if the key size is less than 96. If it is, throw an exception as
  // it's less than the minimum required length
  if (key.size() < 96) { // 256 bit - the longest key; 3 * 256 / 8
    auto err = std::string(
        "The key must be a vector greater than or equal to 128.\nYour vector "
        "has only " +
        std::to_string(key.size()) + " elements");
```

Pozn. Google testy jsou stahovány při kompilaci, je tudíž nutné mít aspoň při první kompilaci přístup k internetu

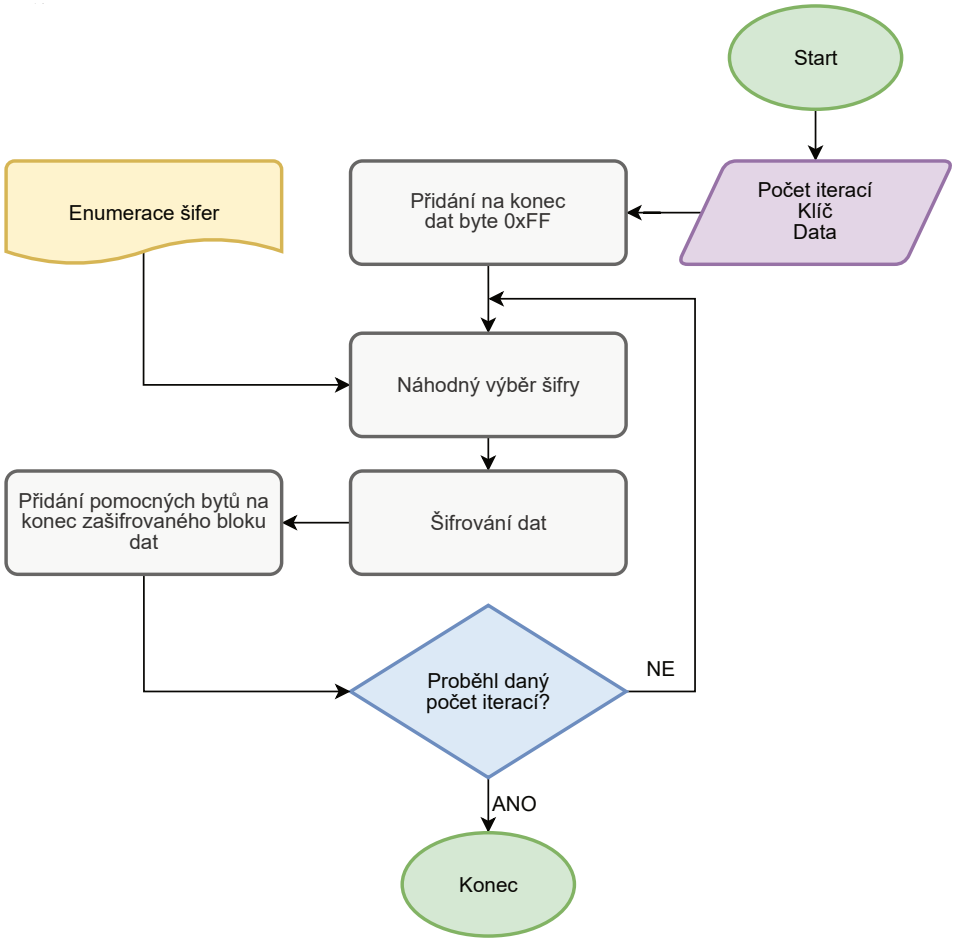
```
    throw std::invalid_argument(err.data());
  }

  // Check if the text vector is empty. If it is, throw an exception because
  // the text vector should not be empty
  if (text.empty()) {
    throw std::invalid_argument("The text vector must not be empty");
  }
}
} // namespace rain_text_core
```

4. 1. 3. Šifrovací metoda

Před zahájením samotného šifrování je nezbytné zajistit přítomnost specifického údaje, který nám poslouží v procesu dešifrace. Tímto údajem je byte '0xFF', který je vložen na konci otevřeného textu.

Metoda šifrování operuje s vstupní hodnotou počtu iterací, jež slouží k inicializaci cyklu, který opakuje proces šifrování tak dlouho, dokud počet iterací cyklu nedosáhne hodnoty vstupního počtu iterací. Proces uvnitř tohoto cyklu je takový, že náhodně vybereme jednu z předem definovaných šifer.

Následující vývojový diagram ilustruje postup šifrování dat:



Obrázek 11. Vývojový diagram šifrovací metody

Tento vývojový diagram symbolizuje funkční průběh kódu, který bude prezentován níže:

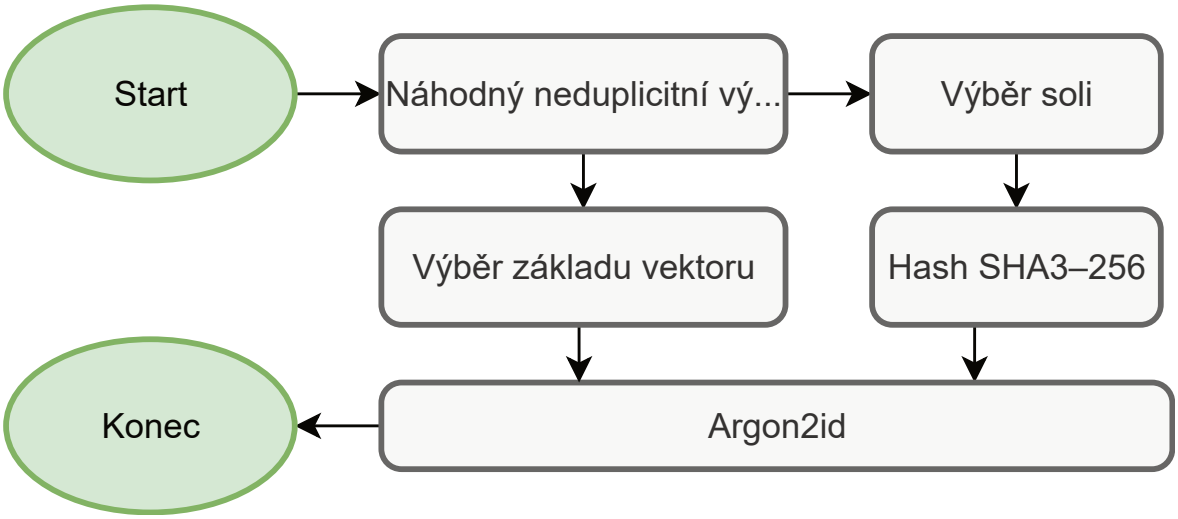
```
// Define the namespace rain_text_core
namespace rain_text_core {

// Define an enumeration for different cipher types
enum CipherType { AES = 0, TWOFISH, CHACHA20, NUM_CIPHERS };

// Method to create a specific cipher object, returns a unique_ptr
// to the created cipher
std::unique_ptr<rain_text_core::ICipher> RainTextCore::CreateCipher(
    CipherType type, const std::vector<uint8_t>& text) {
  switch (type) {
    case AES: // If AES is requested
      // Create an AES cipher
      return std::make_unique<rain_text_core::Aes>(0, key_, text);
    case TWOFISH: // If Twofish is requested
      // Create a Twofish cipher
      return std::make_unique<rain_text_core::Twofish>(1, key_, text);
    case CHACHA20: // If ChaCha20 is requested
      // Create a ChaCha20 cipher
      return std::make_unique<rain_text_core::ChaCha20>(2, key_, text);
    default: // If an invalid cipher type is requested
      throw std::invalid_argument("Invalid cipher type"); // Throw an error
  }
}

// Method to encrypt the text. The encrypted text is passed out
// via the 'output' parameter.
void RainTextCore::Encrypt(std::vector<uint8_t>& output) {
  // Append 0×FF to the text
  text_.push_back(0×FF);
  // Generate a uniform distribution over the range of valid cipher types
  std::uniform_int_distribution<> dist(0, NUM_CIPHERS - 1);
  // Declare a temporary storage vector
  std::vector<uint8_t> tmp;
  // Loop over the desired number of encryption iterations
  for (uint16_t i = 0; i < iterations_; ++i) {
    // Clear the temporary storage vector
    tmp = std::vector<uint8_t>();
    // Generate a random cipher type
    auto random_cipher_type = static_cast<CipherType>(dist(random_));
    // Create the selected cipher
    std::unique_ptr<rain_text_core::ICipher> selected_cipher =
        CreateCipher(random_cipher_type, text_);
    // Encrypt the text using the selected cipher and store the result in tmp
    selected_cipher->Encrypt(tmp);
```

```
    // Update the text to be the encrypted text
    text_ = tmp;
  }
  // Set the output to be the final encrypted text
  output = tmp;
  // Also update the stored text to be the final encrypted text
  text_ = std::move(tmp);
}
} // namespace rain_text_core
```

Každá šifra je konfigurována tak, aby dlouhý vstupní klíč rozdělila na klíče, které odpovídají jejímu vstupu. Jeden z těchto klíčů je použit jako klíč pro šifrovací funkci, další klíč je zvolen jako sůl pro tvorbu inicializačního vektoru, který je poté prohnán skrze hashovací funkci SHA3–512. Nakonec je vybrán klíč pro základ inicializačního vektoru. Klíč pro inicializační vektor a hash reprezentující sůl jsou prohnány skrze funkci argon2id a výsledkem tohoto procesu je inicializační vektor.

Po provedení šifrování se k zašifrovanému textu přidá byte symbolizující základ pro sůl, základ pro inicializační vektor, klíč a index šifry, který je předán konstruktoru dané šifry za účelem lepší udržitelnosti a možnosti modifikace kódu.

Následující vývojový diagram ukazuje postup tvorby inicializačního vektoru:



Obrázek 12. Vyvojový diagram tvorby inicializačního vektoru

Tento vývojový diagram symbolizuje funkční průběh kódu, který bude prezentován níže, na konkrétním případu užití v šifře AES:

```
// Define the namespace rain_text_core
namespace rain_text_core {

void Aes::ComputeInitVector() {
 if (!key_index_ || !init_vector_index_ || !pre_salt_index_) {
  throw std::runtime_error(
    "key_index_ or init_vector_index_ or pre_salt_index_ missing");
 }
```

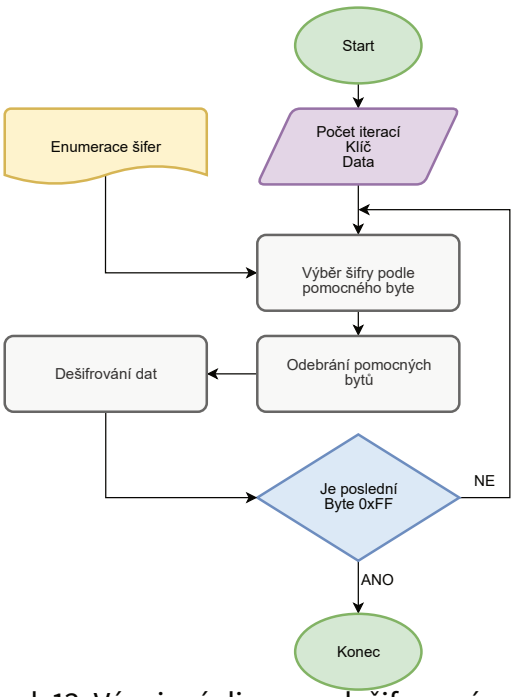
```
 auto pre_init_vector = splited_keys_[*init_vector_index_];
 auto pre_salt = splited_keys_[*pre_salt_index_];
 CryptoPP::SHA3_256 salt_hash;
 salt_hash.Update((CryptoPP::byte *)pre_salt.data(), pre_salt.size());
 std::string salt_text;
 salt_text.resize(salt_hash.DigestSize());
 salt_hash.Final((CryptoPP::byte *)&salt_text[0]);

 auto salt = std::vector<uint8_t>(salt_text.begin(), salt_text.end());

 argon2id_hash_raw(10, 1 << 10, 4, pre_init_vector.data(),
          pre_init_vector.size(), salt.data(), salt.size(),
          init_vector_, 16);
}

