Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Webovské rozhranie pre bezpečné zdieľanie dokumentov v cloude

bakalárska práca

2015

Peter Kovács

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Webovské rozhranie pre bezpečné zdieľanie dokumentov v cloude

bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Rjaško, PhD.

Bratislava, 2015

Peter Kovács

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Peter Kovács
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Webovské rozhranie pre bezpečné zdieľanie dokumentov v cloude
Web Application for Secure Document Sharing in Cloud

Cieľ: Navrhnúť a naprogramovať web aplikáciu, ktorá umožní používateľom bezpečne zdieľať a ukladať dokumenty v cloude. Aplikácia musí zabezpečiť, aby dokumenty boli uložené na serveri v šifrovanej podobe a server nemal prístup k dokumentom v otvorenom tvare. Zároveň treba navrhnúť praktický spôsob na distribúciu kľúčov k jednotlivým dokumentom medzi používateľmi, spôsob zdieľania šifrovaných dokumentov (resp. kľúčov na dešifrovanie týchto dokumentov) a revokáciu použitých kľúčov. Celá aplikácia bude na klientovi bežať v prostredí internetového prehliadača.

Vedúci: RNDr. Michal Rjaško, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 23.10.2014

Dátum schválenia: 28.10.2014 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Ďakujem RNDr. Michalovi Rjaškovi, PhD. za cenné rady, pripomienky a odborné vedenie pri vypracovávaní bakalárskej práce.

Abstrakt

Práca sa zaoberá implementáciou webovského rozhrania, ktoré bude slúžiť na bezpečné zdieľanie dokumentov v cloude. Toto rozhranie tvorí vrstvu medzi cloudom a používateľom, ktorá šifruje všetky odchádzajúce dáta a dešifruje prichádzajúce. Implementovali sme možnosť zdieľať šifrované súbory a tiež zrušenie zdieľania. Systém sme navrhli tak, aby tretia strana nemohla získať informácie o prenášaných dátach v otvorenom tvare.

Kľúčové slová: web aplikácia, cloud, bezpečné zdieľanie dát

Abstract

The thesis focuses on an implementation of the web interface for secure file sharing in cloud. The interface is supposed to create a layer between cloud and the user which will encrypt any outgoing data and decrypt all incoming data. We have implemented functionality to share and unshare files in a secure manner. System was designed to prevent the third party from obtaining the unencrypted data.

Keywords: web application, cloud, secure data sharing

Obsah

Zoznam obrázkov

Úvod

V práci sa budeme venovať vývoju webového rozhrania pre bezpečné zdieľanie dokumentov v cloude. Ukážeme si, ako je možné využiť existujúce cloudové riešenia a zostrojiť vrstvu medzi nimi a používateľom, ktorá bude zabezpečovať kryptografiu a bezpečnosť dát. Systém sa pokúsime navrhnúť tak, aby server a cloud mali čo najmenej informácií o súboroch alebo o kľúčoch v otvorenom tvare. Nakoniec budeme schopní nahrávať, sťahovať a zdieľať šifrované dáta uložené v Google Drive cloude.

Systém s takýmito vlastnosťami môže mať široké uplatnenie. Firemní zamestnanci často používajú cloudové úložiská na zdieľanie dát, ktoré môžu obsahovať obchodné tajomstvá či iné citlivé informácie. V takom prípade je dôležité zabezpečiť dôvernosť informácií, čo je hlavnou úlohou nášho riešenia. Téma bezpečného zdieľania dát sa dostala do povedomia aj laickej spoločnosti v roku 2013 po zverejnení dokumentov o sledovacích aktivitách americkej NSA a znova v roku 2014 po úniku intímnych fotografií z iCloudu niektorých hollywoodskych hviezd. Vďaka tomu môže byť naše riešenie zaujímavé nielen v korporátnej sfére ale aj pre bežného používateľa, ktorý chce svoje dáta chrániť.

Prácu rozdelíme do štyroch kapitol. V prvej zadefinujeme základné pojmy, vysvetlíme, čo je šifrovanie a aké šifrovanie budeme používať. V ďalšej kapitole poskytneme prehľad existujúcich riešení pre bezpečné zdieľanie dát, popíšeme ich funkcionalitu a vymenujeme ich nevýhody. Ďalej ukážeme, ako bude vyzerať naše riešenie a v čom sa bude odlišovať od ostatných.

V tretej kapitole navrhneme spôsob jeho fungovania. V úvode návrhu po-

píšeme, čo musíme spraviť predtým, ako budeme schopní pracovať s dátami. Ďalej si ukážeme, ako bude fungovať nahrávanie, sťahovanie súborov a práca so zdieľanými dátami.

Posledná kapitola sa bude zaoberať implementáciou návrhu. Uvedieme v nej technológie, ktoré sme využili na strane klienta, a ktoré sme použili na strane servera. Zároveň ukážeme najdôležitejšie segmenty zdrojového kódu a vysvetlíme ich fungovanie.

