VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÉ UŽIVATELSKÉ ROZHRANÍ SPRÁVCE HESEL PASS

WEB USER INTERFACE FOR PASS PASSWORD MANAGER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL CHROUST**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19694/2016/xchrou02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Chroust Michal**
Obor: Informační technologie
Téma: **Webové uživatelské rozhraní správce hesel Pass**
Web User Interface for Pass Password Manager
Kategorie: Web

Pokyny:

1. Seznamte se se správcem hesel Pass. Prostudujte způsob šifrování a uložení hesel v úložišti verzovaném pomocí Git.
2. Navrhněte webovou aplikaci pro správu hesel ve verzovaném úložišti Pass nad Git přímo i prostřednictvím aplikace správce hesel Pass. Umožněte přepínání aplikace mezi režimy "běhu ve webovém prohlížeči" (implementace aplikace, šifrování/PGP, verzování/Git v jazyce JavaScript v prohlížeči) a "použití externích nástrojů" (volání Pass, GPG a Git mimo prohlížeč). Umožněte nasazení na serveru i čistě v prohlížeči. Zaměřte se také na bezpečnost správy hesel.
3. Po konzultaci s vedoucím navrženou aplikaci implementujte.
4. Zdokumentujte výsledky a zveřejněte projekt pod svobodnou licencí jako open-source.

Literatura:

- Donenfeld, J.A.: *Pass - the standard unix password manager.* [https://www.passwordstore.org/]
- GitHub - creationix/js-git: A JavaScript implementation of Git. [https://github.com/creationix/js-git]
- GitHub - openpgpjs/openpgpjs: OpenPGP implementation for JavaScript [https://github.com/openpgpjs/openpgpjs]
- Ondřej Žára. *JavaScript: programátorské techniky a webové technologie*. Computer Press, 2015. ISBN 978-80-251-4573-9

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rychlý Marek, RNDr., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom práce je implementácia správcu hesiel s podporou verzovania cez službu git a šifrovaním pomocou knižnice OpenPGP.js. Zaoberá sa vhodnou štruktúrou ich uloženia v pamäti, použitím knižnice Promise, konverziou na Base64 a späť, bezpečnosťou hesiel, HTTPS požiadavkami a prácou so službou git. Zameriava sa tiež na nástroje a technológie použité počas vývoja aplikácie. Aplikácia umožňuje jej použitie na serveri aj v prehliadači.

Abstract

The goal of the thesis is implementation of password manager with support of version control trough git and encryption using OpenPGP.js library. Work addresses suitable structure of storing passwords in memory, Promise library usage, conversion to and from Base64, passwords security, HTTPS requests and git service usage. It also focuses on tools and technologies used troughout application development. Application allows it's use both on server and browser.

Kľúčové slová

správca hesiel, asynchrónne funkcie, šifrovanie verejným kľúčom, JavaScript, Node.js, Browserify, git

Keywords

password manager, asynchronous functions, public key cryptography, JavaScript, Node.js, Browserify, git

Citácia

CHROUST, Michal. *Webové uživatelské rozhraní správce hesel Pass*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Webové uživatelské rozhraní správce hesel Pass

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána RNDr. Mareka Rychlého, Ph.D. a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Chroust
17. mája 2017

Obsah

Kapitola 1

Úvod

V súčasnej dobe by bolo ťažké nájsť človeka, ktorý by za svoj život aspoň raz nepoužíval nejaké heslo. Či už je to PIN kód ku kreditnej karte, účet na sociálnej sieti alebo prístupové údaje ku informačnému systému. Ak je týchto hesiel len pár, je možné si ich zapamätať, problém však nastáva pri používaní veľkého množstva rôznych on-line služieb, ktoré vyžadujú účet. Mnoho ľudí používa jedno heslo pre prístup ku viacerým účtom. To však predstavuje bezpečnostné riziko pri jeho kompromitácii a môže mať ďalekosiahle následky. Štúdia spoločnosti Ofcom z roku 2013 zistila, že viac ako 55% ľudí používa rovnaké heslo na skoro všetkých, ak nie na všetkých webových stránkach. 25% opýtaných ľudí dokonca používa ako heslo svoj dátum narodenia alebo meno. [7]

V roku 2012 spoločnosť LinkedIn oznámila, že bolo kompromitovaných viac ako 7 miliónov prihlasovacích údajov kyberhakermi z Ruska. Neskôr bolo navyše zistené, že ďalších 100 miliónov e-mailových adries a zahašovaných hesiel bolo k dispozícii online. Útočníci pokročili vo svojich technikách a používajú automatizované nástroje, vďaka ktorým vedia tieto údaje porovnávať s mnohými inými stránkami a zistiť validitu týchto údajov. Pri používaní jedného hesla tak môže používateľ prísť o účty na mnohých stránkach.

Takýmto praktikám je možné predísť používaním rôznych silných hesiel na každej stránke. Nutnosť zapamätať si desiatky údajov môže uľahčiť správca hesiel. V súčasnej dobe je dostupné neprieberné množstvo takýchto služieb. Existujú aplikácie do smartfónov, pre operačné systémy na desktopových počítačoch, rozšírenia do prehliadačov a webové stránky, ktoré poskytujú takúto funkcionalitu. Takýto prístup však podľa štatistík veľa ľudí nevolí a riadia sa ním skôr experti v odbore online bezpečnosti. Podľa prieskumu organizácie RoboForm iba 8% ľudí používa správcu hesiel.[9]

Cieľom tejto práce je vytvoriť knižnicu v JavaScripte, ktorá umožňuje ukladanie a správu hesiel. Tie budú zašifrované pomocou kryptografie verejného kľúča, aby bola zachovaná ich bezpečnosť. O históriu úložiska a jeho prípadné zdieľanie medzi viacerými ľudmi sa bude starať verzovanie pomocou služby git. Bude ju možné použiť v prehliadačoch aj v serverových aplikáciách.

Práca je rozčlenená do desiatich kapitol. Kapitola 2 stručne opisuje použité technológie, ich históriu a využitie. Kapitola 3 sa zaoberá správcom hesiel Pass, ktorým je táto knižnica inšpirovaná a štruktúrou úložiska kompatibilná. Kapitola 4 podáva informácie o nástrojoch, ktoré boli použité pri vývoji a ich účele. Kapitola 5 rozoberá momentálne dostupné riešenia a ich funkcionalitu. V kapitole 6 sú rozobraté problémy, s ktorými som sa stretol pri návrhu aplikácie. V kapitole 7 sú nastienené zaujímavé úseky implementácie. V kapitole s poradovým číslom 8 sú popísané možné bezpečnostné riziká pri používaní knižnice. Predposledná kapitola sa zameriava na kompatibilitu s prehliadačmi. V poslednej kapitole je zhŕňa dosiahnuté výsledky a popisuje ďalší možný vývoj.

Kapitola 2

Použité technológie

2.1 JavaScript

JavaScript prvýkrát uzrel svetlo sveta v roku 1995 a umožnil pridávanie programov na webové stránky v priehliadači Netscape Navigator. Odvtedy jazyk prevzali všetky ostatné najpoužívanejšie grafické webové prehliadače. Vďaka nemu existujú moderné webové aplikácie – aplikácie, s ktorými je možné interagovať priamo, bez obnovy stránky pre každú akciu. Je tiež použitý vo viac tadičných webových stránkach, kde poskytuje rôzne formy interaktivity.

Považujem za dôležité zdôrazniť, že JavaScript nemá spoločné nič s programovacím jazykom nazvaným Java. Podobné meno bolo inšpirované trhovými záujmami. Keď bol JavaScript prvýkrát predstavený sa jazyk Java stával veľmi populárnym. Niekoho napadlo, že by bolo dobré zviesť sa na tejto vlne úspechu. Odvtedy jazyku prischlo jeho meno.

Po jeho adaptácii mimo preliadača Netscape bol spísaný dokument, ktorý popisoval spôsob, akým by mal jazyk JavaScript fungovať. Bolo to potrebné, aby rôzny software, ktorý sa hlásil k podpore JavaScriptu, skutočne podporoval ten istý jazyk. Tento štandard nesie názov ECMAScript, po organizácii ECMA International, ktorá štandard vytvorila. V praxi môžu byť pojmy ECMAScript a JavaScript zamenené – hovoria však o tom istom jazyku.[6, str. 6-7]

2.1.1 Promise

Najväčšou výzvou pri väčšom množstve kódu v JavaScripte, ktorý využíva asynchrónne volania, je serializácia jeho vykonávania do série krokov a spracovanie chybových stavov, ktoré môžu nastať. Knižnica Promise rieši tento problém tým, že nám umožňuje zorganizovať callback-y do série diskrétnych krokov, ktoré sú čitateľnejšie a ľahšie na správu. Keď nastane chybový stav, je možné ho spracovať mimo hlavnej logiky aplikácie, bez potreby overovať stav po každom kroku.