} // namespace rain_text_core
```

4. 1. 4. Dešifrovací metoda

Dešifrovací metoda nepotřebuje ke svému fungování znát počet iterací, protože při šifrování se na začátku nakonec textu přidá byte 0xFF, což znamená, že dešifrovací smyčka běží, dokud nenarazí na tento byte. Algoritmus ví, kterou funkci a jaké hodnoty z klíče použít pro dešifrování díky čtyřem posledním bytům. Tyto byty jsou seřazeny od konce následujícím způsobem:

1. index šifrovacího algoritmu
2. index klíče
3. index pro základ inicializačního vektoru
4. index pro sůl inicializačního algoritmu

Tyto informace je třeba před dešifrováním dat odstranit. Princip dešifrování je znázorněn v následujícím vývojovém diagramu:



Obrázek 13. Vývojový diagram dešifrovací metody

Tento vývojový diagram symbolizuje funkční průběh kódu, který bude prezentován níže:

```
// Define the namespace rain_text_core
namespace rain_text_core {

// Method to decrypt the text with the specific cipher from the CipherType
enum
void RainTextCore::Decrypt(std::vector<uint8_t>& output) {
 std::vector<uint8_t> tmp; // Temporary storage for decrypted text

 // While the end-of-text marker is not reached
 while (text_.back() != 0xFF) {
  tmp = std::vector<uint8_t>(); // Reset the temporary storage

  // Retrieve the cipher type from the last byte of the encrypted text
  auto cipher_index = static_cast<CipherType>(text_.back());

  // Create the cipher based on the retrieved type
  std::unique_ptr<rain_text_core::ICipher> selected_cipher =
    CreateCipher(cipher_index, text_);

  // Decrypt the text using the selected cipher and store the result in tmp
  selected_cipher->Decrypt(tmp);

  // Update the original text to be the decrypted text for next iteration
  text_ = tmp;
 }

 // Remove the end-of-text marker from the decrypted text
 tmp.pop_back();