Kapitola 1

Základné pojmy a označenia

Prvá kapitola slúži ako teoretický úvod do práce. Zadefinujeme základné pojmy kryptografie, vysvetlíme, čo je symetrické a asymetrické šifrovanie a ako fungujú. Zároveň si vysvetlíme, čo sú hašovacie funkcie a cloudové úložiská. Budeme vychádzať z knihy Handbook of Applied Cryptography [1].

1.1 Kryptografia

Základným cieľom kryptografie je ukladanie a prenášanie dát vo forme, ktorú dokáže spracovať iba tá entita, ktorej sú dáta určené.

1.2 Množiny a operácie

Zadefinujeme si základné množiny a operácie nad nimi, ktoré budeme používať.

- Množinu $\mathcal{A}$ budeme nazývať abecedou. Abecedou je napríklad slovenská abeceda alebo $\mathcal{A} = \{0, 1\}$ je binárnou abecedou.
- Množina $\mathcal{M}$ je množina všetkých možných správ nad danou abecedou $\mathcal{A}$. Napríklad nad abecedou $\mathcal{A} = \{0, 1\}$ pri správach maximálnej dĺžky

2 je $\mathcal{M} = \{00, 01, 10, 11\}$.

- Množina $\mathcal{C}$ obsahuje všetky šifrované správy nad danou abecedou $\mathcal{A}$.
- Množinu $\mathcal{K}$ nazveme množina kľúčov. Prvok $k \in \mathcal{K}$ nazveme kľúč.
- Každý prvok $e \in \mathcal{K}$ jednoznačne určuje bijekciu z $\mathcal{M}$ do $\mathcal{C}$. Túto transformáciu budeme značiť E_e a budeme ju nazývať šifrovacou funkciou.
- Nech D_d je bijektívna transformácia z $\mathcal{C}$ do $\mathcal{M}$ pomocou prvku $d \in \mathcal{K}$, potom D_d nazveme dešifrovacou funkciou.
- Keď aplikujeme transformáciu E_e na správu $m \in \mathcal{M}$ budeme hovoriť, že šifrujeme správu m. Ak aplikujeme D_d na $c \in \mathcal{C}$ budeme hovoriť o dešifrovaní.
- Šifra alebo aj šifrovacia schéma sa skladá z množiny $\{E_e : e \in K\}$ a množiny $\{D_d : d \in K\}$, kde platí, že pre kazdé $e \in K$ existuje $d \in K$ také, že $D_d = E_e^{-1}$ a teda platí aj $D_d(E_e(m)) = m$ pre všetky $m \in \mathcal{M}$.

1.3 Symetrické šifrovanie

Definícia 1. *Nech šifrovacia schéma pozostáva z množín $\{E_e : e \in \mathcal{K}\}$ a $\{D_d : d \in \mathcal{K}\}$, kde $\mathcal{K}$ je množina všetkých kľúčov. Takúto schému nazveme symetrickou, ak pre každý pár (e, d) platí, že je "ľahké", vypočítať d pomocou e, a opačne.*

Najčastejšie používame $e = d$. Symetrické šifry sú zväčša veľmi rýchle, takže dokážu zašifrovať veľa dát za krátky čas a kľúče sú relatívne krátke. Ďalšou výhodou je možnosť zapúzdrenia, teda na jednu správu môže byť použitých viac šifier, vďaka čomu môžu dosahovať vačšiu mieru bezpečnosti. Pri komunikácii dvoch entít býva dobrým zvykom meniť kľúče relatívne často.

1.4 Asymetrické šifrovanie

Definícia 2. *Nech* $\{E_e : e \in \mathcal{K}\}$ *je množina šifrovacích funkcií a nech* $\{D_d : d \in \mathcal{K}\}$ *je množina príslušných dešifrovacích funkcií a* $\mathcal{K}$ *je množina všetkých kľúčov. Nech pre každý pár* (E_e, D_d) *platí, že je "výpočtovo nemožné"*[1] *získať správu* $m \in \mathcal{M}$ *pomocou* $c \in \mathcal{C}$ *a* E_e*, keď platí* $E_e(m) = c$*. Takéto šifrovanie nazveme asymetrické.*

Z definície zároveň vyplýva, že keď máme $e \in \mathcal{K}$, tak je nemožné získať príslušný kľúč d taký, aby platilo $D_d(E_e(m)) = m$. Aby bola táto vlastnosť zabezpečená, kryptografické funkcie sú založené na matematických problémoch akými sú napr. faktorizácia alebo výpočet diskrétneho logaritmu. Kľúč d budeme nazývať privátnym a e verejným kľúčom.

Pri využití asymetrickej kryptografie nám stačí uchovávať bezpečne len privátny kľúč a kľúče netreba meniť tak často ako pri symetrických šifrách. Nevýhodou oproti symetrickému šifrovaniu môže byť väčšia veľkosť kľúčov a nižšia rýchlosť šifrovania.