Promise je objekt, ktorý slúži ako náhrada za hodnotu. Táto hodnota je väčšinou výsledkom asynchrónnej operácie, ako napríklad HTTP požiadavka alebo čítanie z disku. Po spustení asynchrónnej funkcie môže byť hneď vrátený objekt promise. Použitím tohto objektu je možné nastaviť funkcie, ktoré majú byť spustené po úspešnom dokončení alebo chybe.[8, str. 11]

Knižnica promise jazyk JavaScript podporuje od verzie ES6.

2.2 OpenPGP

OpenPGP je otvorený protokol pre šifrovanie e-mailovej komunikácie pomocou kryptografie s verejným kľúčom. Tento protokol definuje štandardné formáty pre zašifrované správy, podpisy a certifikáty pre výmenu verejných kľúčov.

Je založený na pôvodnom voľne šíriteľnom softvéry PGP (Pretty Good Privacy), ktorý vytvoril v roku 1991 Phil Zimmermann. Kvôli licencii PGP bol tri roky vyšetrovaný, pretože vláda USA tvrdila, že boli porušené obmedzenia na export kryptografického softvéru po tom, čo sa program rozšíril po celom svete.

Napriek neexistencii finančnej podpory, plateného personálu, či spoločnosťou, ktorá by stála za týmto programom a napriek prenasledovaniu vládou sa PGP stalo najpoužívanejším softvérom na šifrovanie e-mailovej komunikácie na svete. Po tom, čo vláda ukončila vyšetrovanie v roku 1996 Zimmermann vytvoril spoločnosť GPG Inc. Táto spoločnosť a jej intelektuálne vlastníctvo odkúpila v decembri 1997 spoločnosť Network Associates Inc (NAI). NAI pokračovala vo vývoji do roku 2002, kedy predala softvér firme PGP Corporation.

OpenPGP je otvorenou verziou šifrovacieho protokolu NAI PGP. Pracovná skupina OpenPGP sa momentálne snaží o kvalifikáciu OpenPGP ako Internetového štandardu IETF.

OpenPGP je momentálne na ceste ku Internetovému štandardu a je v aktívnom vývoji. Mnoho e-mailových klientov poskytuje bezpečnosť vyhovujúcu štandardu, ktorý je popísaný v RFC 3156. Momentálna špecifikácia je RFC 4880 (November 2007). V tomto dokumente je špecifikovaná sada požadovaných algoritmov, skladajúcich sa zo šifrovania ElGamal, DSA, Triple DES a SHA-1. Ako doplnok k týmto algoritmom štandard odporúča RSA pre šifrovanie a podpisovanie, a tiež AES-128, CAST-128 a IDEA. Okrem týchto je podporovaných mnoho ďalších algoritmov. Štandard bol rozšírený v roku 2009, aby podporoval šifru Camellia (RFC 5581) a v roku 2012 bola pridaná podpora kryptografie na báze eliptických kriviek (RFC 6637). Očakáva sa aj pridanie podpory pre EdDSA, ktorá bola navrhnutá v roku 2014.

Podľa dostupných informácií momentálne nie je možné prelomiť tento štandard.

Uvedený štandard môže byť implementovaný akoukoľvek spoločnosťou bez nutnosti platby za licenciu.[5]

2.3 OpenPGP.js

Snahou tohto projektu je poskytnúť otvorenú OpenPGP knihovnu v JavaScripte, vďaka čomu môže byť použitá takmer na každom zariadení. Namiesto iných implementácií, ktoré sa zameriavajú na použitie natívneho kódu, OpenPGP.js zámerne obchádza túto požiadavku (takže ľudia nemusia inštalovať gpg na ich zariadeniach, aby mohli využívať knižnicu). Konceptom je implementácia všetkej potrebnej funkcionality OpenPGP v JavaScriptovej knižnici, ktorá môže byť neskôr použitá v iných projektoch, či už ako rozšírenia prehliadačov alebo serverové aplikácie. Knižnica umožnuje podpisovanie, zašifrovanie, dešifrovanie a overenie akéhokoľvek textu — hlavne e-mailov — a tiež správu kľúčov.[2]

Tento projekt bol spojený s knižnicou GPG4Browsers, ktorá bola vyvýjaná spoločnosťou Recurity Labs.

2.4 GNU Privacy Guard

GNU Privacy Guard (GnuPG alebo GPG) je voľný softvér, ktorý je náhradou pre sadu kryptografického softvéru PGP od spoločnosti Symantec. GnuPG vyhovuje RFC 4880, ktorý je štandardnou špecifikáciou IETF pre OpenPGP. Moderné verzie PGP a programu FileCrypt od spoločnosti Veridis sú schopné spolupracovať s GnuPG a ostatnými systémami vyhovujúcimi Štandardu OpenPGP.

Program pôvodne vytvoril Werner Koch. Prvá produkčná verzia bola uvoľnená dva roky po začiatku vývoja.

GnuPG je súčasťou GNU Project a dostal veľkú dotáciu od Nemeckej vlády na dokumentáciu a naprogramovanie verzie pre Microsoft Windows.

GnuPG je hybridný šifrovací softvér, pretože používa kombináciu konvenčného symetrického šifrovania pre rýchlosť, a kryptografiu verejným kľúčom kvôli jednoduchej výmene kľúčov, typicky použitím verejného kľúča príjemcu na zašifrovanie kľúča relácie, ktorý je použitý iba raz. Tento spôsob fungovania je súčasťou štandardu OpenPGP aj programu PGP od jeho prvej verzie.

Staršie verzie GnuPG používali vstavanú kryptografickú knižnicu, no od verzie 2.x bola nahradená knižnicou Libgcrypt.

GPG šifruje správy použitím párov asymetrických kľúčov individuálne vygenerovaných používateľom. Vzniknuté kľúče môžu byť vymenené s ostatnými používateľmi rôznymi spôsobmi, napríklad pomocou verejného servera kľúčov. Musia byť vymieňané s opatrnosťou, aby nedošlo ku sfalšovaniu identity. Taktiež je možné ku správe pridať kryptografický digitálny podpis, vďaka čomu môže byť overená integrita správy a jej odosielateľa, za predpokladu, že predchádzajúca korešpondencia nebola odhalená.

Tento program tiež podporuje symetrické šifrovacie algoritmy. Implicitne používa symetrický algoritmus CAST5. Nepoužíva žiadne patentovaný alebo inak obmedzený softvér alebo algoritmus. Namiesto toho používa voľne šíriteľné algoritmy.

Po dlhú dobu implicitne nepodporoval šifrovací algoritmus IDEA použitý v PGP, avšak od roku 2012, keď skončila platnosť posledného patentu pre tento algoritmus, to už nie je pravda. Napriek tomu sa nedoporučuje používať tento algoritmus a v programe len kvôli kompatibilite.

V momentálnej verzii 2.0.26 a 1.4.18 podporuje nasledujúce algoritmy:

- Asymetrické šifrovanie: RSA, ElGamal, DSA
- Symetrické šifrovanie: 3DES, IDEA, CAST5, Blowfish, Twofish, AES-128, AES-192, AES-256, Camellia-128, Camellia-192 a Camellia-256
- Hašovanie: MD5, SHA-1, RIPEMD-160, SHA-224, SHA-256, SHA-384, SHA-512
- Kompresia: ZIP, ZLIB, BZIP2

Novšie vydania GPG umožnujú použitie väčšiny kryptografických funkcií a algoritmov, ktoré poskytuje knižnica Libgcrypt, napríklad kryptografiu na báze eliptických kriviek (ECDSA, ECDH a EdDSA).[10]

GnuPG je navrhnutý pre Linux, avšak existuje jeho verzia pre Windows, ktorá sa nazýva Gpg4win.

2.5 Node.js

Node.js je open-source, multiplatformné interpretačné prostredie pre vykonávanie kódu v JavaScripte na strane servera. Historicky bol JavaScript využívaný primárne pre skriptovanie na strane klienta, kedy sú skripty napísané v JavaScripte obsiahnuté v HTML kóde webovej stránky a následne spustené pomocou JavaScript engine-u v prehliadači používateľa. Node.js umožňuje aby bol JavaScript použitý pre skriptovanie na strane serveru, čím umožňuje produkciu obsahu dynamických webových stránok predtým než je poslaný do prehliadača používateľa. Vďaka tomuto sa stal jedným zo základných prvkov paradigmatu „JavaScript všade“ (angl. „JavaScript everywhere“), umožnujúc zjednotenie webových aplikácií v jednom programovacom jazyku namiesto písania skriptov pre server v inom jazyku.