 // Update the original text and output with the final decrypted text
 text_ = tmp;
 output = text_;
}
} // namespace rain_text_core
```

4. 1. 5. Metoda pro změnu vstupních dat

Tato metoda je prakticky totožná s částí v konstruktoru, jenž přiřazuje proměnou `text_`, nejdříve se ověří že vektor s textem není prázdný a pokud není, změní se proměnná `text_`. V případě že vstupní vektor je prázdný, vyvolá se výjimka špatného argumentu metody.

4. 1. 6. Zhodnocení principů funkčnosti

Omezení délky klíče je 255-násobek délky nejkratšího klíče. Zároveň je možné zahrnout až 254 šifrovacích algoritmů, protože poslední hodnota je rezervována pro ukončení šifrování.

Takový systém má však nevýhodu ve formě narůstající velikosti během šifrování, jelikož každá iterace přidá 4 byty a první 5 bytů.

4. 2. Grafické uživatelské prostředí

Pro grafické rozhraní aplikace byla zvolena odlišná souborová struktura, která lépe vyhovuje potřebám tvorby aplikace, než struktura používaná pro knihovny. Tato struktura se skládá z následujících částí:

RainText – Kořenový adresář projektu.
- assets – Adresář pro vizuální části kódu
  - icons – Adresář pro všechny ikony.
    - iconsax – Adresář pro ikony z projektu iconsax.[81]
- lib – Adresář pro externí knihovny.
  - md5 – Knihovna pro tvorbu názvů databází, upraveno pro Cmake.[82]
  - rain_text_core – knihovna šifrovacího jádra knihovny.
  - sha3 – Knihovna pro sha3 algoritmus, upraveno pro Cmake.[83]
- src – Adresář pro zdrojové soubory.
  - headers – Adresář pro hlavičkové soubory.
- tests – Adresář pro testování projektu.
  - lib – adresář pro knihovnu google testů.

Jak již bylo uvedeno v části nástroj pro sestavování, v kapitole **3. Technologie implementace**, pro projekt je využíván nástroj CMake. Proto bylo nutné vytvořit soubory CMakeList.txt v následujících cestách:

1. `./CMakeList.txt`
   - Požaduje verzi Cmake 3.22 a vyšší.
   - Po kompilátoru vyžaduje dodržení standardu C++ 23.
   - Přidává podsložky./lib a./tests.
   - Kód pro své fungování vyžaduje najití knihovny SQLite3 a threads.[70][74]
2. `./lib/CMakeList.txt` – Soubor pro připojení knihoven (pro převod Makefile knihoven na CMake byl využit chatGPT)
   - Do kódu zahrnuje `./md5`, `./rain_text_core` a `./sha3`.
3. `./tests/CMakeList.txt`
   - Přidání zdrojových kódů testů
   - Nastavení cesty spustitelného souboru
   - Nastavení cesty hlavičkových souborů
   - Připojení Google testů
   - Nastavení projektu jako knihovny
4. `./tests/lib/CMakeList.txt` – Soubor pro získání Google testů z GitHubu. [78]

Podobně jako v případě jádra aplikace, i zde využíváme nástroje clang-format k definování pravidel formátování kódu, která jsou v souladu s manuálem Google.

Tento segment projektu je značně složitější. Proces aplikace pramení z třídy `Gui`, jež je zodpovědná za vykreslování uživatelského grafického prostředí. Z této třídy jsou následně volány další třídy a jmenné prostory. Abychom mohli vhodně popsat fungování celé aplikace, budeme se nejprve věnovat vzhledu jednotlivých oken a jejich interaktivitě. Po tomto podrobném seznámení se s okny následně přejdeme k analýze funkcionality, která za těmito okny leží.

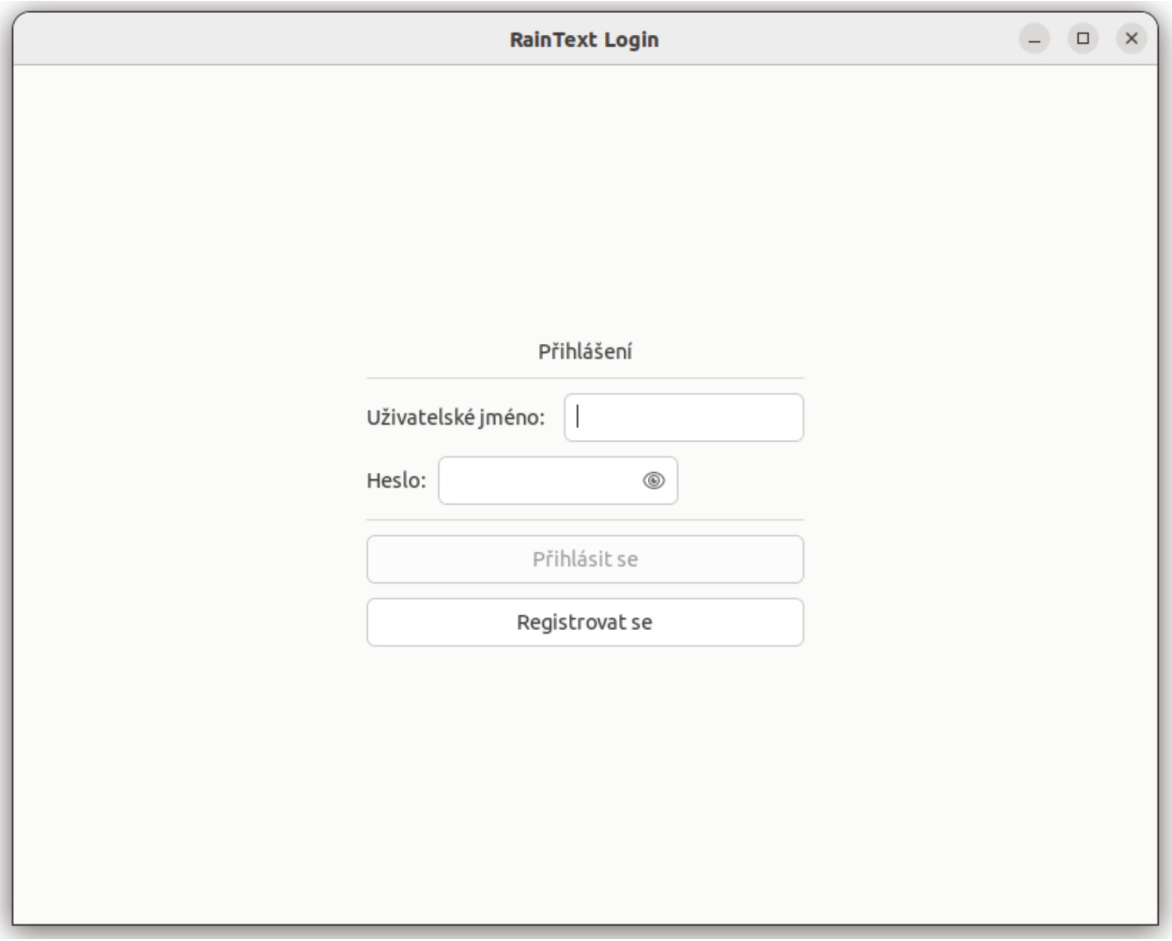
4. 2. 1. Přihlašovací okno

Toto okno se zobrazí po spuštění aplikace a obsahuje jednoduchý přihlašovací formulář s políčkem pro uživatelské jméno a políčkem pro heslo. Dále také dvě tlačítka, jedno pro přihlášení a druhé pro registraci uživatele.

Tlačítko pro přihlášení nelze stisknout, dokud nejsou vyplněna obě pole. Pokud po stisknutí tlačítka nalezne systém uživatele v databázi úspěšně, ověří jeho přihlašovací údaje, uživatel je přihlášen do hlavního okna aplikace. Pokud uživatel není nalezen, pod heslem se objeví chybová hláška, která informuje o neúspěšném přihlášení.

Poslední tlačítko umožňuje uživateli přepnout do registračního okna, kde se může zaregistrovat jako nový uživatel.

Na vzhled přihlašovacího okna se můžete podívat na následujícím obrázku.



Obrázek 14. Screenshot prihlašovacího okna aplikace

4. 2. 2. Registrační okno

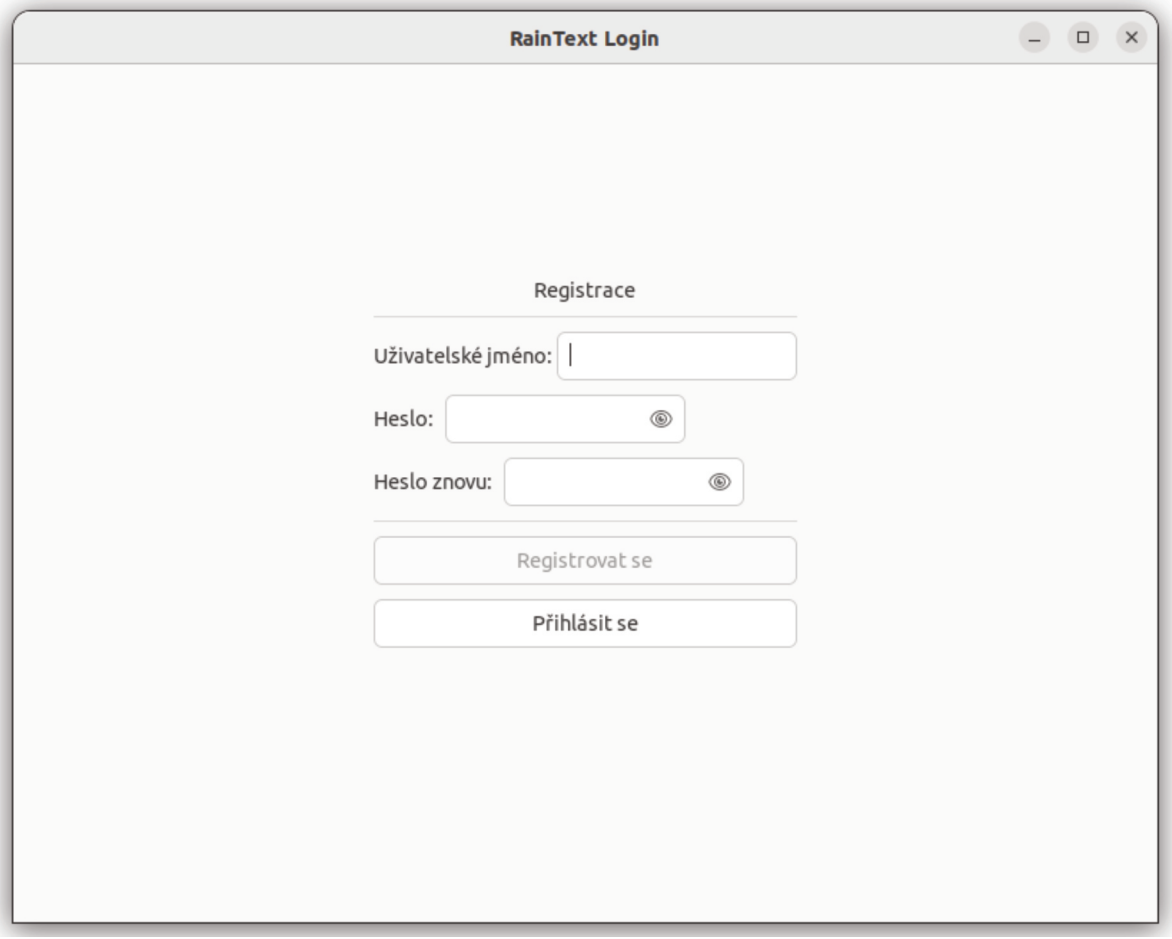
V registračním okně se nachází tři pole k vyplnění, uživatelské jméno, heslo a potvrzení hesla. Dále jsou zde dvě tlačítka, jedno pro registraci a druhé pro přepnutí zpět do okna přihlášení.

Registrační tlačítko se aktivuje pouze tehdy, když je vyplněné uživatelské jméno a hesla jsou shodná, zároveň musí být delší než 7 znaků.

Po stisknutí tlačítka registrace mohou nastat dva scénáře: První scénář nastane, když

uživatel v systému ještě neexistuje. V tomto případě se nový uživatelský účet vytvoří a uživateli se zobrazí hlavní okno aplikace. Druhý scénář nastane, pokud uživatelský účet se zadaným uživatelským jménem již existuje. V takovém případě se uživateli zobrazí chybová hláška, která upozorňuje na existenci účtu se stejným jménem. Uživatel poté může zvolit jiné uživatelské jméno a zkusit registraci znovu.

Opět se můžete podívat na obrázek, jak vypadá registrační okno.



Obrázek 15. Screenshot registračního okna

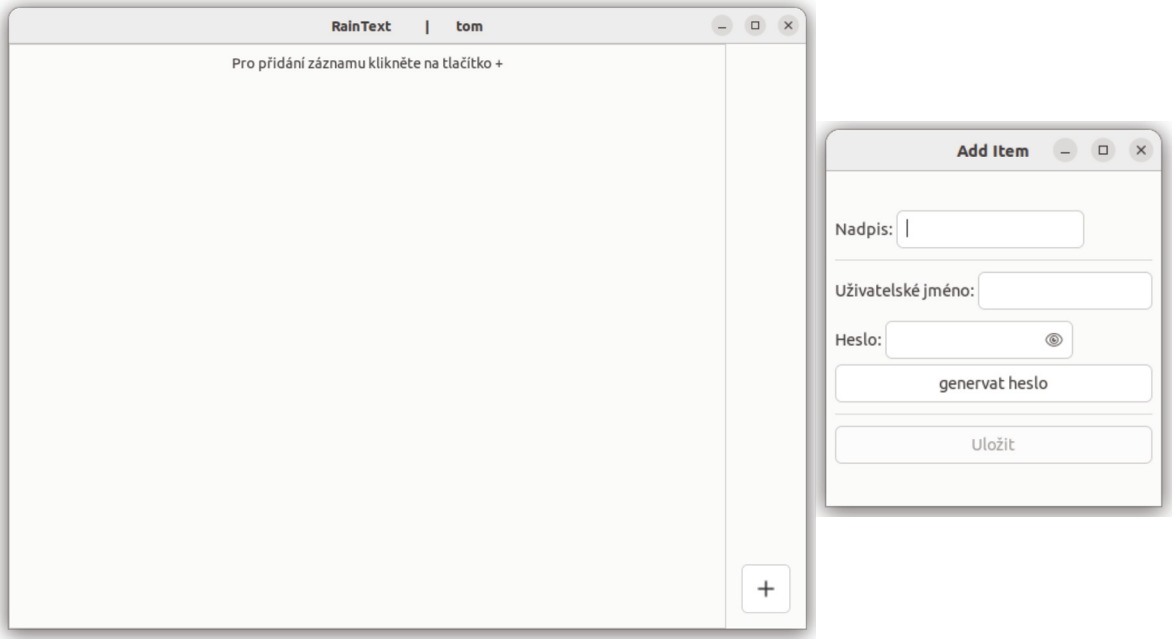
4. 2. 3. Hlavní okno

Pokud uživatel ve své databázi nemá uložený žádný záznam, v hlavním okně se mu zobrazuje pouze hláška s navigující uživatele ke stisknutí tlačítka „+“ pro vytvoření záznamu.

Tlačítko plus se nachází v pravém dolním rohu a otvírá modální okno pro přidání nového hesla.

Pokud uživatel již má uložená hesla, zobrazují se mu ve formátu názvu, uživatelského jména a hesla, které je bez zásahu uživatele reprezentováno pouze jako řada zástupných znaků. Tento záznam dále obsahuje tlačítko na odhalení hesla a tlačítko na odstranění záznamu.

To jak vypadá modální okno pro přidání nového záznamu spolu s hlavním oknem, zobrazuje následující obrázek.



Obrázek 16. Hlavní okno a okno pro přidání záznamu

V případě, že uživatel chce přidat nový záznam, stiskne tlačítko plus a otevře se modální okno. Zde může vyplnit název, uživatelské jméno a heslo pro nový záznam. Poté stiskne tlačítko "Uložit" a nový záznam se uloží do databáze. Nový záznam se poté zobrazí v hlavním okně aplikace spolu s ostatními uloženými záznamy.

Uživatel může také odstranit záznam stisknutím tlačítka na odstranění záznamu. Po stisknutí tlačítka se záznam odstraní z databáze a již se nezobrazuje v hlavním okně aplikace.

4. 2. 4. Funkcionalita přihlašovacího okna

Jak bylo již dříve naznačeno, přihlašovací okno z programátorské perspektivy nepředstavuje zvláště fascinující prvek. Zahrnuje callback funkci, jenž je spuštěna při vyplňování přihlašovacího jména a hesla. Tato funkce ověřuje, zda obě pole obsahují minimálně jeden znak – pokud ano, je aktivováno tlačítko pro přihlášení.

Nicméně, co se odehrává po kliknutí na tlačítko přihlášení je mnohem zajímavější. Je zde vyvolána metoda třídy `MainDatabase` pro přihlášení uživatele. Tato metoda kontroluje, zda se zadané údaje shodují s údaji v databázi. Pokud ano, uživatel je úspěšně přihlášen. V případě neshody se pod polem „heslo“ zobrazí chybová hláška. Podrobněji se na fungování třídy `MainDatabase` podíváme v kapitole **4. 2. 7. Třída MainDatabase**.

4. 2. 5. Funkcionalita registračního okna

Podobně jako přihlašovací okno, i registrační formulář disponuje callback funkcí, která ovládá aktivaci tlačítka pro registraci. Tato funkce kontroluje, zda uživatelské jméno obsahuje minimálně jeden znak, dále ověřuje, zda jsou heslo a potvrzení hesla shodné a současně mají délku minimálně 8 znaků.

Po kliknutí uživatelem na aktivované tlačítko pro registraci je zavolána metoda třídy MainDatabase pro registraci uživatele. Tato metoda pro daného uživatele vytvoří

Pozn. Instance třídy `MainDatabase` je vytvořena v okamžiku kliknutí na tlaíčtko přihlášení. Obdobně tomu tak je u registrace. Třída v zápětí zaniká v okamžiku ověření uživatele.

databázi a jeho záznam uloží do centrální databáze programu. Blíže opět v kapitole **4. 2. 7. Třída MainDatabase**.

4. 2. 6. Funkcionalita hlavního okna

Hlavní okno slouží primárně pro přidávání a odebírání záznamů do databáze. Při načítání tohoto okna se vytvoří instance třídy `UserDatabase`, jsou získány veškeré záznamy z uživatelské databáze, tyto záznamy jsou následně dešifrovány a zobrazeny v předem určeném prostoru pro zobrazování záznamů.

Po kliknutí na tlačítko „+“ může uživatel, kromě ručního zadání hesla pro záznam, vygenerovat heslo. Tato funkce volá jmenný prostor pass_gen, který náhodně generuje šestnáctimístné heslo založené na celé anglické abecedě, včetně velkých i malých písmen, číslic a sady speciálních znaků „`!@#$%^&*()`“. Kód tohoto generátoru je následující:

```
// Define the namespace rain_text::pass_gen
namespace rain_text::pass_gen {

// Function to generate a password of a given length
std::string GeneratePassword(uint8_t len) {
  // Create a random device
  std::random_device random;

  // Define a constant string of characters that the password can use
  static const char characters[] =
      "abcdefghijklmnopqrstuvwxyz"
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      "0123456789"
      "!@#$%^&*()";

  // Initialize an empty password string
  std::string password;

  // Add 'len' random characters to the password
  for (int i = 0; i < len; ++i) {
    // Define a uniform distribution to pick a character from 'characters' array
    std::uniform_int_distribution<int> dist(0, (sizeof(characters) - 1));

    // Pick a character from 'characters' array
    auto c = characters[dist(random)];