1.5 Hašovacie funkcie

Definícia 3. *Hašovacou funkciou* $\mathcal{H} : \mathcal{I} \to \mathcal{O}$ *nazveme funkciu zobrazujúcu z množiny vstupov ľubovoľnej dĺžky* $\mathcal{I}$ *na množinu reťazcov bitov pevnej dĺžky* $\mathcal{O}$*. Prvky množiny* $\mathcal{O}$ *budeme nazývať haše.*

Väčšina kryptografických schém využívajúcich hašovacie funkcie využíva takzvané kryptografické hašovacie funkcie. Kryptografická hašovacia funkcia obvykle spĺňa nasledujúce vlastnosti:

- Jednosmernosť: k danému hašu $o \in \mathcal{O}$ je "výpočtovo nemožné" získať akýkoľvek vstup $i \in \mathcal{I}$ spĺňajúci $\mathcal{H}(i) = o$.

[1] Pojem "výpočtovo nemožné"je postavený na predpoklade, že problém, ktorý riešime nie je efektívne riešiteľný, tzn. žiadny efektívny algoritmus nevie nájsť s nezanedbateľnou pravdepodobnosťou riešenie v potrebnom čase.

- Odolnosť voči kolíziám: je "výpočtovo nemožné" nájsť dva rôzne vstupy s rovnakým hašom $i_1 \neq i_2$, kde $i_1, i_2 \in \mathcal{I}$ pre ktoré platí, že $\mathcal{H}(i_1) = \mathcal{H}(i_2)$.

Kryptografické využitie nachádza v digitálnych podpisoch, konštrukciách autentizačných kódov, pri ukladaní hesiel alebo pri kontrole integrity údajov.

1.6 Použité šifry

V tejto časti predstavíme šifry, ktoré budeme používať v našom riešení.

AES

Veľmi rýchla bloková šifra založená na permutáciách a substitúciách, s veľkosťou bloku 128bitov. Dĺžka kľúča sa rôzni od 128 bitov cez 192 bitov až po 256 bitov. V našom riešení ju budeme používať v jej 128-bitovej forme na účely symetrickej kryptografie.

Podľa amerického úradu pre štandardizáciu(NIST), by šifra AES s dĺžkou kľúča 128 bitov mala byť bezpečná aspoň do roku 2030. Jej implementáciu je možné nájsť napríklad v knižnici SJCL [9], ktorú budeme používať. Viac informácií o nej môžeme nájsť v štandarde FIPS - 197 [11].

ElGamalova šifrovacia schéma

Algoritmus založený na probléme diskrétneho logaritmu slúžiaci na asymetrické šifrovanie. Plánujeme ho využívať pri zdieľaní súborov, aj pri šifrovaní hesiel k súborom. Budeme využívať krivku P-256, ktorá je štandardizovaná organizáciou NIST a jej predpokladaná bezpečnosť je aspoň do roku 2030. Jeho implementáciu tiež môžeme nájsť v SJCL [9].

1.7 Cloudové úložiská

Cloudové úložisko je služba, ktorá nám umožňuje manipulovať s diskovým priestorom prenajatým od poskytovateľa služby. Táto služba by mala byť škálovateľná, čo znamená, že vieme jednoducho zväčšiť priestor, za ktorý platíme. Ďalej by mala zabezpečovať prístupnosť dát, čo zahŕňa ochranu voči strate a poškodeniu dát.

Kapitola 2

Súčasné riešenia

V tejto kapitole opíšeme už existujúce riešenia podobné tomu nášmu. Rozdelíme si ich na také, ktoré využívajú vlastné cloudové riešenia, a také, ktoré používajú cloudové úložisko poskytované treťou stranou. Pozrieme sa na to, akú kryptografiu používajú, koľko si účtujú a ako vedia zdieľať dáta. V závere si popíšeme, čo bude ponúkať naša služba, v čom sa odlišuje od ostatných a v čom je lepšia.

2.1 Existujúce riešenia na šifrované ukladanie dát

Súčasné služby môžeme rozdeliť do dvoch kategórií. Jedny sa rozhodli používať vlastný cloud a ďalšie sa používajú na cloudy tretej strany. Väčšina z nich poskytuje natívnu aplikáciu do počítača a mobilného zariadenia. Tá menšia skupina sa zamerala na tvorbu webovej aplikácie, prostredníctvom ktorej užívateľ spravuje svoje dáta. Kryptografia prebieha na strane klienta pred prenosom dát do cloudu. Väčšina služieb sa snaží dodržiavať zásadu "zero-knowledge", ktorá zaručuje používateľovi, že poskytovateľ cloudového úložiska nebude mať o jeho dátach nijaké informácie.