Bol vytvorený nad Google Chrome V8 engine-om a jeho ECMAScript-om, čo znamená že väčšina jeho syntaxe je podobná front-end JavaScript-u (ďalšej implementácii ECMAScriptu), vrátane objektov, funkcií a metód.

Umožňuje vytváranie webových serverov a sieťových nástrojov pomocou JavaScriptu a kolekcie modulov, ktoré zabezpečujú rôznorodú základnú funkcionalitu. Dostupné sú moduly pre vstupno-výstupné operácie nad súborovým systémom, sieťové protokoly (DNS, HTTP, TCP, UDP, TLS/SSL), binárne dáta (vyrovnávacia pamäť - Buffer), kryptografické funkcie, dátove toky a iné esenciálne funkcie. API modulov Node.js je navrhnuté tak, aby znížilo komplexnosť písania serverových aplikácií.

Aplikácie môžu bežať v operačných systémoch Linux, macOS, Microsoft Windows, NonStop a na UNIX-ových serveroch. Môžu byť napísané aj v jazyku CoffeeScript (alternatíve JavaScriptu), Dart alebo Microsoft TypeScript, či v akomkoľvek jazyku, ktorý je možné preložiť do JavaScriptu.

Primárne je určený pre tvorbu webových serverov. Najväčším rozdielom medzi Node.js a PHP je, že väčšina funkcií je blokujúcich, zatiaľ čo Node.js je navrhnutý na paralelný beh funkcií a používanie tzv. `callback`-ov.

Pre toto prostredie boli vytvorené tisíce knižníc – väčšina z nich je dostupná na webovej stránke npm. Komunita vývojárov má dve elektronické konferencie a tiež IRC kanál #node.js na servery freenode. Každoročne sa usporadúva konferencia vývojárov, ktorá sa nazýva NodeConf.

Open-source komunita vytvorila niekoľko serverových framework-ov na urýchlenie vývoja aplikácií. Sú to napríklad Connect, Express.js, Socket.IO, Koa.js, Hapi.js, Sails.js, Meteor, Derby a mnoho iných.[11]

2.6 UMD

V súčasnej dobe existujú na modularizáciu kódu v JavaScripte dve populárne knižnice, sú to AMD a CommonJS. Pri vývoji aplikácií sa však môže stať, že jednotlivé komponenty aplikácie nie sú navzájom kompatibilné, pretože každá je napísaná v inej špecifikácii. Keďže obidve knižnice sú veľmi populárne a zatiaľ nedošlo k dohode o používaní jednej z nich, tento problém rieši špecifikácia UMD. Kód napísaný v tejto špecifikácii síce nevyzerá veľmi prívetivo, no je kompatibilný s AMD, CommonJS a klasickou definíciou globálnej premennej.

Kapitola 3

Správca hesiel Pass

`Pass` je správca hesiel, ktorý sa riadi filozofiou Unix-u. Pre každé heslo existuje separátny súbor, zašifrovaný pomocou `gpg`, ktorého názov reprezentuje stránku alebo iný zdroj vyžadujúci heslo. Tieto súbory môžu byť usporiadané do zmysluplnej hierarchie priečinkov, skopírované z jedného počítača na druhý a spravované pomocou štandardných nástrojov príkazového riadka pre správu súborov.

Pass umožnuje veľmi jednoducho spravovať tieto individuálne súbory s heslami. Všetky heslá sú uložené pri základnom nastavení v priečinku `~/.password-store`. Aplikácia ponúka príkazy pre pridanie, úpravu, generovanie a získanie hesiel. Je to veľmi krátky a jednoduchý skript pre interpret príkazov (angl. `shell`). Je schopný požadovaného heslo vložiť do schránky (aby ho napríklad niekto nemohol prečítať z obrazovky) a heslá môžu byť verzované pomocou služby git.

Heslá nemusia byť spravované len cez pass, použité môžu byť aj klasické príkazy unixového príkazového prostredia. Nevyužíva žiadne nové formáty súborov alebo paradigmaty. Vytvorený je tiež program pre dokončovanie názvov pre `bash`, `zsh` a `fish`, vďaka čomu môže používateľ pomocou stlačenia klávesy tab doplniť názvy hesiel a príkazov. Pre tento program vytvorila komunita mnoho klientov a grafických rozhraní pre ostatné platformy a tiež rožšírenia pre `pass` samotný.

Úložisko používateľa nenúti použiť žiadnu konkrétnu schému alebo typ organizácie dát, keďže využíva len obyčajné textové súbory, ktoré môžu obsahovať akékoľvek dáta. Aj keď najčastejším využitím je uloženie jedného hesla v každom súbore, niektorí pokročilý používatelia doň ukladajú napríklad odpovede na tajné otázky, URL stránok a iné citlivé informácie alebo metadáta. Každý si môžu vybrať, ako si dáta zorganizuje. Existuje mnoho možností a pass poskytuje príkazy na niektoré z nich, napríklad príkaz `clip`, ktorý vyberie len prvý riadok súboru. Používateľ tak môžu mať na ďalších riadkoch uložené iné informácie.[3]

Pass využíva pre uloženie identifikátorov šifrovacích kľúčov súbory s názvom `gpg-id`. Je to jednoduchý textový súbor, v ktorom sú jednotlivé identifikátory oddelené medzerou. Každý priečinok, pre ktorý sú nastavené odlišné kľúče obsahuje tento súbor.

Kapitola 4

Vývojové nástroje

4.1 Gulp

Gulp.js je systém na automatizáciu opakujúcich sa úkonov pri vývoji webových aplikácií, ako sú minimalizácia, spájanie, testovanie, optimalizácia atď. Vyvinula ho spoločnosť Fractal Innovations a komunita na servery GitHub. Pre túto utilitu existuje množstvo doplnkov (momentálne viac ako 300), ktoré umožňujú splniť rôznorodé úkony. Implementuje rúry (angl. pipelines), vďaka čomu je možné výsledky operácií zreťaziť podobne ako v unixoových operačných systémoch. Jednotlivé úlohy sú uložené v súbore `gulpfile.js`, ako kód v JavaScript-e.

4.2 Browserify

Browserify slúži na nahrávanie modulov v JavaScripte, vďaka čomu obchádza momentálne chýbajúcu podporu importovania modulov v prehliadiačoch. Slúži ako „pre-procesor“ pre kód. Podobne ako je tomu v prípade rozšírení pre CSS, ako sú SASS a LESS, ktoré priniešli rozšírenú syntaktickú podporu do kaskádových štýlov, Browserify rozširuje klientské aplikácie v JavaScripte rekurzívnym prechádzaním ich zdrojového kódu a hľadaním globálnej funkcie `require`. Keď Browserify nájde takéto volania, ihneď načíta vyžadovaný modul (použitím rovnakej funkcie `require` aká je dostupná v node.js) a skombinuje ich to jedného zmenšeného súboru (bundle), ktorý môže byť načítaný v prehliadači.

Tento jednoduchý, no elegantný prístup umožňuje využiť potenciál a prívetivosť CommonJS(metóda, ktorou sú načítané moduly v node.js) v prehliadači. Navyše odstraňuje komplexnosť kódu vyžadovanú pre nahrávanie s nástrojmi používajúcimi Asynchronous Module Definition (AMD), ako napríklad RequireJS. [1, str. 101]

4.3 Browserify-derequire

Pri vytváraní verzií knižníc a aplikácií pre prehliadače pomocou nástroja Browserify, sú všetky volania funkcie `require` ponechané, aj keď je použitý mód `standalone`. Ak má byť balíček znova zabudovaný do inej knižnice, spôsobuje to ťažkosti. Tento doplnok pre `Browserify` premenuje všetky volania `require` tak, aby nespôsobovali konflikty.[4]

4.4 Uglifyify

Uglifyify je transformácia pre Browserify, ktorá dokáže vygenerovať lepší výstup individuálnym spracovaním súborov namiesto bežného spracovania ako celku vygenerovaného cez Browserify. Spolu s uglifyify by mal byť tiež použitý nástroj uglify, čím sa dá dosiahnuť najmenší výstupný súbor. Poskytuje dodatočnú optimalizáciu, avšak nie všetky optimalizácie, ktoré využíva uglify, takže nie je jeho náhradou.