    // Add the chosen character to the password
    password += c;
  }

  // Return the generated password
  return password;
}
}  // namespace rain_text::pass_gen
```

Jak je z kódu patrné, funkce je již připravena na variabilitu délky hesla. Tato funkcionalita ovšem v našem programu zatím není implementována.

Po stisknutí tlačítka „Uložit“ voláme na instanci třídy `UserDatabase` metodu pro vložení dat, následně se zničí modální okno a nový záznam je zobrazen v hlavním okně.

Každý záznam lze smazat pomocí tlačítka s ikonou popelnice u každého záznamu. Toto tlačítko odstraní vizuální reprezentaci záznamu z hlavního okna a zavolá metodu třídy `UserDatabase`, která smaže záznam. Podrobnější fungování třídy `UserDatabase` je popsáno v kapitole **4. 2. 8. Třída UserDatabase**.

4. 2. 7. Třída MainDatabase

Třída MainDatabase spravuje centrální databázi aplikace. Obsahuje tři veřejné metody: konstruktor, `CreateUser` a `LoginUser`. Konstruktor slouží k vytvoření a připojení k databázi `./database/central.db` – pokud databáze existuje, pouze se k ní připojí.

Metoda pro vytvoření uživatele, `CreateUser`, vytvoří název uživatelské databáze pomocí upravené funkce MD5 z uživatelského jména, o které se stará jmenný prostor hash. Tento název je využit k vytvoření nové uživatelské databáze. Pokud však tento název již existuje, pošle zpět do třídy `Gui` prázdný řetězec, což je zachyceno a vypsána chybová hláška. Pokud jméno neexistuje, metoda jej vytvoří a pošle zpět cestu k databázi, kterou poté použijeme jako vstupní parametr pro konstruktor třídy `UserDatabase`. Dále tato metoda vytvoří záznam do centrální databáze pomocí funkce `GetPswdHash` z jmenného prostoru hash.

Metoda pro přihlášení uživatele, LoginUser, funguje obdobně jako metoda pro registraci uživatele. Rozdíl spočívá pouze v tom, že hledáme v centrální databázi vytvořený hash. Pokud jej najdeme, vrátíme cestu k databázi a pokud ne, vrátíme prázdný řetězec, což umožňuje odchycení a vypsání chybové hlášky.

Více informací o funkcích jmenného prostoru hash najdete v kapitole **4. 2. 9. Jmenný prostor hash**. Stojí také za zmínku, že pro thread-safe zápisy do databáze jak třídy `MainDatabase` tak třídy `UserDatabase` je zodpovědný jmenný prostor `database_utils`, který slouží pouze k zajištění thread-safe operací pro obě databázové třídy.

4. 2. 8. Třída UserDatabase

S uživatelskými daty se manipuluje skrze třídu `UserDatabase`. Tato třída pracuje s uživatelskou databází a zároveň využívá jádro aplikace. Během inicializace se spustí privátní metoda `EnrolmentManager` pro správu zápisů a mazání z databáze. Tato privátní metoda běží v samostatném vlákně. Toto vlákno využívá knihovny `<condition_variable>` pro čekání na signál z hlavního vlákna bez nutnosti aktivního čekání. Metoda je konstruována jako nekonečná smyčka během provozu programu, což je zajištěno statickou proměnnou `gui_stopped_`. Na počátku smyčky se čeká na změnu proměnné `gui_stopped_` nebo `add_item_` na hodnotu `true`. Po změně se proměnná `add_item_` nastaví zpět na hodnotu `false` a získají se data o záznamech pomocí metody `GetPlainData`. Tyto data se zašifrují a postupně se zapisují do stringu reprezentujícího SQL příkaz. Po vytvoření celého příkazu se obsah celé databáze smaže a nahradí novým zápisem. Poté smyčka opět přejde na svůj začátek do stavu čekání. Pokud je `gui_stopped_` nastaveno na hodnotu true, dojde k ukončení spojení s databází.
Kód této metody můžete následně prostudovat:

```
// Define the namespace rain_text
namespace rain_text {
```

```
// EnrolmentManager is a function to manage enrollment into the user database
void UserDatabase::EnrolmentManager(UserDatabase * user_db) {  // Thread

  // Obtain the plaintext data from the user database
  thread_local auto plain_data = user_db->GetPlainData();

  // Set up random distribution for separators
  thread_local std::random_device random;
  thread_local std::uniform_int_distribution<uint64_t> dist(
      0, (sizeof(separators) - 1));

  // As long as the GUI is running, there will be a thread
  while (!UserDatabase::gui_stopped_) {
    std::unique_lock<std::mutex> lock(encrypt_mutex);

    // Wait for variable change
    encrypt_cond_var.wait(lock, [&user_db] {
      return rain_text::UserDatabase::add_item_ || rain_text::UserDatabase::gui_stopped_;
    });

    // Reset add_item flag and get fresh plain data
    add_item_ = false;
    plain_data = user_db->GetPlainData();

    // Print debug output
#ifdef DEBUG
    std::cout<< "\tEnrolmentManager"<<std::endl;
    for(auto &i : plain_data) {
      std::cout << "content_id:\t" << i.content_id << std::endl;
      std::cout << "username:\t" << i.username << std::endl;
      std::cout << "password:\t" << i.password << std::endl;
      std::cout << "————————————————————————————————" << std::endl;
    }
#endif

    // Initialize maps to store and encrypt data
    std::map<std::string, std::map<size_t, std::string>> map_data;
    std::map<std::string, std::map<size_t, std::string>> map_encrypt;

    // Process plaintext data and create map_data
    for (auto &i : plain_data) {
      auto username = map_data.find(i.username);
      if (username != map_data.end()) {
        // map contains a key
        username->second[i.content_id] =
            i.headline + (char)separators[dist(random)] + i.password;
      } else {
```

```
      // map does not contain a key
      std::map<size_t, std::string> tmp_map;
      tmp_map[i.content_id] =
          i.headline + (char)separators[dist(random)] + i.password;
      map_data[i.username] = tmp_map;
    }
  }

  // Prepare the SQL statements for inserting encrypted data
  std::string sql;
  // i is a pair with first part being username and the second part being another map
  for (auto &i : map_data) {
    // Convert username to vector of uint8_t to prepare for encryption
    std::vector<uint8_t> plain_username_uint8_t(i.first.begin(),
                                                i.first.end());
    // Instantiate RainTextCore object with predefined number of iterations (16 here),
    // the user_db key, and the username
    auto rtc = std::make_unique<rain_text_core::RainTextCore>(
                    16, user_db->key_, plain_username_uint8_t);

    // Encrypt the username
    std::vector<uint8_t> encrypt_username_vector;
    rtc->Encrypt(encrypt_username_vector);

    // Convert the encrypted username to a hexadecimal string representation
    auto encrypt_username = uint8ToHexString(encrypt_username_vector);

    // Create an SQL statement to insert encrypted username into the Username table
    sql += "INSERT OR IGNORE INTO Username (username_id) VALUES ('";
    sql += encrypt_username;
    sql += "');\n";

    // Iterate over the second map (content_id mapped to the combined headline
    // and password string)
    for (auto &j : i.second) {
      // Convert content data to vector of uint8_t to prepare for encryption
      std::vector<uint8_t> plain_data_uint8_t(j.second.begin(), j.second.end());

      // Reset the text of the RainTextCore object to the content data
      rtc->SetText(plain_data_uint8_t);

      // Encrypt the content data
      std::vector<uint8_t> encrypted_data_vector;
      rtc->Encrypt(encrypted_data_vector);

      // Convert the encrypted content data to a hexadecimal string representation
      std::string encrypted_data_string = uint8ToHexString(encrypted_data_vector);
```

```
        // Create an SQL statement to insert encrypted content into the Content table
        sql += "INSERT INTO content (content_id,username_id, data) VALUES (";
        sql += std::to_string(j.first);
        sql += ",'";
        sql += encrypt_username;
        sql += "', '";
        sql += encrypted_data_string;
        sql += "');\n";
      }
    }

    // Remove all entries from tables
    std::string delete_sql = "DELETE FROM content; DELETE FROM username;";
    database_utils::ExecuteSql(user_db->connection_, delete_sql);

    // Execute the SQL statements if any
    if (!sql.empty()) {
      std::cout << sql << std::endl << std::endl << std::endl;
      database_utils::ExecuteSql(user_db->connection_, sql);
    }