Služby využívajúce vlastné cloudové riešenie

Jeden z najznámejších poskytovateľov šifrovaného cloudového úložiska je Mega [2]. Okrem webovskej aplikácie ponúka mobilné aj desktopové aplikácie, ktoré môžu niektorí používatelia preferovať pred webovým rozhraním. Pri ukladaní súborov Mega vygeneruje náhodný 128-bitový kľúč a následne dáta zašifruje šifrou AES-128. Všetky kľúče sú zašifrované pomocou univerzálneho hesla, ktoré sme si zvolili pri registrácii. To je zahašované kryptografickou hašovacou funkciou a uložené na serveroch. Celé šifrovanie prebieha na počítači klienta, takže Mega nemá žiadne informácie o obsahu uloženom v cloude a nepozná naše heslo. Pri zdieľaní súborov sa používa 2048 RSA kľúčový pár, pričom jeho privátna časť je zašifrovaná univerzálnym kľúčom. Služba ponúka 50 GB zdarma a za 500 GB používateľ zaplatí mesačne 9,99$. Keď sa rozhodneme zdieľať nejaký súbor, Mega nám poskytne webovú adresu a kľúč, ktorým je súbor zašifrovaný. Žiaľ distribúciu webovej adresy a hlavne hesla už nechá na nás.

Služby SpiderOak [6] a Wuala [5] tiež využívajú vlastný cloud, ale neposkytujú webové rozhranie a všetky operácie musíme robiť pomocou aplikácie na našom počítači. SpiderOak pracuje na podobných princípoch ako Mega ale namiesto AES-128 používa AES-256. Zadarmo sú dostupné 2 GB úložného priestoru a za 12$ mesačne je možné ho rozšíriť na 1 TB. SpiderOak ponúka takzvané zdieľacie miestnosti, ktoré sú opäť definované nejakou adresou a heslom. Každý, kto sa v tejto miestnosti nachádza má prístup ku kľúčom, ktoré šifrujú súbory dostupné v miestnosti.

Wuala využíva systém Cryptree [7], v ktorom používa AES-256 na šifrovanie, RSA 2048 na podpisy a zdieľanie dát a SHA-256 na zabezpečenie integrity. Cena sa pohybuje od 0,99 € za 5 GB do 159,90 € za 2 TB. Zdieľanie súborov v aplikácii Wuala funguje podobne ako pri službe Mega. Najprv si od aplikácie vypýtame link a následne ho pošleme aj s heslom nášmu adresátovi.

Všetky tri služby sú kompatibilné s operačnými systémami Windows, Mac, Linux, Android a iOS.

Využívanie dostupných cloudových riešení

Viivo [3] na rozdiel od služby Mega využíva už existujúce cloudové riešenia, ktoré poskytujú API na prácu so súbormi. Vytvoril tak vrstvu medzi klientom a jeho obľúbenými cloudovými úložiskami Drive, Dropbox, Box alebo SkyDrive. Viivo vytvorí 2048 bitový kľúčový pár, ktorý sa používa pri zdieľaní dát. Kľúč je zabezpečený pomocou hesla, ktoré si používateľ zvolí pri registrácii. Pre súkromné využitie je zadarmo, firmy si priplatia od 4,99$ do 9,99$ mesačne. Pri zdieľaní súborov pomocou Viiva používateľ nahrá svoje súbory do Dropboxu pomocou aplickácie Viivo a následne musí na Dropbox stránke prizvať ďalšieho používateľa k zdieľanej zložke. Následne adresát musí prijať pozvanie, či už v aplikácii alebo na webe. Ako posledný krok zdieľajúci musí potvrdiť zdieľanie v natívnej aplikácii Viivo. Na pozadí prebehne kryptografia, ktorá umožní dešifrovať súbory druhej strane.

Boxcryptor [4] je ďalšia služba podobná Viivu, ktorá vytvorila vrstvu medzi cloudom a používateľom. Podporuje rovnaké cloudy ako Viivo a naviac ešte SugarSync. Kryptografia funguje na rovnakých princípoch ako Mega, ale využíva silnejšie kľúče, t.j. AES-256 pri symetrickej a RSA s kľúčom dĺžky 4096 bitov pri asymetrickej kryptografii. Základné šifrovanie je poskytované zadarmo a neobmedzený firemný účet stojí 96$ ročne. Aby sme mohli zdieľať stačí, aby bol adresát zaregistrovaný na www.boxcryptor.com a mal aplikáciu BoxCryptor. Následne treba vybrať súbory, ktoré chceme zdieľať a v aplikácii zadať jeho e-mail. Na výmenu kľúčov sa používa server, ktorý ukladá všetky kľúče. Podrobnejšie informácie o výmene kľúčov môžeme nájsť na ich stránke [4].

Viivo ani Boxcryptor neposkytujú webové rozhranie, takže používateľ je nútený inštalovať dodatočný softvér, ktorý je kompatibilný s Windowsom, Macom ako aj s iOS a Androidom.

2.2 Naše riešenie

Cieľom našej práce bude implementovať službu, ktorá umožní používateľom bezpečne a jednoducho ukladať, sťahovať a zdieľať súbory v cloude. Rozhodli sme sa využívať cloudové úložisko poskytované treťou stranou, konkrétne GoogleDrive. Všetko budeme ovládať cez webové rozhranie, ktoré má rôzne výhody, t.j. netreba inštalovať žiadny softvér okrem webového prehliadača, ktorý je štandardne predinštalovaný, čím vieme pomerne jednoducho zabezpečiť kompatibilitu aplikácie od mobilu až po desktop a nemusíme sa starať o to, aký operačný systém náš používateľ uprednostňuje. Riešenie sa pokúsime implementovať tak, aby bolo jednoduché ho rozšíriť o iné cloudové úložiská ako napríklad Dropbox či SkyDrive. Našu službu sme sa rozhodli nazvať SecureCloud.