4.5 JSDoc

JSDoc je program, ktorý sa používa na generovanie dokumentácie k zdrojovým súborom napísaným v JavaScripte. Je to jeden z najpoužívanejších nástrojov k tejto činnosti. Používa značkovací jazyk, ktorý je umiestnený v komentároch. Pomocou neho je možné popísať parametre funkcií, návratové hodnoty, výnimky, alebo aj štruktúru modulov. Syntax, ktorú používa je implementovaná aj v mnohých iných nástrojoch na dokumentáciu kódu, vďaka čomu nie je nutné učiť sa nový jazyk, ak má používateľ skúsenosti s dokumentovaním kódu.

4.6 Budo

Budo je vývojový server pre Browserify, ktorý značne uľahčuje inkrementálny vývoj. Umožňuje automaticky vytvoriť bundle pomocou browserify a obnoviť stránku v prehliadači. Takáto funkcionalita sa môže zdať ako zbytočná, no pri vývoji aplikácie pre prehliadač, kedy je nutné pri každej zmene kódu urobiť opakujúce sa úkony, je neocentiteľným pomocníkom.

4.7 Co

Co je knižnica, ktorá pomocou generátorov umožňuje prehľadný zápis kódu, ktorý používa JavaScript knižnicu Promise. Vďaka nej môže byť takýto kód zapísaný rovnako ako synchrónny. Kľúčové je slovo `yield`, vďaka ktorému generátor spustený touto knižnicou počká na dokončenie asynchrónnej operácie. Zároveň prívetivým spôsobom umožňuje spracovať prípadne chyby.

Kapitola 5

Dostupné riešenia

V súčasnosti existuje neprieberné množstvo rôznych správcov hesiel. Vďaka ním si nemusíme zapamätať veľké množstvo hesiel na rôznych stránkach a tiež nám ponúkajú možnosť generácie silných náhodných hesiel. Na druhej strane predstavujú riziko, pretože po zistení hlavného hesla má utočník prístup ku mnohým účtom. Pri hľadaní správcu hesiel pre node.js som ich veľa nenašiel.

5.1 unipassman

Tento balíček ponúka rozhranie cez príkazový riadok aj API. Zatiaľ však podporuje len KDE Wallet. Prvá verzia bola vypustená pred dvoma rokmi a doteraz bolo vydaných 5 verzií. Podľa informácií na jej stránke je stále vo vývoji.

5.2 sweetp-password-manager

Služba ponúka možnosť uloženia zašifrovaných hesiel lokálne v `json` súbore. Umožňuje prihlasovať sa k rôznym online službám pomocou príkazového riadka a tejto utility.

Žiadne z týchto ponúkaných riešení v node.js neposkytuje šifrovanie, ktoré by prešlo auditom a taktiež nepodporuje verzovanie cez git. Aj keď mnoho webových správcov, alebo aplikácií pre operačný systém podporuje synchronizáciu cez WebDAV, v node.js takéto riešenie zatiaľ chýba.

5.3 Rozšírenia pre Pass

Pre tohto správcu súborov existuje veľké množstvo rozšírení pre mnohé operačné systémy a aplikácie. To ešte viac dokazuje, aká obľúbená je táto krátka aplikácia pre bash.

Zoznam rozšírení:

- passmenu: Skript pre dmenu.
- qtpass: Multiplatformý grafický klient.
- Android-Password-Store: Android aplikácia.
- passforios: Aplikácia pre iOS.

- pass-ios: Staršia aplikácia pre iOS.
- passff: Plugin pre Firefox.
- browserpass: Plugin pre Chrome.
- Pass4Win: Klient pre Windows.
- pext_module_pass: Module pre Pext.
- gopass: Aplikácia pro GO GUI.
- upass: Používateľské rozhranie s interaktívnym terminálom.
- alfred-pass: Integrácia pre Alfred.
- pass-alfred: Integrácia pre Alfred.
- simple-pass-alfred: Integrácia pre Alfred.
- pass.applescript: Intergrácia do OS X.
- pass-git-helper: Uloženie prihlasovacích údajov k službe git.
- password-store.el: Balík pre Emacs.
- XMonad.Prompt.Pass: Aplikácia pre XMonad.

Pre túto utilitu je možné stiahnuť taktiež veľké množstvo skriptov pre konverziu dátových štruktúr z iných spŕavcov. Ich kompletný zoznam je možné nájsť na oficiálnej stránke Pass `https://www.passwordstore.org/`.

5.4 KeeWeb

KeeWeb je správca súbor (nielen) pre prehliadač. Je kompatibilný s aplikáciou KeePass a podporuje aj synchronizáciu cez WebDAV. Sama o sebe webová aplikácia neuskutočňuje žiadné externé požiadavky, je kompletne offline a všetky dáta su uložené lokálne. Taktiež neobsahuje žiadne skripty na štatistiku, alebo iné potenciálne bezpečnostné hrozby.

Grafické webové rozhranie tejto aplikácie je veľmi prehľadné a prívetivé. Aplikácia používateľa k ničomu nenúti. K dispozícii je aj skúšobné prostredie, v ktorom je možné otestovať dostupnú funkcionalitu.

KeeWeb je možné použiť aj priamo v operačnom systéme, jeho verzie existujú pre MacOS, Windows aj Linux.

Kapitola 6

Návrh

Táto časť práce sa zaoberá návrhom knižnice pre prácu s heslami. Nazval som ju JSPass. Jej funkcionalita má umožňovať šifrovanie a dešifrovanie hesiel (prípadne aj iných textových informácií) a tiež správy týchto informácií (premenovanie, presun, kopírovanie). Knižnica by mala byť schopná bezpečne uchovať dáta používateľa tak, aby sa k nim nedostala žiadna tretia strana. Na zabezpečenie obsahu hesiel je využitá knižnica OpenPGP.js, ktorá implementuje šifrovacie algoritmy pomocou kryptografie verejného kľúča.

OpenPGP.js taktež ponúka aj správu kľúčov používateľa, či už privátnych alebo verejných. Túto kľúčenku (angl. keyring) bude taktiež knižnica ponúkať cez svoje API. Vďaka tejto funkcii môže používateľ vyhľadať kľúč na serveri, uložiť alebo načítať uložené kľúče z perzistentnej pamäte. Navyše som sa rozhodol implementovať rozšírenie tohto správcu kľúčov, ktorý bude podporovať uchovanie odomknutého privátneho kľúča na stanovený čas v pamäti. Používateľ tak nemusí zakaždým zadávať heslo, ak bol kľúč nedávno použitý, no tiež nenecháva kľúč nechránený po príliš dlhú dobu.

Aby mohla táto knižnica fungovať na serveri aj v prehliadači, bude potrebné, aby rozpoznala, kde momentálne beží a na základe toho vybrala vhodnú funkciu k dosiahnutiu cieľa. Tak isto bude potrebné knižnicu exportovať tak, aby mohla byť v iných projektoch nielen ako modul pre node.js, ale tiež ako samostatný UMD modul.

Pre uloženie hesiel som sa rozhodol využiť stromovú štruktúru tak, ako to býva v súborových systémoch. Vďaka tomu je možné efektívne implementovať aj zložitejšie operácie.

6.1 Spracovanie asynchrónnych volaní

Pri implementácii tohto projektu sa na mnohých miestach budú využívať asynchrónne volania. Knižnica OpenPGP.js ich hojne vracia ako výsledok operácií šifrovania a dešifrovania, či generovania nových kľúčov. Pre tento účel je použitá moderná knižnica `Promise` a spracovanie zložítých operácií v oddelenom vlákne tak, aby bolo grafické prostredie aplikácií využívajúcich túto knižnicu vždy responzívne. Pre zachovanie koherencie kódu a aj jeho sprehľadnenie som sa rozhodol ju použiť tiež, oproti tradičným `callback`-om je to veľký rozdiel.

Jej podstata spočíva v tom, že funkcia, ktorá ju používa, vráti objekt Promise. Tento objekt môže mať v atribúte `then` funkciu, ktorá je zavolaná po ukončení asynchrónnej operácie, pomocou `resolve`. V prípade neúspechu operácie je možné zavolať `reject`, ktorý sa dá zachytiť pomocou pridania callback funkcie do atribútu `catch`.