    // Clear plain data for next iteration
    plain_data.clear();
  }
  // Call destructor
  delete user_db;
}
}  // namespace rain_text
```

Po inicializaci se v aplikaci volá metoda `GetData`, která vrací vektor zašifrovaných dat, pro něž je vytvořena vlastní struktura, a zároveň jsou data udržována v rámci instance třídy. Návratová hodnota je přítomná pouze pro zachování konvence, že get metoda vrací data.

Získaná data je potřeba dešifrovat a uložit do proměnné `plain_data_`. O tuto funkcionalitu se stará metoda DecryptData, která má jako vstupní parametr klíč a využívá jádro aplikace.

Jelikož se celá databáze neustále přepisuje, bylo nutné vyřešit vlastní systém pro správu ID, nazvaný ID manager. Tento manager je vektor jedniček a nul. Pokud je na určité pozici zapsána nula, záznam pro toto ID neexistuje a může se použít. Pokud je však hodnota 1, ID je obsazeno. Důležité je zmínit, že ID indexujeme od 1, i když se v jazyce C++ standardně indexuje od nuly.

Pro získání nejnižšího volného ID slouží metoda `GetId`. Tuto metodu využíváme obvykle před voláním metody `SetData`, jelikož ta jako argument přijímá strukturu pro záznam v prostém textu. `SetData` rozšíří vektor `plain_data_` a probudí vlákno s EnrolmentManagerem.

Metoda `DeleteData` přijímá ID prvku, podle něhož upraví ID manager a smaže správnou hodnotu z vektoru `plain_data_`. Toto odstranění však musí být provedeno iterační

metodou, jelikož může nastat případ, kdy data nejsou seřazena podle ID.

V destruktoru třídy se pouze probouzí `EnrolmentManager` s nastavením `gui_stopped_` na true, což ukončí smyčku společně s posledním přepisem dat a ukončí spojení s databází.

4. 2. 9. Jmenný prostor hash

Jak už název napovídá, tento rafinovaný segment se zabývá různými funkcemi, které nesou charakteristiku hashování. Naleznete zde tři dostupné funkce, konkrétně pro vytvoření názvu databáze, funkci pro vytvoření hash pro ventrální databázi a funkci pro vytvoření klíče.

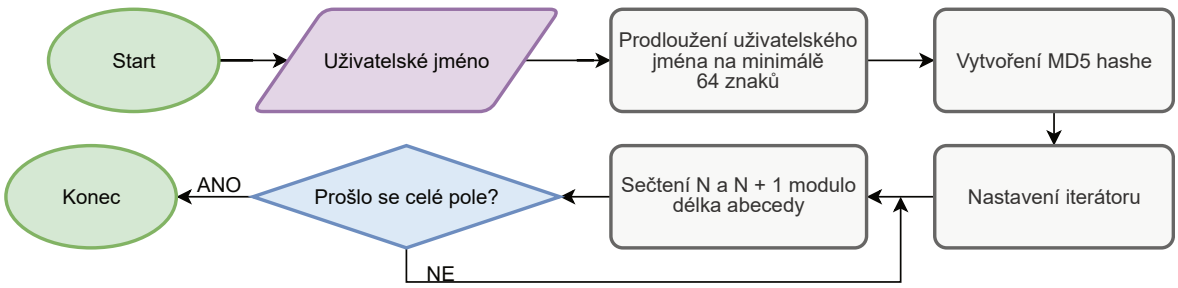
Jméno databáze je vytvořeno tak, že uživatelské jméno převedeme do jednoho řetězce tak, aby nebylo kratší než 64 znaků. Tento řetězec následně předáme funkci MD5 a její výstup upravíme tak, že proiterujeme výstup tak, abychom vždy sečetli prvek n s prvkem n+1. Na tento součet aplikujeme operaci modulo 37, abychom výsledné číslo mohli využít jako index pro řetězec abecedy doplněný o čísla. Výsledná písmena ukládáme do proměnné, která je po dokončení iterace odeslána jako výstup funkce.

Druhá funkce, která nese název `GetPswdHash`, je zasvěcena vytváření hash hodnoty z uživatelského hesla. Začíná tím, že vytvoří takzvanou „sůl“ (salt) z hesla uživatele, pomocí metody `sha3_HashBuffer`. Následně se vytváří hash hodnota hesla, která je poté převedena na hexadecimální řetězec a vrácena jako výstup funkce. Tato funkce tak zabezpečuje převod hesla na formát, který lze bezpečně uložit v databázi a který nemůže být snadno přečten.

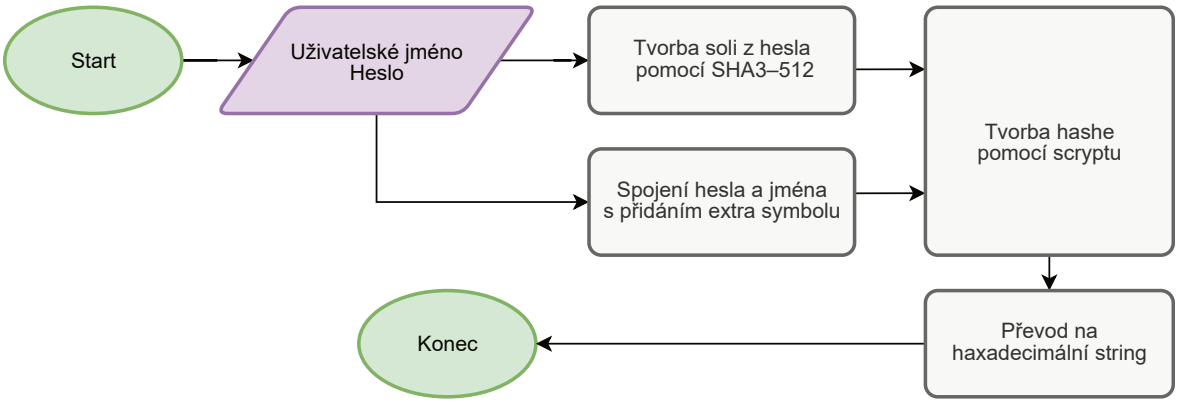
Důležitou součástí funkce je využití funkce `libscrypt_scrypt`. Tato funkce využívá metodu scrypt pro vytváření klíče z hesla. Metoda scrypt je výpočetně náročná a odolává tak pokusům o útok hrubou silou.

`GetPswdHash` využívají pomocnou funkci `uint8_array_to_hex_string` na formátování výstupu, která převádí pole typu `uint8_t na hexadecimální řetězec`. Tato funkce je zásadní pro uchování dat v čitelné a bezpečné formě.

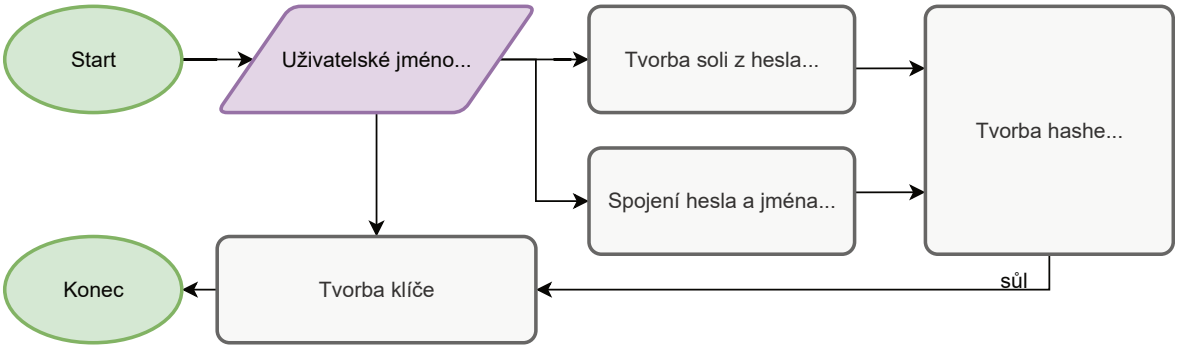
Funkce `GetKey` se stará o vytvoření klíče z uživatelského hesla a jména. Prvně vytváří „sůl“ z uživatelského jména, poté se tato „sůl“ využívá společně s uživatelským jménem a heslem pro vytvoření klíče pomocí metody scrypt. Výsledek je poté zpracován funkcí argon2id_hash_raw, která poskytuje další vrstvu bezpečnosti vytvořením odolného hash. Výsledný klíč je poté vrácen jako výstup funkce.
Nyní se na všechny funkce můžeme podívat v podobě diagramů, které jsou následovány jejich kódem.



Obrázek 17. Vývojový diagram funkce získávání jména databáze



Obrázek 18. Vyvojový diagram získávání hashe pro databázi



Obrázek 19. Vývojový diagram pro zjiskání klíče

```
// Define the namespace rain_text
namespace rain_text::hash {

// Define a set of symbols to use in the generation of the database name
char symbols[37] = "abcdefghijklmnopqrstuvwxyz0123456789";

// Function to generate a database name based on the input username
std::string GetDbName(std::string username) {
  // Initialize an array to hold the MD5 hash digest
  uint8_t digest[16];

  // Ensure the length of the username is at least 64 characters long by
  // appending copies of itself
  while (username.size() < 64) {
    username += username;
  }

  // Convert the string to a byte array for the MD5 function
  const uint8_t* initial_msg = reinterpret_cast<const uint8_t*>(username.c_str());

  // Set the length of the message to be hashed
  size_t initial_len = username.size();
```

```
  // Compute the MD5 hash of the username
  md5::md5(initial_msg, initial_len, digest);

  // Initialize an empty string to hold the result
  std::string result;

  // Use pairs of bytes from the hash digest to generate characters
  // for the database name
  for (int i = 0; i < 16; i = i + 2) {
    uint8_t tmp = (digest[i] + digest[i + 1]) % 37;
    result += symbols[tmp];
  }

  // Return the resulting database name
  return result;
}

// The GetPswdHash function takes a password and a username as arguments and
// returns the scrypt hash of the password as a hexadecimal string.
std::string GetPswdHash(std::string password, const std::string& username) {
  // Initialize a byte array to store the salt for the scrypt function
  uint8_t salt[64];

  // Use the SHA-3 hash function to generate a salt from the password
  sha3_HashBuffer(512, SHA3_FLAGS_NONE, password.data(), password.size(), salt, 64);

  // Initialize a byte array to store the output of the scrypt function
  uint8_t password_hash[64];

  // Concatenate the password and the username with a special separator to
  // form the input to the scrypt function
  std::string data = password + "\x1E" + username;

  // Generate the scrypt hash of the password using the salt
  libscrypt_scrypt(reinterpret_cast<const uint8_t*>(data.data()), data.size(),
                                    salt, 64, 65536, 8, 1, password_hash, 64);

  // Convert the scrypt hash to a hexadecimal string and return it
  return uint8_array_to_hex_string(password_hash, 64);
}

// The GetKey function generates a cryptographic key from a password and a
// username using a combination of the scrypt and Argon2id hash functions.
std::vector<uint8_t> GetKey(std::string password, std::string username) {
  // Initialize byte arrays to store intermediate and final results
  uint8_t pre_salt[64];
  uint8_t salt[64];
  uint8_t pre_result[256];
```

```
  // Use the SHA-3 hash function to generate a salt from the username
  sha3_HashBuffer(512, SHA3_FLAGS_NONE, username.data(), username.size(),
                  pre_salt, 64);

  // Concatenate the username and the password with a special separator
  // to form the input to the first scrypt function
  std::string data = username + "\x1C" + password;

  // Generate an intermediate scrypt hash using the salt
  libscrypt_scrypt(reinterpret_cast<const uint8_t*>(data.data()),
                                                    data.size(),
                                                    pre_salt,
                                                    64, 131072,
                                                    16, 1, salt,
                                                    64);

  // Generate the Argon2id hash of the password using the scrypt hash as salt
  argon2id_hash_raw(7, 1 << 18, 2, password.data(), password.size(), salt, 64,
                    pre_result, 256);