2.3 Porovnanie

V tejto časti vysvetlíme, v čom sa bude naše riešenie líšiť od ostatných služieb a aká je naša motivácia vytvoriť vlastné riešenie.

Mega vs SecureCloud

Mega patrí medzi najpopulárnejších poskytovateľov šifrovaných cloudových riešení na trhu. Okrem toho, že ponúka webovské rozhranie dáva nám aj dostatok priestoru zdarma. No zmena cloudového úložiska implikuje problémy ako nedostupnosť starých súborov na novom úložisku, prípadne potreba zorientovať sa v novom používateľskom prostredí. Medzi najpoužívanejšie a najznámejšie cloudové úložiská rozhodne patrí GoogleDrive a Dropbox, ale ani jedno neponúka šifrovanie dát. Výhoda nášho riešenia oproti službe Mega spočíva v možnosti pokračovať vo využívaní služieb Googlu alebo Dropboxu a zároveň v zabezpečení šifrovania dát.

Zdieľanie v našom riešení naviac nevyžaduje nič iné ako registráciu používateľa, s ktorým chceme zdieľať, a pár klikov myškou. Zároveň pri zdieľaní

zabezpečíme bezpečnú distribúciu kľúču k druhej strane a nedáme jej možnosť kompromitovať kľúč.

Viivo a Boxcryptor vs SecureCloud

Napriek tomu, že Viivo aj Boxcryptor ponúkajú využívanie vrstvy medzi obľúbenými poskytovateľmi úložísk a používateľom, nemajú nijaké webové rozhranie, čo zaťažuje používateľa okrem registrácie aj inštaláciami mobilných a desktopových aplikácií na všetkých zariadeniach, na ktorých chcú službu využívať. Naopak naše riešenie vyžaduje iba prihlásenie pomocou už existujúceho Google účtu. Oproti Viivo máme aj veľkú výhodu v jednoduchšom zdieľaní súborov.

Kapitola 3

Návrh funkcionality

V tejto kapitole opíšeme, ako je naše riešenie navrhnuté. Vysvetlíme, ako budeme postupovať v prípade nahrávania, sťahovania, zdieľania a čo spravíme, keď už nechceme zdieľať dáta. Na záver kapitoly uvedieme, aký môže mať toto riešenie problém a ako by sme ho vedeli riešiť.

3.1 Prerekvizity

Aby naše riešenie fungovalo, budeme potrebovať server, ktorý bude poskytovať API na komunikáciu s databázou a bude hostovať naše webové rozhranie, databázu, kam budeme ukladať používateľove kľúče a kľúče k súborom, a cloudové úložisko, ktoré bude zabezpečovať prácu so súbormi. Cloud musí poskytovať rozhranie, pomocou ktorého vieme nahrávať a sťahovať dáta a musí byť schopný autorizovať rôznych používateľov pre prístup k dátam iných používateľov. Vďaka tomu budeme môcť zdieľať súbory.

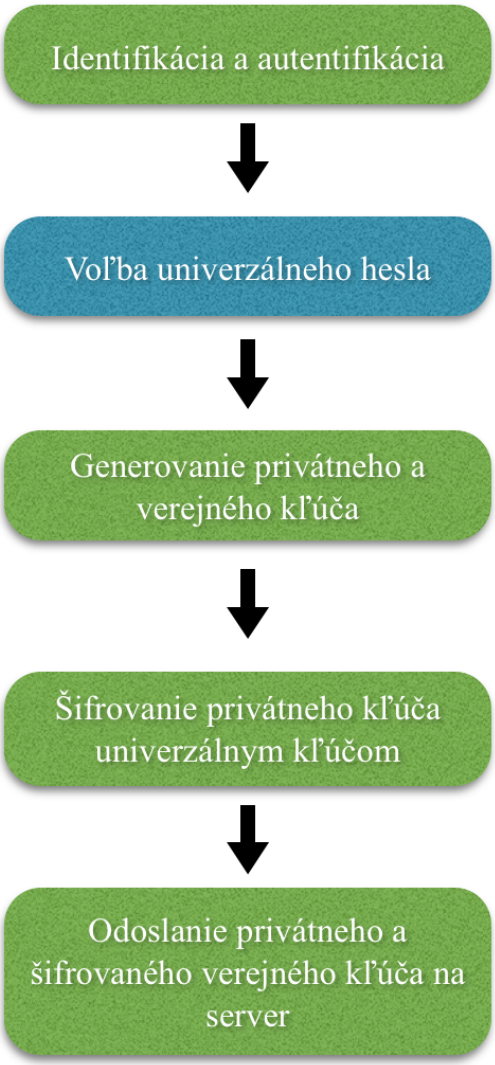
3.2 Registrácia

Keď sa používateľ rozhodne používať našu službu, musí sa nejakým spôsobom identifikovať a autentifikovať. Identifikácia vyžaduje, aby si každý používateľ zvolil meno a aby sme ho mohli autentifikovať potrebuje heslo. Druhou