6.2 Verzovanie pomocou git

Knižnica má podporovať verzovanie cez službu git. Používateľ tak môže jednoducho vrátiť vykonané zmeny. Vďaka tejto funkcii môžu viacerý ľudia zdieľať heslá a pridávať ich do aktuálnej verzie repozitára, kde sa zmeny zlúčia. To bolo pre mňa jednou z najťažších častí tejto práce. Pôvodne mala práca využívať knižnicu `js-git` od používateľa creatonix. Zo začiatku som skúšal použiť ju, no zdá sa, že niektoré jej funkcie už nefungujú, konkrétne to bolo ukladanie repozitára na git. Jej rozhranie tiež nie je veľmi používateľsky prívetivé a chýbajú návody na dosiahnutie základných operácií. Po študovaní fungovania tejto služby som zistil, že nie je možné priamo upravovať dáta na vzdialenom servery, tak ako to robí program git, pretože máloktorý server má povolené prijímanie požiadaviek od prehliadačov, ktorých sokety fungujú na inom princípe. Najprv ma napadla implementácia cez ďalší proxy server, ktorý by zabezpečoval klasické sokety operačného systému, čo by ale narušilo univerzálnosť knižnice. Rozhodol som sa teda pre použitie API rozhraní, ktoré poskytujú mnohé git servery.

6.3 Chybové stavy

Ak pri vykonávaní metódy dôjde ku chybovému stavu, knižnica vždy spustí výnimku. Pre takýto spôsob riešenia som sa rozhodol, pretože pri návratových kódoch môže dôjsť k prehliadnutiu chyby pri používaní knižnice. Takto používateľ musí explicitne spracovať chybový stav a ten sa neprenáša do ďalších častí programu. Tento prístup je tiež výhodnejší z hľadiska spracovania chybových stavov, keďže je možné ho odchytiť na akejkoľvek úrovni v programe, kde je to výhodné a kód je prehľadnejší, pretože nie je potrebné zakaždým overovať výsledok operácie.

6.4 Popis architektúry

Hlavným objektom, cez ktorý sa bude pristupovať k úložisku, je `JSPass`. Jeho modul nesie rovnaký názov. Bude poskytovať metódy na importovanie a generovanie kľúčov, vkladanie a získavanie hesiel z koreňového priečinka, vyváranie adresárov a na presun a kopírovanie položiek pomocou zadania absolútnej cesty. Bude obsahovať referencie na koreňový priečinok ako aj metódu na jeho získanie, pamäť odomknutých privátnych kľúčov, všetkých importovaných kľúčov a na rozhranie pre git. Pre rozhranie git budú dostupné metódy na nastavenie repozitára, jeho klonovanie a odoslanie na server. Bude obsahovať aj funkciu, pomocou ktorej sa bude dať vyhľadávať regulárnym výrazom v celom úložisku.

Pre pamäť odomknutých kľúčov bude využitý objekt s knižnice OpenPGP.js. Jeho rozhranie je dostupné na URL https://openpgpjs.org/openpgpjs/doc/module-keyring_keyring-Keyring.html.

Priečinok bude reprezentovať objekt `Directory`. Pomocou jeho metód bude možné spravovať heslá a priečinky, ktoré obsahuje – pridávanie, mazanie, zmena obsahu (u hesiel), kopírovanie a presun. Parametre týchto metód sa budú od správy cez hlavný objekt `JSPass` líšiť tým, že ich argumentom budú priamo cieľové objekty namiesto absolútnej cesty. Bude možné v ňom vyhľadávať pomocou vzoru pre regulárny výraz. Taktiež bude možné získať identifikáciu jeho kľúčov a nastaviť nové kľúče pre priečinok. Obsahovať bude referencie na obsiahnuté heslá a priečinky, jeho názov, identifikáciu šifrovacích kľúčov (ak budú pre neho nastavené) a referenciu jeho rodičovského adresára.

Obr. 6.1: Návrh prototypov

Každé heslo bude uložené v objekte s prototypom `Password`. Rovnako ako priečinok bude umožňovať presun a kopírovanie hesla, premenovanie a získanie alebo nastavenie nového obsahu hesla. Jeho funkcionalita bude ďalej zabezpečovať získavanie identifikátorov kľúčov, pomocou ktorých bolo zašifrované a metódu na zistenie dešifrovateľnosti hesla (prítomnosť validných odomknutých privátnych kľúčov). Medzi jeho vlastnosti bude patriť odkaz na rodičovský priečinok, jeho názov a obsah.

Odomknuté privátne kľúče budú uložené v objekte `privateKeyCache`, ktorý ich automaticky vymaže po stanovenom čase. Bude obsahovať čas, počas ktorého majú byť kľúče uložené a samotné kľúče, ku ktorým sa bude pristupovať cez Proxy objekt. Ten zabezpečí, že pri uložení nového kľúča, alebo prístupe k už existujúcemu kľúču bude čas do vymazania inicializovaný respektíve obnovený.

Pre prístup k službe `git` bude použité rozhranie s rovnomenným názvom. Prototypy kompatibilné s týmto rozhraním budú musieť implementovať funkcie `createAuthToken`, `setToken`, `setRepository`, `clone`, `commit`, `changeContent`, `createFile`, `moveFile`, `renameFile` a `deleteFile`. V knižnici bude dostupný modul pre GitHub, ktorý pomocou HTTPS požiadaviek zabezpečí odosielanie a prijímanie dát zo servera.

Kapitola 7

Implementácia

Táto kapitola sa venuje implementácii knižnice. Sú v nej obsiahnuté riešenia zaujímavých problémov, ktoré boli načrtnuté v kapitole o návrhu. V jednotlivých častiach sú popísané aj funkcie, pomocou ktorých je možné s úložiskom pracovať. Najprv je čitateľ oboznámený s konkrétnou hierarchiou uloženia hesiel a spôsobom práce s priečinkami. Nasleduje časť o získavaní a ukladaní šifrovaného obsahu. Pre tento účel sú potrebné odomknuté privátne kľúče, ktorých implementácia je popísaná v podkapitole 7.4. Ďalej je možné dozvedieť sa o exportovaní na úložiska na git, ukladaní do indexovanej databázy a nakoniec aj o vytváraní výsledného balíka, ktorý môže byť použitý samostatne v ďalších projektoch. Jazyk JavaScript je kvôli jeho benevolentnosti náchylný k niekedy ťažko odhaliteľným chybám, vo všetkých moduloch je preto použitý striktný mód, aby sa tak aspoň čiastočne zabránilo týmto problémom.

7.1 Stromová štruktúra

Pre uloženie adresára existuje prototyp `Directory`. Objekt tohto prototypu obsahuje heslá, ktoré sú uložené v adresári a tiež adresáre, ktoré sú jeho potomkom. Tieto objekty sú uložené v dynamickom poli, jedno pre heslá a jedno pre adresáre. K súborom sa je možné dostať cez metódy objektu Directory. K dispozícii sú metódy pre pridanie, získanie a vymazanie hesla pomocou jeho mena. Používateľ má tiež možnosť dostať všetky heslá v adresári. Rovnaké metódy sú dostupné pre obsiahnuté adresáre s doplnením o funkciu pridania adresára rekurzívne.

Ak sú pre adresár priradené kľúče, ich identifikácia je uložená taktiež v tomto objekte. Ich implementácia odráža štruktúru adresárov štandardného správcu hesiel v UNIX-e Pass, vďaka čomu je možné knižnicu použiť s už existujúcimi úložiskami hesiel vytvorenými pomocou tohto programu. Pri ukladaní sú kľúče uložené ako textový súbor s medzerou medzi jednotlivými identifikátormi, jeho názov je gpg-id. Pri zmene kľúčov pre adresár sú všetky heslá v tomto priečinku a jeho podpriečinkoch znova zašifrované na nové kľúče, s výnimkov podadresárov, pre ktoré sú už nastavené iné kľúče. Pre tieto účely existuje metóda `reencryptTo`, avšak používateľ by ju nemal využívať priamo, ale cez metódu `setKeyIds`.

7.2 Absolútna cesta vs. objekty

V rôznych situáciach môže byť výhodnejšie spracovanie hesiel rôznymi spôsobmi. Knižnica preto ponúka dva spôsoby manipulácie s nimi.

Prvým z nich je prístup k heslám a priečinkom pomocou celej cesty. K získaniu objektu sú pre túto metódu k dispozícii funkcie `getPasswordsByPath`, `getDirectoryByPath` a `getItemByPath`. Posledná z nich uprednostňuje výber hesla, no ak neexistuje, hľadá adresár s týmto názvom. Používateľ môže takisto plnou špecifikáciou cesty heslá a adresáre presúvať, premenovať, vymazávať alebo inak upravovať.

Ak je výhodnejšia práca cez objektové rozhranie, nad každým objektom hesla či adresára je možné volať ich metódy pre vyššie spomenuté akcie. Pri presúvaní či kopírovaní sú v tomto prípade ich parametrami priamo cieľové objekty priečinkov.