  // Convert the Argon2id hash to a vector of bytes and return it
  std::vector<uint8_t> result;
  for (auto& i : pre_result) {
    result.emplace_back(i);
  }
  return result;
}
}  // namespace rain_text::hash
```

5. TESTOVÁNÍ IMPLEMENTACE

Při tvorbě takto komplexní aplikace je testování naprosto nezbytné pro zajištění kvality a funkčnosti všech komponent. Každý element aplikace, od jádra až po samotné uživatelské rozhraní, byl pečlivě testován, aby bylo zajištěno jeho správné fungování a odolnost vůči chybám.

5. 1. Testování jádra

V rámci automatického testování jádra bylo využito jednotkových testů. Tyto testy byly vytvořeny za pomoci frameworku Google Test. K tomuto účelu byla napsána řada testů, které ověřují funkčnost a robustnost jádra. V unit testech byli prozkoušeny jednotlivé třídy a jejich ošetření vstupů a jejich správné fungování.

Dále se testovala výstupní třída, zde bylo testováno i několik variant iterací šifrování a dešifrování, aby bylo zajištěno, že počet iterací nemá vliv na funkčnost. Následující obrázek je reprezentací skutečného výpisu z unit testů.

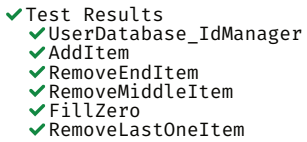
```
✔Test Results
  ✔RainTextCoreUtils_SplitKey
    ✔AccurateSize
    ✔SmalIerSize
    ✔BiggerSize
  ✔Cipher_Aes
    ✔NormalKey
    ✔Minimumkey
    ✔BadKey
    ✔BadText
    ✔BadTextSet
    ✔BadKeySet
  ✔Cipher_ChaCha20
    ✔NormalKey
    ✔MinimumKey
    ✔BadKey
    ✔BadText
    ✔BadTextSet
    ✔BadKeySet
  ✔Cipher_Twofish
    ✔NormalKey
    ✔Minimumkey
    ✔BadKey
    ✔BadText
    ✔BadTextSet
    ✔BadKeySet
  ✔Cipher_RainTextCore
    ✔Onetteration
    ✔Tenlteration
    ✔Hundretlteration
    ✔Zerolteration
```

Obrázek 20. Výpis z unit testů aplikace RainTextCore

Na druhé straně, manuální testování jádra bylo zásadní při vývoji samotného principu fungování šifrování. Zde byli využity manuální testy k zajištění očekávaného chování, z těchto manuálních testů vzešli unit testy.

5. 2. Testování samotné aplikace

Samotná aplikace nebyla vynechána z procesu automatického testování. Zde však do automatizovaných testů dostali pouze testy ID manažera ve třídě UserDatabase, jako kritická součást této třídy, kterou bylo obtížné testovat manuálně. Pro tyto účely byla třída rozšířena o některé funkcionality, které jsou dostupné pouze při testování. Tyto funkcionality obsahují getter a setter pro proměnou `id_manager_`, setter pro proměnou `key_` a `plain_data_` dále také proměnou pro uchování cesty k databázi, aby bylo možné databázi i se složkou smazat. Při testování se vyskytl malý problém s tím, že destrukce a mazání databází je asynchronní. Proto každá databáze musí mít vlastní složku pro ukládání. Zároveň je nutné v posledním testu přidat uspání aktuálního vlákna, aby nebyl přerušen proces mazání. Výsledky testů reprezentuje následující obrázek.



Obrázek 21. Výpis z unit testů aplikace RainText

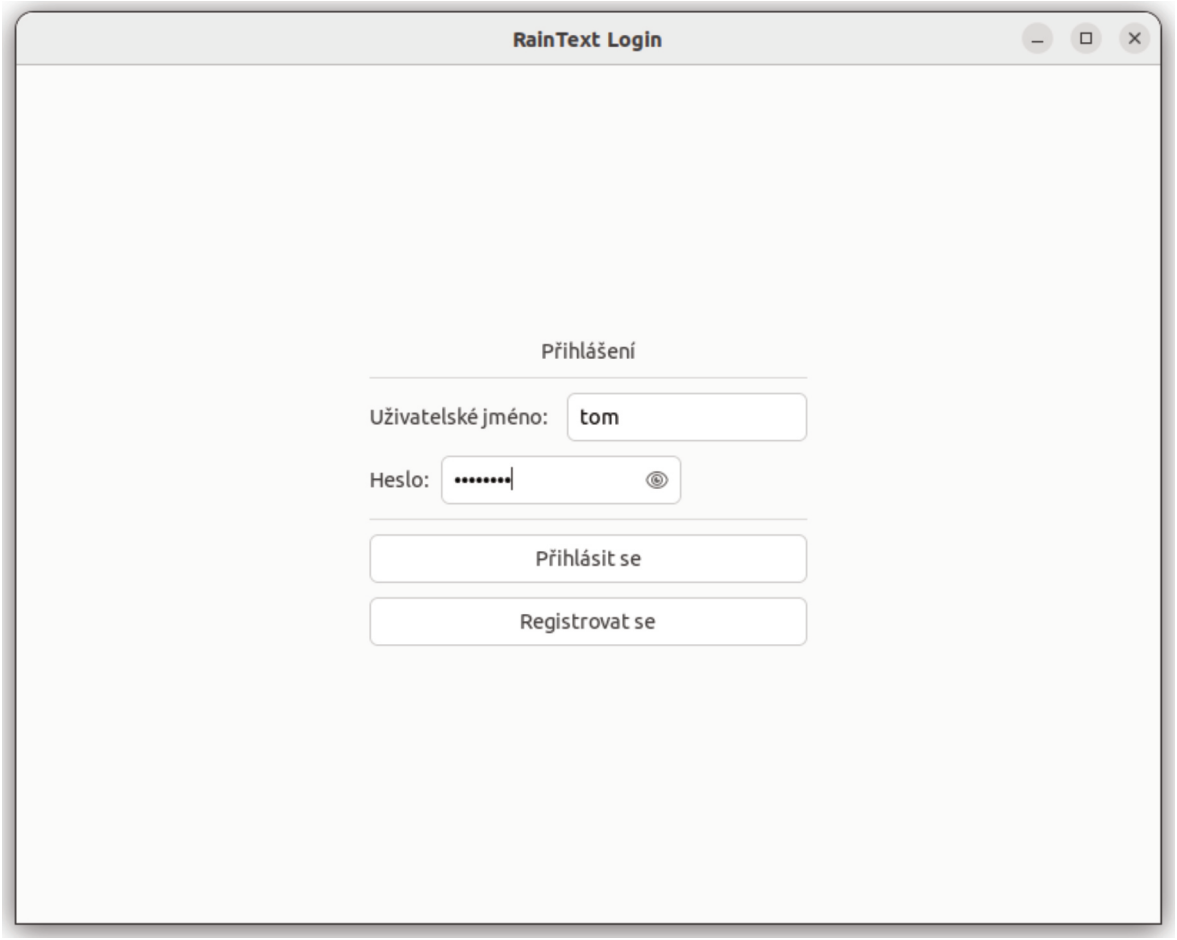
Kromě automatických testů bylo samozřejmě velmi důležité provést také manuální testování samotné aplikace a jejího grafického uživatelského rozhraní. V průběhu tohoto procesu byly testovány všechny prvky uživatelského rozhraní, a to včetně tlačítek, textových polí a dalších interaktivních prvků. Cílem bylo ověřit, že vše funguje jak má. Pro tyto účely jsou v debug módu kompilace připraveny výpisy do konzole. K tomuto řešení bylo přistoupeno z důvodu obtížnosti debugování více vláknové aplikace společně s faktem, že pracujeme s poměrně dost daty a lépe se sleduje jejich pohyb, pokud jsou srozumitelně vypsány do konzole.
Výpis takových dat v konzoli mohl vypadat následovně:

```
    GetPlainData
CreateIdManager: {1, 1, 1, 1, 1, 1}
    GetPlainData
content_id:   1
username:     111111111111
password:     7M)tPcmPW3dN&WI#
-----------------------------------
content_id:   2
username:     jmeno
password:     hUXNQ0)Y(pQT&lt6
-----------------------------------
content_id:   3
username:     jmeno2
password:     U6M%WLr)P08b2AS6
-----------------------------------
content_id:   4
username:     jmeno
password:     w0(86b0t2b2ir^KH
-----------------------------------
content_id:   5
username:     jmeno2
password:     9N157&!XF3NNlA!6
-----------------------------------
content_id:   6
username:     jmeno2
password:     mRq5$ywKQ8(5ehdt
-----------------------------------
-------------------------------
headline: banka
username: 111111111111
password: 7M)tPcmPW3dN&WI#
content_id: 1
-------------------------------

-------------------------------
headline: nadpis1
username: jmeno
password: hUXNQ0)Y(pQT&lt6
content_id: 2
-------------------------------

-------------------------------
headline: nadpis2
username: jmeno2
password: U6M%WLr)P08b2AS6
content_id: 3
```

```
------------------------------

------------------------------
headline: nadpis3
username: jmeno
password: w0(86b0t2b2ir^KH
content_id: 4
------------------------------