možnosťou je využiť tretiu službu ako napríklad cloud, kde budeme ukladať súbory. Toto je možné napríklad pomocou protokolov oAuth alebo openID. Nech už vyberieme jeden alebo druhý spôsob, budeme jednoznačne vedieť s kým komunikujeme. S touto informáciou vyzveme používateľa, aby si zvolil svoje univerzálne heslo, pomocou ktorého bude autorizovať akcie ako nahrávanie, sťahovanie, zdieľanie a zrušenie zdieľania dát.

Následne v používateľovom prehliadači vygenerujeme verejný a privátny kľúč, ktorý symetricky zašifrujeme pomocou už zvoleného univerzálneho hesla. Verejný aj zašifrovaný privátny kľúč uložíme na serveri, kde budú mať všetci používatelia prístup k verejným kľúčom, no k privátnym len jeho vlastníci. Vďaka tomu, že privátny kľúč je zašifrovaný, server nemá žiadnu informáciu o privátnom kľúči a teda schéma ostane bezpečná.

Ďalšou možnosťou bolo uložiť privátny kľúč na užívateľovom počítači. Toto sme sa rozhodli nevyužiť z dôvodu náročného manažmentu kľúčov na rôznych zariadeniach. Napríklad pri strate zariadenia by to vyžadovalo zmenu privátneho kľúča, čo by implikovalo nutnosť prešifrovania všetkých súborových kľúčov. Distribúcia kľúča do rôznych zariadení by bola v rukách používateľa, čo by mohlo byť nepohodlné.

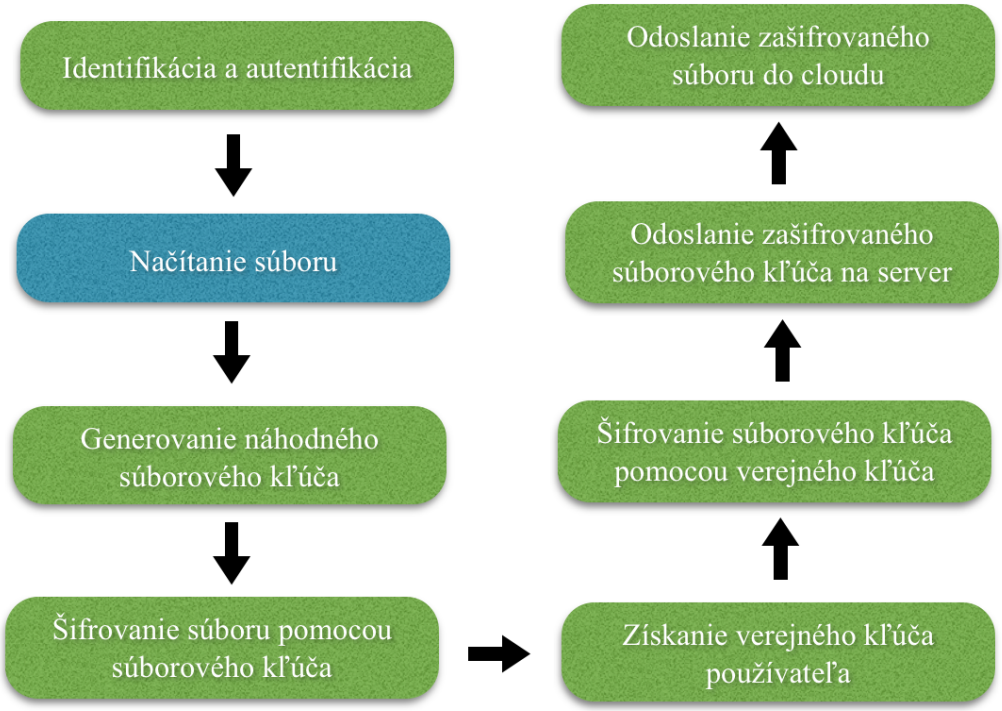


Obr. 3.1: Navrhovaný priebeh registrácie v systéme SecureCloud

3.3 Nahrávanie dát

Používateľ sa autentifikuje a v prehliadači sa pomocou pseudonáhodného generátora vygeneruje náhodný kľúč. Ten využije na zašifrovanie súboru a

vypýta si od servera svoj verejný kľúč. Pomocou verejného kľúča zašifruje heslo k súboru a v takomto tvare ho už môže poslať na server. Keďže heslo je zašifrované a server nemá žiadnu informáciu o privátnom kľúči, nevie získať dáta v otvorenom tvare. Následne už len stačí zašifrovaný súbor nahrať na cloudové úložisko.