7.3 Ukladanie hesiel

Pre uloženie hesiel existuje prototyp `Password`. Tento objekt zabezpečuje funkcie pre získanie obsahu hesiel, ich uloženie, kopírovanie, či presun hesiel. Posledné dve operácie implicitne zaručujú, že heslo bude šifrované pre kľúče nového umiestnenia, ak sú rozdielne.

V objekte existuje metóda `isDecryptable`, ktorá by mala byť zavolaná pred snahou o dešifrovanie hesla. Ak existuje odomknutý privátny kľúč pre toto heslo, metóda vracia pravdivý výsledok. V opačnom prípade je nutné zavolať metódu `getAvailableKeysInfo`, ktorá poskytuje informácie o identifikácii dostupných kľúčoch a ich používateľoch. Pomocou tejto metódy môže používateľ jednoducho dostať sadu kľúčov pre heslo a vybrať si, ktorý z nich následne odomkne. To je možné pomocou metódy `DecryptKey`, ktorej parametrom sú odtlačok alebo dlhá identifikácia kľúča a heslo na jeho odomknutie. Po odomknutí je automaticky pridaný do pamäte odoknutých kľúčov a zotrvá tam po dobu, ktorá je natavená pre úložisko hesiel buď pri incializácii úložiska, alebo neskôr počas chodu, cez `setUnlockedTime`.

7.3.1 Práca s OpenPGP.js

7.4 Uchovávanie odomknutých privátnych kľúčov

Pri používaní programu gpg v linuxe existuje démon (angl. daemon), ktorý prácu s heslami uľahčuje tým, že uchová odomknutý privátny kľúč po užívateľom stanovenú dobu v pamäti. Používateľ tak nemusí znova zadávať heslo, ak bol kľúč nedávno použitý a tým uľahčuje prácu. Túto funkcionalitu v knižnici zabezpečuje prototyp `UnlockedKeyring`.

Najprv som ju implementoval pomocou Proxy objektu, cez ktorý sa pristupovalo k poľu, v ktorom sú uložené privátne kľúče knižnice OpenPGP.js. Neskôr som toto rozhodnutie zmenil, pretože ho podporujú len novšie verzie prehliadačov a nedá sa preložiť do staršej špecifikácie ECMAScript, keďže je zabudovaný na hlbokej úrovni interpreta jazyka.

Prototyp `Keyring` používa na ukladanie kľúčov objekt `KeyArray`. Prekryl som jeho metódy a pridal vlastný kód. Pri uložení nového kľúča je zavolaná funkcia `setTimeout` s časom, ktorý nastavil používateľ. Po uplynutí času je zavolaná anonymná funkcia, ktorá kľúč vymaže. Výsledný identifikátor, vďaka ktorému je možné zrušiť toto volanie, je uložený spolu s kľúčom. Pri každom ďalšom použití kľúča je volanie zrušené pomocou funkcie `clearTimeout` a čas pre prístup ku kľúču je obnovený na vopred stanovenú hodnotu.

7.5 Ukladanie na git

Pred akoukoľvek prácou s repozitárom musí používateľ najprv zavolat funkciu `gitInit`, ktorá nastaví cestu k repozitáru a tiež voliteľne agenta používateľa, ktorý musí povinne

obsahovať každá hlavička HTTPS požiadavku ku API. Taktiež automaticky vyberie vhodné API pre server ak je podporovaný.

Následne je pre prácu s už existujúcim repozitárom potrebné zavolať funkciu `clone`, ktorá vytvorí lokálnu kópiu repozitára. Všetky ďalšie zmeny v úložisku sú potom premietané do tejto lokálnej kópie.

Knižnica momentálne podporuje GitHub API v3, no je možné pridávať ďalšie servery pod podmienkou, že budú implementovať rozhranie pre git. Toto rozhranie musí implementovať nasledujúce funkcie:

- setRepository(url, userAgent) - Nastaví URL adresu repozitára a identifikáciu agenta pre posielanie požiadavkov.
- clone(): Array<Object> - Stiahne všetky súbory z repozitára a vráti ich v poli objektov. Každý objekt musí obsahovať premenné path, kde je definovaná cesta a content, kde je definovaný obsah v kódovaní Base64.
- commit(message): Promise - Vytvorí nový commit s aktuálnym obsahom lokálneho stromu a odošle zmeny na server.
- changeContent(path, newContent) - Zmení obsah súboru.
- createFile(path, content) - Vytvorí nový súbor.
- moveFile(oldPath, newPath) - Premiestni súbor.
- renameFile(oldPath, newPath) - Premenuje súbor.
- deleteFile(path) - Vymaže súbor.

Ak je pre úložisko súborov nastavený git, všetky zmeny v ňom sa automaticky odrážajú v zmene stromu, ktorá je uložená lokálne v pamäti, podobne ako je tomu u klasického programu git. Pri zavolaní funkcie `commit` sa najprv vytvorí pre každý zmenený súbor jeho `blob` na serveri, odošle sa aktualizovaný strom a jeho commit, ktorý sa nastaví ako posledný pre vetvu `master`.

7.5.1 OAuth 2.0 vs. základná autentifikácia

Väčšina moderných serverov používa OAuth autentifikáciu. Výnimkou nie je ani GitHub, pre ktorý som vytvoril metódy používajúce jeho API. Táto autentifikácia spočíva v presmerovaní priamo na stránky GitHub, kde sú poslané aj údaje zvyšujúce bezpečnosť a identifikujúce aplikáciu, ktorá požaduje prístup. Používateľ je po prihlásení presmerovaný na prednastavenú adresu, prípadne adresu zaslanú spolu s požiadavkou. Aplikácia následne obdrží autorizačný kód, ktorý zamení za token, s ktorým môže neskôr vykonávať zmeny v repozitári. Po obdržaní tokenu ho používateľ nastaví pomocou funkcie setToken. Pri ďalších volaniach bude automatický použitý v hlavičke ako `Authorization: token obdrzany-token`.

Ak z nejakého dôvodu, napríklad pre testovanie používateľ nechce použiť autentifikáciu cez OAuth, token môže získať cez `Authorizations API` pomocou funkcie `createAuthorizationToken`, ktorej parametrami sú meno a heslo používateľa. Takýto spôsob však nemusí byť dostupný na všetkých serveroch, keďže závisí od poskytovaného aplikačného rozhrania.

7.5.2 Práca s GitHub API v3

Aby sme získali súbory, ktoré sú uložené v repozitári, musíme postupne prejsť niekoľkými krokmi. Najprv je nutné zistiť `SHA` haš posledného pridaného commit-u. Táto hodnota sa nachádza vo vetve `master`. Môžeme sa k nemu dostať pomocou požiadavky

```
GET /repos/:owner/:repo/git/refs/heads/master
```

`:owner` nahradíme konkrétnym používateľom a `:repo` názvom repozitára, s ktorým pracujeme. Ak všetko prebehne hladko v odpovedi získame jeho hodnotu. Teraz môžeme získať commit samotný pomocou

```
GET /repos/:owner/:repo/git/commits/:sha
```

kde `:sha` nahradíme získanou hodnotou. V tomto okamihu potrebujeme získať posledný strom (angl. tree). Jeho haš je vo výsledku odpovede servera. Pošleme ďalšiu požiadavku

```
GET /repos/:owner/:repo/git/trees/:sha
```

v odpovedi získame štruktúru súborov repozitára, avšak nie ich obsah. Každý súbor je potrebné stiahnuť pomocou ďalšej požiadavky

```
GET /repos/:owner/:repo/git/blobs/:sha
```

vo výsledku každej takejto požiadavky získame obsah súboru zakódovaný v `Base64`.

Pri ukladaní hesiel na git je potrebné ich obsah zakódovať. K tomuto účelu sa používa kódovanie Base64. V prostredí node.js to dosiahneme zavolaním funkcie `Buffer`, ktorá je vstavaná a následne konverziou výsledneho objektu metódou `toString` s parametrom `„base64"`. V prehliadači je situácia o niečo zložitejšia a najprv musíme hodnoty nachádzajúce sa v UInt8Array (poli kladných bajtových celých čísel) konvertovať na reťazec pomocou funkcie `String.fromCharCode`. Výsledný reťazec použijeme ako vstupný argument pre funkciu `btoa`. Takto spracovaný obsah hesla je možné zaslať na server pomocou požiadavky

```
POST /repos/:owner/:repo/git/blobs
```

Potrebujeme pripojiť dva parametre:

- content - obsah súboru
- encoding - „utf-8“ alebo „base64“, pre zašifrované heslá knižncia využíva base64, súbory s identifikátormi kľúčov sú zaslané v utf-8, pričom na servery sú taktiež automaticky skonvertované na base64.