------------------------------
headline: nadpis4
username: jmeno2
password: 9N157&!XF3NNlA!6
content_id: 5
------------------------------
------------------------------
headline: nadpis5
username: jmeno2
password: mRq5$ywKQ8(5ehdt
content_id: 6
------------------------------
```

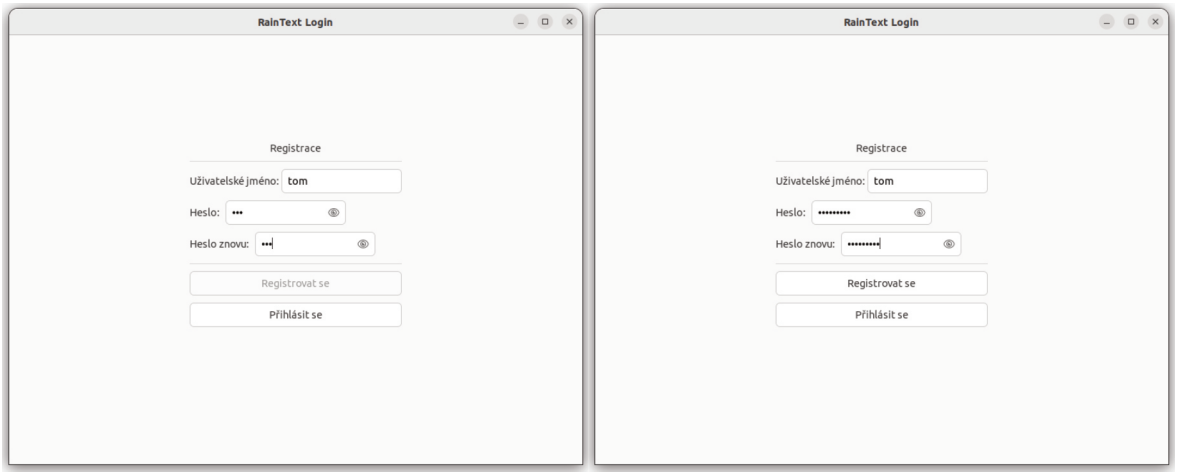
Kromě těchto výpisů bylo prováděné výše zmíněné testování grafického rozhraní, Následně budou popsány různé testy a doplněny o screenshoty.

V tomto případě testujeme, že se v okně s přihlášením zpřístupní tlačítko „Přihlásit se“ po zadání minimálně jednoho písmene pro každé textové pole.



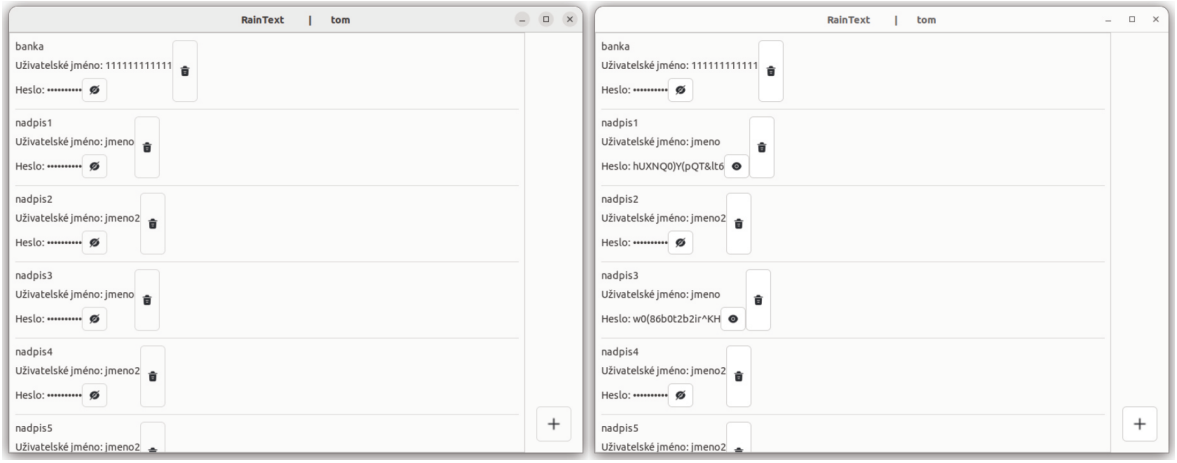
Obrázek 22. Screenshot testu zpřístupnění tlačítka „Přihlásit se“

Další test se věnuje registračnímu oknu, konkrétně zadávání hesla, v případě že se hesla shodují ale jsou příliš krátka a potom případ kdy se hesla shodují a mají dostatečnou délku.



Obrázek 23. Screenshot testu zpřístupnění tlačítka „Registrovat se“

Další a poslední obrázek se věnuje dvěma testům zároveň, test týkající se možností skrolování když je více hesel než se vejde do velikosti okna a zobrazení hesla.



Obrázek 24. Screenshot testu skrolování a zobrazení hesel

ZÁVĚR

Výsledkem této diplomové práce je funkční open-source aplikace pro správu hesel, která je přizpůsobená pro uživatele operačního systému Linux. Aplikace poskytuje silné zabezpečení a snadnou použitelnost, čímž splňuje hlavní cíle této práce.

Při vývoji byly použity moderní technologie, jako je programovací jazyk C++23, který umožnil vytvořit efektivní a bezpečnou aplikaci. Díky detailně popsánému šifrovacímu a dešifrovacímu procesu, je aplikace schopná účinně chránit uživatelská data. Navíc, díky open-source povaze projektu, mohou další vývojáři a uživatelé přispět k jeho dalšímu vývoji a zlepšení.

Během testování se aplikace ukázala jako stabilní a spolehlivá.

V budoucnu je plánováno zlepšení uživatelského prostředí, zvýšení multiplatformnosti, přidání možnosti šifrování většího objemu dat a rozšíření funkcí na poznámky a mindmapy. Takové rozšíření by mohlo zvýšit hodnotu aplikace pro uživatele a poskytnout jim více možností pro správu svých osobních údajů a informací.

Výsledky této práce ukazují, že vývoj open-source aplikací pro správu hesel je proveditelný a může přinést mnoho výhod pro uživatele, kteří se zajímají o bezpečnost svých dat.

SEZNAM POUŽITÉ LITERATURY

[1] Of History & Hashes: A Brief History of Password Storage, Transmission, & Cracking. *Trusted Sec* [online]. 2015, 2015 [cit. 2023-03-23]. Dostupné z: https://www.trustedsec.com/blog/passwordstorage/

[2] LENNON, Brian. The long history, and short future, of the password. *THE CONVERSATION* [online]. 2017, 2017 [cit. 2023-03-23]. Dostupné z: https://theconversation.com/the-long-history-and-short-future-of-the-password-76690

[3] MCMILLAN, Robert. The World's First Computer Password? It Was Useless Too. *WIRED* [online]. 2012, 2012 [cit. 2023-03-23]. Dostupné z: https://www.wired.com/2012/01/computer-password/

[4] FOGEL, Karl. *Tvorba open source softwaru: jak řídit úspěšný projekt svobodného softwaru*. Praha: CZ.NIC, [2012]. CZ.NIC. ISBN 978-80-904248-5-2.

[5] A brief history of software licensing. *LICENSEWARE* [online]. 2023, 2023 [cit. 2023-05-17]. Dostupné z: https://licenseware.io/a-brief-history-of-software-licensing/

[6] THOMAS, Amy. The First Software Licensing Agreement and its Relationship with Copyright Law. *CREATe* [online]. 2018, 2018 [cit. 2023-05-17]. Dostupné z: https://www.create.ac.uk/blog/2018/11/14/the-first-software-licensing-agreement-and-its-relationship-with-copyright-law/

[7] GNU Operating System [online]. 2023 [cit. 2023-05-22]. Dostupné z: https://www.gnu.org

[8] AUJEZDSKÝ, Josef. Licence. *Root.cz* [online]. © 1997 – 2023, © 1997 – 2023 [cit. 2023-05-17]. Dostupné z: https://www.root.cz/specialy/licence/

[9] Open Source Licenses: Types and Comparison [online]. [cit. 2023-05-22]. Dostupné z: https://snyk.io/learn/open-source-licenses/

[10] Licenses. *Choose an open source license: Choose a License* [online]. [cit. 2023-05-22]. Dostupné z: https://choosealicense.com/licenses/

[11] OSI Approved Licenses. *Open source initiative* [online]. [cit. 2023-05-22]. Dostupné z: https://opensource.org/licenses/

[12] ŘEZÁČ, Jan. *Web ostrý jako břitva: návrh fungujícího webu pro webdesignery a zadavatele projektů*. Vydání druhé. [Brno]: House of Řezáč, 2016. ISBN 978-80-270-0644-1.

[13] PEARSON, Carol S. The 12-Archetype System. *Carol S. Pearson's Website: Archetypal Narrative Intelligence (NQ)* [online]. [cit. 2023-05-18]. Dostupné z: https://www.carolspearson.com/about/the-pearson-12-archetype-system-human-development-and-evolution

[14] SCHWARTZMAN, Jamie. The Powerful Nature of Brand Archetypes. *Flux Branding: Los Angeles Brand Identity Agency* [online]. 2020, 2020 [cit. 2023-05-18]. Dostupné z: https://fluxbranding.com/intro-to-brand-archetypes/

[15] Co jsou brand archetypy a proč jsou opravdu potřebné. *Ecommerce Bridge* [online]. 2020, 2020 [cit. 2023-05-18]. Dostupné z: https://www.ecommercebridge.cz/co-jsou-brand-archetypy-a-proc-jsou-opravdu-potrebne/

[16] GNU Software. *GNU Operating System* [online]. 2021, 2021 [cit. 2023-05-19]. Dostupné z: https://www.gnu.org/software/software.html

[17] Number of Internet Users in 2022/2023: Statistics, Current Trends, and Predictions. *FinancesOnline.com* [online]. 2023, 2023 [cit. 2023-05-19]. Dostupné z: https://financesonline.com/number-of-internet-users/#:~:text=Of%20which%2C%2092.6%25%20access%20the%20world%20wide%20web,as%20of%20January%202021.%20Source%3A%20Internet%20World%20Stats

[18] SIDDI, Francesco. Blender by the Numbers: 2020. *Blender* [online]. 2021, 2021 [cit. 2023-05-23]. Dostupné z: https://www.blender.org/news/blender-by-the-numbers-2020/

[19] SINGH, Simon, Petr KOUBSKÝ a Dita ECKHARDTOVÁ. *Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii*. 2. vyd. v českém jazyce. Praha: Dokořán, 2009. Aliter (Argo: Dokořán): Dokořán. ISBN 978-80-7363-268-7.

[20] KAŠPÁRKOVÁ, Lenka. *Vývoj písma a typografie; vývoj psaného písma* [online]. In: . 2011 [cit. 2023-03-26]. Dostupné z: https://www.sspu-opava.cz/static/UserFiles/File/_sablony/Technologie_grafiky_III/VY_32_INOVACE_B-04-11.pdf

[21] STANGER, James. The Ancient Practice of Steganography: What is it, How is it Used and Why Do Cybersecurity Pros Need to Understand it?. *CompTIA* [online]. 2020, 2020 [cit. 2023-04-07]. Dostupné z: https://www.comptia.org/blog/what-is-steganography

[22] Praktické základy Kryptologie a Steganografie. *Security-Portal.cz* [online]. 2004, 2004 [cit. 2023-04-07]. Dostupné z: https://www.security-portal.cz/clanky/praktick%C3%A9-z%C3%A1klady-kryptologie-steganografie

[23] MCBRIEN, Scott. Linux file permissions explained: Understanding Linux file permissions (how to find them, read them, and change them) is an important part of maintaining and securing your systems. *Red Hat* [online]. 2023, 2023 [cit. 2023-04-07]. Dostupné z: https://www.redhat.com/sysadmin/linux-file-permissions-explained

[24] GARN, Damon. How to manage users and groups in Linux: User and group management are two must-have skills for every Linux administrator. *Red Hat* [online]. 2021, 2021 [cit. 2023-04-08]. Dostupné z: https://www.redhat.com/sysadmin/linux-user-group-management

[25] KYSELA, Martin. Sedláme Linux, 5. díl: uživatelé a skupiny. *Živě* [online]. 2003, 2003 [cit. 2023-04-08]. Dostupné z: https://www.zive.cz/clanky/sedlame-linux-5-dil-uzivatele-a-skupiny/sc-3-a-112712/default.aspx

[26] KODY. Bypass Locked Windows Computers to Run Kali Linux from a Live USB. *Wonder How To* [online]. 2018, 2018 [cit. 2023-04-07]. Dostupné z: https://null-byte.wonderhowto.com/how-to/bypass-locked-windows-computers-run-kali-linux-from-live-usb-0185093/

[27] History of cryptology. *Britannica* [online]. ©2023, ©2023 [cit. 2023-04-08]. Dostupné z: https://www.britannica.com/topic/cryptology/History-of-cryptology

[28] BURDA, Karel. *Úvod do kryptografie*. Brno: Akademické nakladatelství CERM, 2015. ISBN 978-80-7204-925-7.

[29] Caesarova šifra. *Matematika polopatě* [online]. 2006—2022, 2006—2022 [cit. 2023-04-13]. Dostupné z: https://www.matweb.cz/caesarova-sifra/

[30] [30] OLŠÁK, Petr. Afinní transformace. In: *Petr Olsak* [online]. 2010, 2010 [cit. 2023-04-13]. Dostupné z: https://petr.olsak.net/bilin/afinita.pdf

[31] O'NEAL, Bryan, Charles H. DYER, Daniel GREEN, et al., , ed. *The Moody Bible Commentary*. Moody Publishers, 2014. ISBN 9780802490186.

[32] Vigenèrova šifra. *Matematika polopatě* [online]. 2006—2022, 2006—2022 [cit. 2023-04-16]. Dostupné z: https://www.matweb.cz/vigenerova-sifra/

[33] Jak spočítat délku klíče Vigenèrovy šifry. *Matematika polopatě* [online]. 2006—2022, 2006—2022 [cit. 2023-04-16]. Dostupné z: https://www.matweb.cz/kasiskeho-test/

[34] Symetrické techniky šifrování. *Khan Academy* [online]. © 2023, © 2023 [cit. 2023-04-16]. Dostupné z: https://cs.khanacademy.org/computing/informatika-pocitace-a-internet/x8887af37e7f1189a:internet/x8887af37e7f1189a:sifrovani-dat/a/symmetric-encryption-techniques

[35] 5 An Unbreakable Cipher. *Stony Brook University* [online]. 2005, 2005 [cit. 2023-04-16]. Dostupné z: https://www.math.stonybrook.edu/~scott/papers/MSTP/crypto/5Unbreakable_Cipher.html

[36] FROEHLICH, Andrew. DEFINITION: one-time pad. *TechTarget* [online]. Copyright 2000 - 2023, Copyright 2000 - 2023 [cit. 2023-04-23]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/one-time-pad