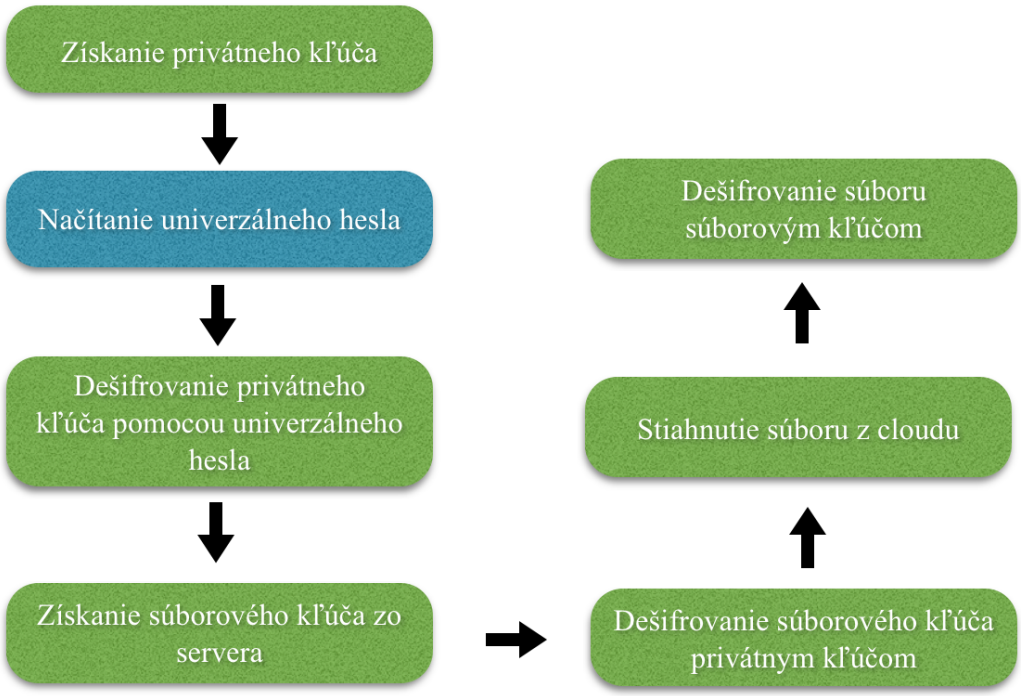


Obr. 3.2: Nahrávanie dát

3.4 Sťahovanie dát

V prípade, že privátny kľúč je uložený na serveri, vypýtame si ho a požiadame používateľa o jeho univerzálne heslo, aby sme mohli privátny kľúč dešifrovať. S privátnym kľúčom môžeme následne dešifrovať heslo k súboru, ktoré si

opäť vypýtame od servera. V tejto chvíli máme k dispozícii kľúč k súboru a teda nám stačí stiahnuť súbor z cloudu a rozšifrovať ho pomocou zmieneného kľúča.