7.5.3 Spájanie stromov

Ak jedno verzované úložisko zdieľajú viacerý používatelia, môže sa stať, že lokálny strom uložený v pamäti bude zastaralý. Pre odoslaním zmien na server je preto overené, či náhodou nebol strom aktualizovaný. V kladnom prípade sú najprv zmeny zlúčené. V momentálnej implementácii sa vždy zachovávajú novšie zmeny, teda ak jeden užívateľ najprv súbor vymazal a druhý zmenil jeho obsah, v novom strome bude súbor so zmeneným obsahom.

7.6 Ukladanie do indexovanej databázy

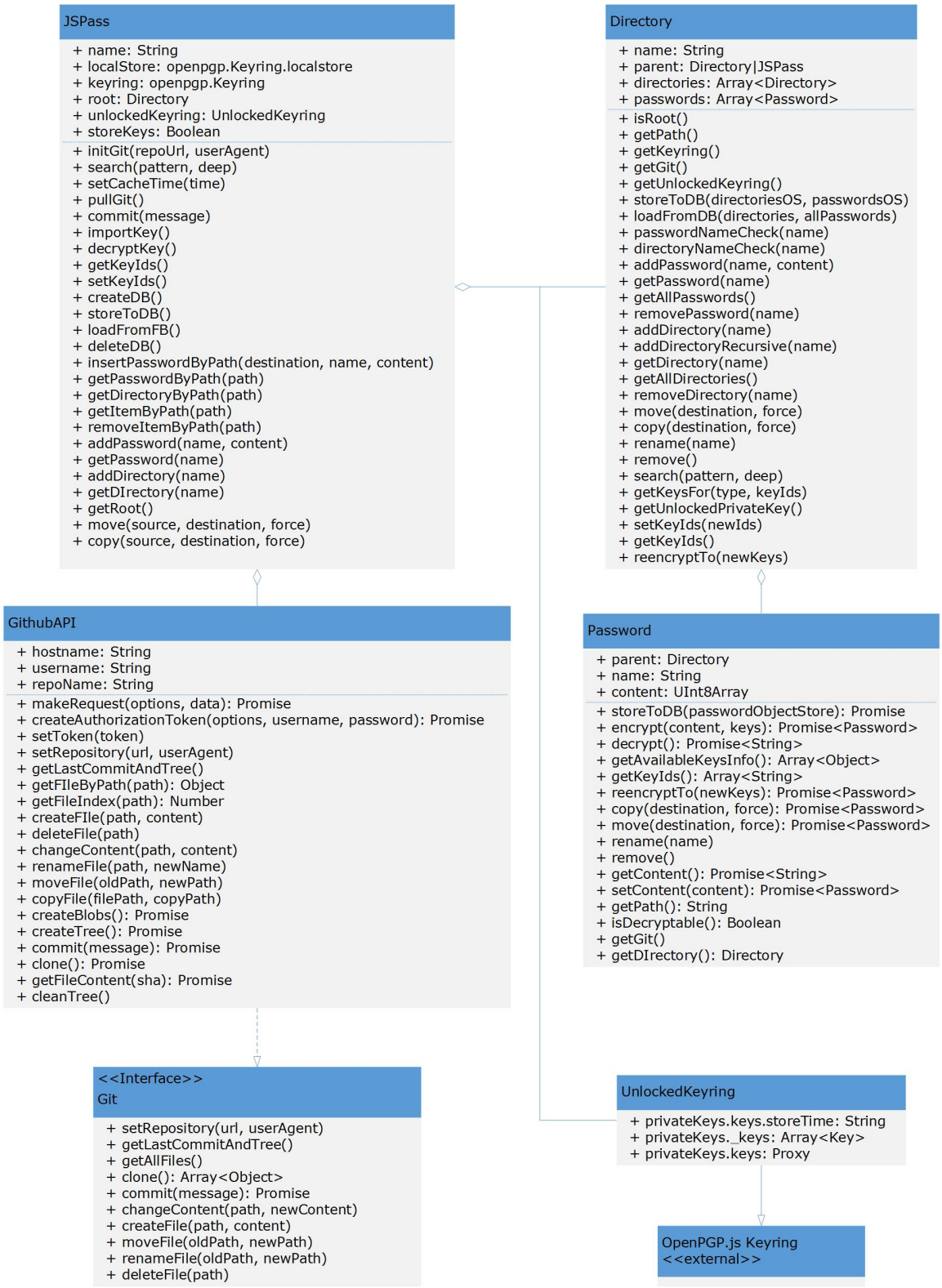
V projekte som sa rozhodol implementovať ako alternatívu k službe git uloženie do lokálnej indexovanej databázy. V tejto databáze prestavuje `OjectStore` niečo podobné, ako tabuľka v klasickej SQL databáze. Vytvoril som jeden pre adresáre a jeden pre heslá. Ďalej bolo potrebné vytvoriť indexy, tie predstavujú stĺpce v relačnej databáze a vytvárajú sa pomocou funkcie `createIndex` v spracovaní udalosti `onupgradeneeded`.

7.7 Podpora pre servery aj prehliadače

Pri interpretácii kódu musí knižnica zistiť, v akom prostredí momentálne beží. Toto je možné vďaka prítomnosti špecifických premenných, ktoré existujú v node.js a prehliadačoch. Konkrétne v node.js je to premenná `global` a v prehliadači `window`. Na základe toho sa určí, aká metóda bude použitá pre asynchrónne volania na servery git. V node.js je to knižnica `https`, v prehliadači zasa `XMLHttpRequest`.

Z dôvodu čo najväčsej možnej kompatibilite som vytvoril aj balíček, ktorý môže byť použitý ako UMD modul. Ten je vytvorený pomocou `browserify`.

Knižnica `https`, musí byť nastavená ako externý zdroj, inak by bola nesprávne exportovaná a výsledný balík by nefungoval v prostredí node.js. V ďalšom kroku je kód preložený do špecifikácie ES2015, aby mohol byť zminimalizovaný pomocou `uglifyify`. V takýchto `standalone` balíčkoch je ešte potrebné odstrániť volania funkcie `require`, ktorá nie je všade dostupná. To je zabezpečené pomocou `derequire`. V poslednom kroku je výsledný balík uložený.



Obr. 7.1: Diagram prototypov implementovanej aplikácie

Kapitola 8

Bezpečnosť

Pri práci s citlivými údajmi, akými heslá sú, je vždy potrebné klásť dôraz na bezpečnosť. Knižnica, ktorú používam na šifrovanie hesiel prešla dvoma auditmi nemeckej firmy Cure53. Výsledky prvého auditu sú zverejnené na https://github.com/openpgpjs/openpgpjs/wiki/Cure53-security-audit. Počas používania JSPass sa heslo k privátnym kľúčom nikdy neukladá, po použití na odomknutie privátneho kľúča je vymazané. Najväčším rizikom je uloženie tohto nezašifrovaného kľúča, keďže by mohol byť použitý na prístup k údajom používateľa. Nikdy sa neukladá perzistentne a ostáva iba v pamäti po dobu určenú pri inicializácii a nie je posielaný prostredníctvom internetu. Ak by však útočník mal fyzický prístup k ziariadeniu, mohlo by byť kompromitované. Toto je však otázkou obozretnosti konkrétneho používateľa. Získať podpisy pre dôveru kľúča je zdĺhavá a dosť náročná záležitosť, preto by mali byť pre prístup radšej použité podkľúče, ktorej môžu byť v prípade kompromitácie zrušené a útočník sa tak nedostane k ďalším citlivým informáciám. Podkľúče sú súčasťou OpenPGP a ponúkajú možnosť vytvorenia kľúča, ktorý je podriadený hlavnému kľúču. Môžu byť identifikované pomocou ich vlastného odtlačku, no väčšinou sú spojené s odtlačkom hlavného kľúča.

Do úložiska môžu byť taktiež pridané ďalšie nežiadané súbory, pretože ku verajným kľúčom má prístup každý. V tomto prípade ide znova o fyzickú bezpečnosť konkrétneho zariadenia a zabezpečenie účtov na službe git, ak ju úložisko využíva.

Všetky prenosy prebiehajú cez zabezpečený protokol HTTPS, ktorý by mal byť dostatočne bezpečný, no aj v prípade prelomenia by sa útočník dostal len k zašifrovanému obsahu.