[37] Šifra Playfair. *Shaman.cz* [online]. 2021, 2021 [cit. 2023-04-16]. Dostupné z: http://www.shaman.cz/sifrovani/sifra-playfair.htm

[38] Playfair Cipher. *Crypto-IT* [online]. 2020, 2020 [cit. 2023-04-16]. Dostupné z: https://www.crypto-it.net/eng/simple/playfair-cipher.html

[39] [4.0] Codes & Codebreakers In World War I: [4.3] GEORGES PAINVIN & THE ADFGX / ADFGVX CIPHERS. *Wayback Machyne* [online]. 2010, 2010 [cit. 2023-04-17]. Dostupné z: https://web.archive.org/web/20100503103848/http://www.vectorsite.net/ttcode_04.html#m3

[40], Neso Academy. Hill Cipher (Encryption). In: *YouTube* [online]. 2021, 2021 [cit. 2023-04-17]. Dostupné z: https://www.youtube.com/watch?v=-EQ8UomTrAQ

[41] BADAYALYA, Aditya. Cryptanalysis of Hill Cipher. *CodeStudio* [online]. 2023, 2023 [cit. 2023-04-18]. Dostupné z: https://www.codingninjas.com/codestudio/library/cryptanalysis-of-hill-cipher

[42] HILL CIPHER: CRYPTANALYLSIS [online]. [cit. 2023-04-18]. Dostupné z: https://hillcipher.weebly.com/cryptanalysis.html

[43] Četnost znaků v českém textu. *Algoritmy.net* [online]. © 2016 [cit. 2023-04-21]. Dostupné z: https://algoritmy.net/article/40/Cetnost-znaku-CJ

[44] OWEN, Jared. How did the Enigma Machine work?. In: *YouTube* [online]. 2021, 2021 [cit. 2023-04-26]. Dostupné z: https://www.youtube.com/watch?v=ybkkiGtJmkM

[45] M, Julia. Enigma machine: 3D Mode. *Sketchfab* [online]. 2018, 2018 [cit. 2023-04-26]. Dostupné z: https://sketchfab.com/3d-models/enigma-machine-f8cf72314e334b0d9760a90f40b3edb3

[46] ERBEN, Lukáš. Úsvit hackerů: Marian Rejewski a Enigma. *Root* [online]. 2013, 2013 [cit. 2023-04-27]. Dostupné z: https://www.root.cz/clanky/usvit-hackeru-marian-rejewski-a-enigma/

[47] ERBEN, Lukáš. Úsvit Hackerů: Enigma a Ultra. *Root* [online]. 2013, 2013 [cit. 2023-04-27]. Dostupné z: https://www.root.cz/clanky/usvit-hackeru-enigma-a-ultra/

[48] ERBEN, Lukáš. Úsvit hackerů: Bombe Alana Turinga. *Root* [online]. 2013, 2013 [cit. 2023-04-27]. Dostupné z: https://www.root.cz/clanky/usvit-hackeru-bombe-alana-turinga/

[49] What Is DES (Data Encryption Standard)? DES Algorithm and Operation. *Simplilearn: Online Courses - Bootcamp & Certification Platform* [online]. 2022, 2022 [cit. 2023-05-23]. Dostupné z: https://www.simplilearn.com/what-is-des-article

[50] TOMANOVÁ, Amaya. Zpátky do minulosti: Prolomení DES a registrace videoher od Atari (17. 6. 2021). *Jablíčkář.cz* [online]. 2021, 2021 [cit. 2023-05-23]. Dostupné z: https://jablickar.cz/zpatky-do-minulosti-prolomeni-des-a-registrace-videoher-od-atari-17-6-2021/

[51] LAKE, Josh. What is 3DES encryption and how does DES work?. *Comparitech* [online]. 2022, 2022 [cit. 2023-05-23]. Dostupné z: https://www.comparitech.com/blog/information-security/3des-encryption/#Is_3DES_safe

[52] RIMKIENĖ, Rūta. What is AES encryption and how does it work?. In: *Cybernews* [online]. 2022, 2022 [cit. 2023-05-23]. Dostupné z: https://cybernews.com/resources/what-is-aes-encryption/

[53] AWATI, Rahul. Blowfish. *TechTarget* [online]. Copyright 2000 - 2023, Copyright 2000 - 2023 [cit. 2023-05-23]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/Blowfish#:~:text=Blowfish%20features%20a%2064-bit%20block%20size%20and%20takes,The%20Blowfish%20algorithm%20consists%20of%20two%20major%20parts%3A

[54] ZAHORSKI, Alexiei. Everything You Need to Know About the Twofish Encryption Algorithm. *MUO: Technology, Simplified* [online]. 2022, 2022 [cit. 2023-05-23]. Dostupné z: https://www.makeuseof.com/twofish-encryption-algorithm-explained/

[55] YAMBADWAR, Snigdha. What is RC4 Encryption?. *GeeksforGeeks* [online]. 2021 [cit. 2023-05-23]. Dostupné z: https://www.geeksforgeeks.org/what-is-rc4-encryption/

[56] NAGARAJ, Karthikeyan. Understanding ChaCha20 Encryption: A Secure and Fast Algorithm for Data Protection | 2023. *Medium* [online]. 2023, 2023 [cit. 2023-05-23]. Dostupné z: https://cyberw1ng.medium.com/understanding-chacha20-encryption-a-secure-and-fast-algorithm-for-data-protection-2023-a80c208c1401

[57] BERNSTEIN, Daniel J. The Salsa20 family of stream ciphers. In: *Cr.yp.to* [online]. [cit. 2023-05-23]. Dostupné z: https://cr.yp.to/snuffle/salsafamily-20071225.pdf#:~:text=Salsa20%20expands%20a%20256-bit%20key%20and%20a%2064-bit,stream%20and%20discarding%20the%20rest%20of%20the%20stream.

[58] LAKE, Josh. What is RSA encryption and how does it work?. *Comparitech* [online]. 2021, 2021 [cit. 2023-05-23]. Dostupné z: https://www.comparitech.com/blog/information-security/rsa-encryption/

[59] ARAMPATZIS, Anastasios. How Does Elliptic Curve Cryptography Work?: Explore the world of elliptic curve cryptography. *DZone* [online]. 2019, 2019 [cit. 2023-05-23]. Dostupné z: https://dzone.com/articles/how-does-elliptic-curve-cryptography-work

[60] The ElGamal cryptosystem. *Cryptography Academy* [online]. © 2023, © 2023 [cit. 2023-05-23]. Dostupné z: https://cryptographyacademy.com/elgamal/

[61] TYSON, Matthew. Understand Diffie-Hellman key exchange. *InfoWorld* [online]. 2022,

2022 [cit. 2023-05-23]. Dostupné z: https://www.infoworld.com/article/3647751/understand-diffie-hellman-key-exchange.html

[62] ElGamal Encryption Algorithm. *GeeksforGeeks* [online]. [cit. 2023-05-23]. Dostupné z: https://www.geeksforgeeks.org/elgamal-encryption-algorithm/

[63] MASOOD, Fawad, Jawad AHMAD, Syed Aziz SHAH, Sajjad Shaukat JAMAL a Iqtadar HUSSAIN. A Novel Hybrid Secure Image Encryption Based on Julia Set of Fractals and 3D Lorenz Chaotic Map. *Entropy* [online]. 2020, **22**(3), 274. ISSN 1099-4300. Dostupné z: doi:10.3390/e22030274

[64] Cross-Site Request Forgery (CSRF) Attack: What It Is, How It Works, and How to Prevent It. *Akshay's Blog* [online]. 2023, 2023 [cit. 2023-04-06]. Dostupné z: https://www.akshaykhot.com/how-csrf-attack-works-cross-site-request-forgery/

[65], KirstenS. Cross Site Request Forgery (CSRF). *OWASP Foundation* [online]. Copyright 2023, Copyright 2023 [cit. 2023-04-06]. Dostupné z: https://owasp.org/www-community/attacks/csrf

[66], KirstenS. Cross Site Scripting (XSS). *OWASP Foundation* [online]. Copyright 2023, Copyright 2023 [cit. 2023-04-06]. Dostupné z: https://owasp.org/www-community/attacks/xss/

[67] The State of Developer Ecosystem 2017-2022. *JetBrains* [online]. © 2000-2023, © 2000-2023 [cit. 2023-03-31]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2022/

[68] Screenshots. *WxWidgets* [online]. © 2023, © 2023 [cit. 2023-05-16]. Dostupné z: https://www.wxwidgets.org/about/screenshots/

[69] Filesystem library: (since C++17). *Cppreference.com* [online]. 2022, 2022 [cit. 2023-05-15]. Dostupné z: https://en.cppreference.com/w/cpp/filesystem

[70] Threads. *Cppreference.com* [online]. 2022, 2022 [cit. 2023-05-15]. Dostupné z: https://en.cppreference.com/w/cpp/thread

[71] Modules: (since C++20). *Cppreference.com* [online]. 2023, 2023 [cit. 2023-05-15]. Dostupné z: https://en.cppreference.com/w/cpp/language/modules

[72] C++ compiler support. *Cppreference.com* [online]. 2022, 2022 [cit. 2023-05-15]. Dostupné z: https://en.cppreference.com/w/cpp/compiler_support

[73] Google Trends. *Google Trends* [online]. 2023, 2023 [cit. 2023-05-22]. Dostupné z: https://trends.google.com/trends/explore?date=2018-01-01%202023-05-15&gprop=youtube&q=%2Fm%2F0cxh7f,%2Fg%2F11cmy51gz6,%2Fm%2F0173j0,Bazel,%2Fg%2F11f03_yf9j

[74] SQLite [online]. 2023 [cit. 2023-05-21]. Dostupné z: https://www.sqlite.org/index.html

[75] About SQLite. *SQLite* [online]. [cit. 2023-05-16]. Dostupné z: https://sqlite.org/about.html

[76] P-H-C / phc-winner-argon2. *GitHub* [online]. 2021, 2021 [cit. 2023-05-20]. Dostupné z: https://github.com/P-H-C/phc-winner-argon2

[77] Technion / libscrypt. *GitHub* [online]. 2021, 2021 [cit. 2023-05-20]. Dostupné z: https://github.com/technion/libscrypt

[78] google/googletest. *GitHub* [online]. 2023, 2023 [cit. 2023-05-21]. Dostupné z: GitHub - google/googletest: GoogleTest - Google Testing and Mocking Framework

[79] *Crypto++® Library 8.7: Free C++ Class Library of Cryptographic Schemes* [online]. 2021 [cit. 2023-05-21]. Dostupné z: https://www.cryptopp.com

[80] *Google C++ Style Guide* [online]. [cit. 2023-04-06]. Dostupné z: https://google.github.io/styleguide/cppguide.html

[81] *Iconsax* [online]. © 2021 [cit. 2023-05-21]. Dostupné z: https://iconsax.io

[82] Pod32g / MD5. *GitHub* [online]. 2022, 2022 [cit. 2023-05-21]. Dostupné z: https://github.com/pod32g/MD5

[83] Brainhub / SHA3IUF. *GitHub* [online]. 2022, 2022 [cit. 2023-05-21]. Dostupné z: GitHub - brainhub/SHA3IUF: C implementation of the SHA-3 and Keccak with Init/Update/Finalize hashing API (NIST FIPS 202/Etherium)

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

- **API** – Application Programming Interface (Aplikační programovací rozhraní).
- **GIMP** – GNU Image Manipulation Program (Program na manipulaci s obrázky GNU).
- **GNU** – GNU's not Unix (GNU není unix).
- **GPL** – GNU General Public License (Všeobecná veřejná licence GNU).
- **GTK** – Gimp Toolkit (Sada nástrojů GIMP).
- **GUI** – Graphical user interface (Grafické uživatelské rozhraní).
- **IDE** – Integrated Development Environment (Integrované vývojové prostředí).

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

- Adresář Text diplomové práce – obsahuje text diplomové práce ve formátu PDF.
- Adresář Zdrojové kódy – obsahuje projekty s jádrem a aplikací