Kľúče môžu byť uložené v `localStorage`, čo je úložisko prehliadača. Knižnica poskytuje možnosť túto funkcionalitu vypnúť, v tomto prípade je nutné nahrať kľúče po každom ukončení práce s úložiskom. Rovnako je možné uložiť heslá do indexovanej databázy prehliadača. V obidvoch prípadoch je znova kritickým prvkom fyzická bezpečnosť zariadenia, na ktorom sú uložené.

Rovnako dôležitá je aj komplexnosť použitých hesiel. Jednoduché heslá sa môžu stať ľahko obeťou slovníkových útokov, kratšie z nich môžu byť prelomené aj brute-force metódou.

Z môjho pohľadu je knižnica najviac náchylná na XSS (Cross-Site Scripting) útok. Používateľ pri ňom nevedomky pošle aplikácii požiadavku, ktorá ako parameter obsahuje škodlivý kód. Ak s ním nie je zaobchádzané opatrne (nie je prevedený escape), môže zaslať citlivé údaje uložené v pamäti na server útočníka. Úspešnosť takéhoto útoku zaléži na skúsenostiach vývojára, ktorý sa rozhodne knižnicu použiť.

Kapitola 9

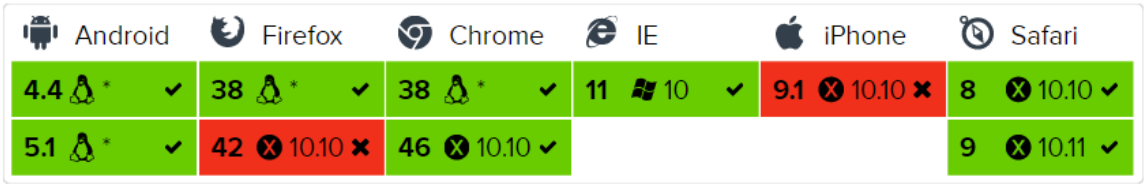
Podpora prehliadačov

Pri vytváraní akejkoľvek aplikácie pre web je dôležitá podpora čo najväčšieho množstva prehliadačov. Nie je to vždy možné, keďže niektoré operácie môžu byť implementované iba v novších verziách a ich alternatíva v starších výrazne ovplyvňuje výkon, alebo zväčšuje veľkosť celkovej knižnice. Snažil som sa, aby zbytočne nevyužívala najnovšie funkcie a mohlo ju tak použiť čo najviac ľudí, hlavne v prostrediach, kde musí byť použitá staršia verzia. Odhad skutočného podielu prehliadačov na trhu nie je veľmi exaktná veda a štatistiky použitia konkrétneho prehliadača sa môžu líšiť aj v desiatkach percent, väčšinu trhu však tvorí Chrome, Firefox, Internet Explorer, Edge, Safari a Opera. Najviac ľudí si podľa všetkých štatistík obľúbilo Chrome, okrem tabletov, kde vedie Safari.

V knižnici som najprv používal knižnicu `Proxy`. Keďže Internet Explorer ju nepodporuje vôbec, Chrome až od 25. februára 2016, Safari ešte neskôr, nakoniec som sa rozhodol pre iný spôsob implementácie cez prekrývanie pôvodných metód. Viac je o tom napísané v 7.4.

V zdrojovom kóde používam deklaráciu let, ktorá sa musí objaviť v cykloch for..of využitých v niektorých funkciách (let sa dá prípadne zameniť za const). Tá je pri vytváraní balíčka zamenená za podporovanejšiu konštrukciu.

Knižnica ako celok je po preložení pomocou `Babel` kompatibilná so špecifikáciou ECMAScript 5. Sama nepoužíva žiadnu špeciálnu funkcionalitu, no je obmedzená knižnicou OpenPGP.js, ktorá vyžaduje funkciu prehliadača `window.crypto.getRandomValues`. Podporu je možné vidiet na nasledujúcom obrázku:



Obr. 9.1: Podpora prehliadačov knižnicou OpenPGP.js v2.5.4

Kapitola 10

Záver

Cieľom tejto práce bola implementácia webového rozhrania správcu hesiel, ktorý je možné použiť na servery aj v prehliadači. Štruktúra súborov výslednej aplikácie je spätne kompatibilná so štandardným správcom hesiel v Unix-e Pass. Pomocou knižnice OpenPGP.js podporuje šifrovanie hesiel cez kryptografiu verejného kľúča, čím zvyšuje bezpečnost dát v dobe, kedy majú informácie čoraz väčšiu hodnotu a čoraz väčší počet jednotlivcov sa snaží o ich kompromitáciu. Aplikácia zabezpečuje verzovanie dát na server GitHub pomocou požiadaviek na API tejto stránky. Taktiež umožňuje základné úkony správy úložiška, akými sú presun, kopírovanie a ďalšie akcie, na ktoré sme zvyknutí z operačných systémov. Pre jednotlivé priečinky môžu byť nastavené rôzne kľúče, čo dovoľuje používateľovi lepšiu správu prístupu k podadresárom. Importované kľúče je možné prezistentne uložiť v LocalStorage, aby nemuseli byť znovu pridávané pri každom použití. Počas behu programu sú odomknuté privátne kľúče ponechávané v pamäti po stanovenú dobu, vďaka čomu nie je nutné (hlavne pri dávkových operáciách) zakaždým zadávať heslo na dešifrovanie kľúča.

Hlavným prínosom oproti existujúcim riešeniam je verzovanie úložiska. Táto funkcionalita môže byť veľmi užitočná pre organizácie alebo skupiny ľudí. Môžu ho zdielať viacerí používatelia, zmeny sa zlúčia a história zostáva uložená. Oprávnenia pre prístup k údajom sú nastaviteľné pomocou priradenia rôznych kľúčov pre podadresáre.

Knižnica je jednoducho použiteľná vo viacerých vývojových prostrediach. Jedným z možných rozšírení je podpora nových serverov pre systém git. Užitočný by tiež mohol byť export úložiska do zip archívu a podpora ďalších verzovacích systémov, napr. WebDAV.

Mojou najväčšou motiváciou bolo zverejnenie tohto programu na internete tak, aby ho mohol hocikto využiť bez poplatkov. V minulosti som rozmýšľal, že prispejem svojou troškou ku čoraz väčšiemu počtu projektov s otvoreným kódom. Preto som rád, že sa mi to podarilo aj zrealizovať. Projekt je zverejnený pod licenciou MIT na adrese https://github.com/pr0digi/jspass a je tiež zverejnená ako NPM balík na https://www.npmjs.com/package/jspass.

Literatúra

[1] Ambler, T.; Cloud, N.: *JavaScript Frameworks for Modern Web Dev*. Apress, 2015, ISBN 978-1484206638.

[2] Cash, F.: *OpenPGP JavaScript Implementation*. [Online; navštívené 07.05.2017].
URL https://openpgpjs.org/

[3] Donenfeld, J. A.: *Pass: The Standard Unix Password Manager*. [Online; navštívené 13.05.2017].
URL https://www.passwordstore.org/

[4] Engelschall, R. S.: *NPM: browserify-derequire*. [Online; navštívené 13.05.2017].
URL https://www.npmjs.com/package/browserify-derequire

[5] Group, O. W.: *OpenPGP - History*. [Online; navštívené 07.05.2017].
URL http://openpgp.org/about/history/

[6] Haverbeke, M.: *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press, 2010, ISBN 978-1593275846.

[7] Ofcom: *UK adults taking online password security risks*. [Online; navštívené 13.05.2017].
URL https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2013/uk-adults-taking-online-password-security-risks

[8] Parker, D.: *JavaScript with Promises: Managing Asynchronous Code*. O'Reilly Media, 2015, ISBN 978-1449373214.

[9] Siber Systems, I.: *Password Security Survey Results - Part 1*. [Online; navštívené 13.05.2017].
URL https://www.roboform.com/blog/password-security-survey-results

[10] Wikipedia: *GNU Privacy Guard*. [Online; navštívené 07.05.2017].
URL https://en.wikipedia.org/wiki/GNU_Privacy_Guard

[11] Wikipedia: *Node.js*. [Online; navštívené 12.05.2017].
URL https://en.wikipedia.org/wiki/Node.js

Príloha A

Obsah CD

Priložené CD obsahuje:

- `jspass` – Zdrojové súbory a dokumenmtáciu ku JSPass.
- `BP` – Zdrojové súbory tejto bakalárskej práce v jazyku LaTeX.
- `xchrou02-web-rozhrani-pass.pdf` – Bakalárska práca v elektronickej podobe.
- `xchrou02-web-rozhrani-pass-zadanie.pdf` – Bakalárska práca v elektronickej podobe spolu so zadaním.