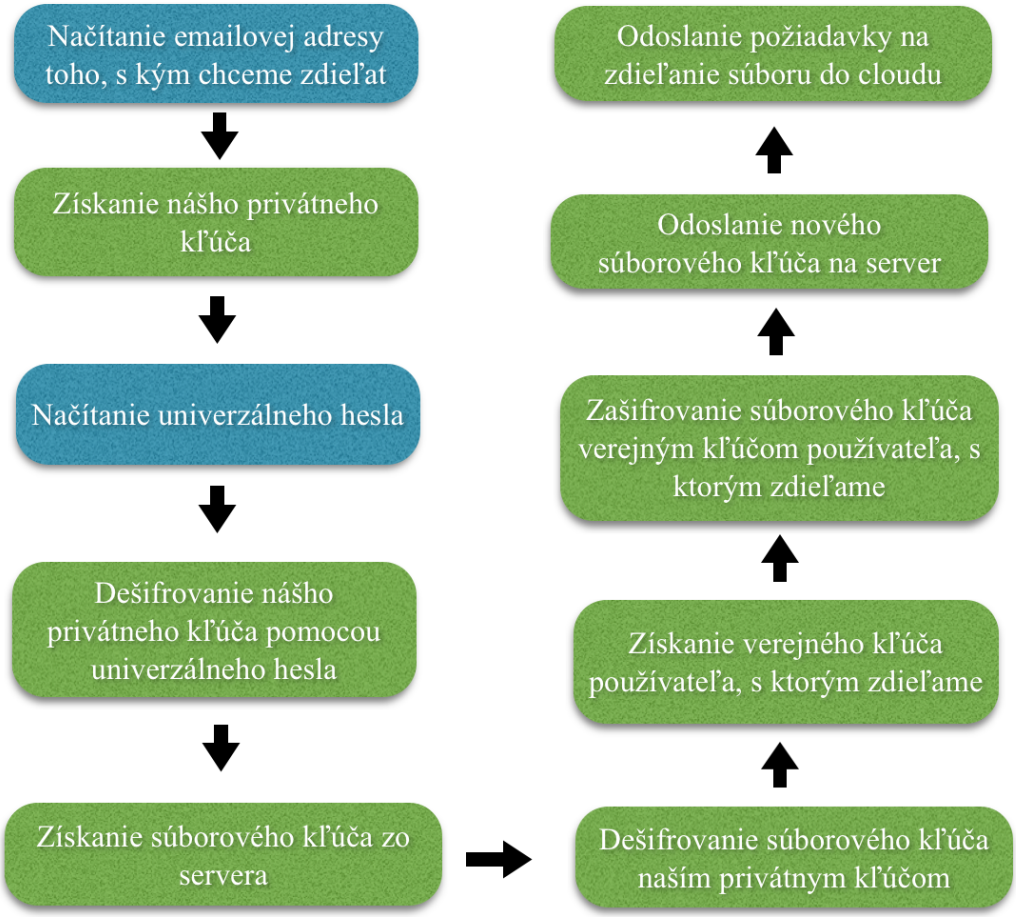


Obr. 3.3: Sťahovanie dát

3.5 Zdieľanie

Chceme zdieľať súbor, ktorý je uložený v zašifrovanej forme na cloude. Na zdieľanie je nutné, aby cieľový príjmateľ bol zaregistrovaný na našom serveri a mal vygenerovaný privátny a verejný kľúč. Zdieľanie bude prebiehať tak, že používateľ si od servera vypýta privátny kľúč, ktorý dešifruje pomocou svojho univerzálneho kľúča. Následne požiada server o zašifrovaný kľúč k súboru,

ktorý dešifruje pomocou privátneho kľúča. Dešifrovaný kľúč k súboru potom zašifruje verejným kľúčom používateľa, s ktorým chce dáta zdieľať a takýto kľúč k súboru uloží na serveri. Následne pošle požiadavku na cloud, aby povolil prístup k súboru nášmu adresátovi. Ten má potom prístup ku kľúču na dešifrovanie súboru a tiež si môže stiahnuť zašifrovaný súbor z cloudu.



Obr. 3.4: Zdieľanie dát

3.6 Zrušenie zdieľania

Keď sa rozhodneme zrušiť zdieľanie súboru, stačí, aby sme požiadali cloudové úložisko o revokovanie prístupu k súborom. Keby sa používateľ, s ktorým ten súbor zdieľame, rozhodol zverejniť ho alebo inak distribuovať, šifrovanie nám nepomôže, pretože už si mohol spraviť kópiu nešifrovanej verzie. Preto stačí revokovať prístup na cloude a nemusíme súbor prešifrovať. V prípade, že sa súbor zmení a budeme ho znova uploadovať, prebehne opäť procedúra ako pri nahrávaní dát. Nové informácie teda nebudú kompromitované.

3.7 Bezpečnostný problém

V prípade, že by sa poskytovateľ cloudového úložiska a server rozhodli ukradnúť používateľove dáta, nastane problém. Predstavme si teda nasledujúcu situáciu. Používateľ chce zdieľať súbor, poprosí server o súborový kľúč, ktorý dešifruje. Vypýta si od serveru verejný kľúč používateľa, s ktorým chce zdieľať, zašifruje ním súborový kľúč a pošle ho na server. V tejto chvíli sa mohlo stať, že server odoslal verejný kľúč, ku ktorému vlastní privátny kľúč. Ak sa mu podarilo dohodnúť s poskytovateľom cloudového úložiska, tak v tejto chvíli vedia dešifrovať súbor. Cloud poskytne súbor a server poskytne kľúč k dešifrovaniu.

Riešenie

Ako čiastočné riešenie problému by sme mohli navrhnúť podpisovanie verejného kľúča. Čiastočným preto, lebo aj tu musíme byť schopní overiť, komu patrí verejný kľúč. Týmto by sme do problému zahrnuli ďalšiu stranu, ktorou by mohla byť certifikačná autorita, čo by mohlo znížiť pravdepodobnosť hrozby. No v našom riešení toto vynecháme.

Kapitola 4

Implementácia

V tejto kapitole sa pozrieme, s akými technológiami sme sa rozhodli pracovať a popíšeme najhlavnejšie časti zdrojového kódu. Ukážeme, čo spravíme pri registrácii nového používateľa, ďalej čo pri nahrávaní súborov a na záver predvedieme, ako funguje sťahovanie a zdieľanie.

4.1 Použité technológie

Zhrnieme si, aké technológie naša implementácia využíva na strane klienta, aké na strane servera a v krátkosti si ich popíšeme.

4.1.1 Back-end

Pod pojmom back-end máme na mysli implementáciu riešenia mimo používateľovho počítača. Na implementáciu serverovej časti sme sa rozhodli použiť Node.js s frameworkom Express, databázový systém MongoDB s pomocou objektového modelovacieho nástroju mongoose.

Node.js

Je platforma postavená na googlovskom V8 javascript engine, ktorý je napísaný v C++. Písať v jednom jazyku front-end aj back-end už dnes vďaka Node.js nie je problém. Celý model tejto platformy je postavený na udalostiach a neblokujúcich operáciách, takže je vhodný pre real-timeové aplikácie.

Express

Express je webový framework postavený na Node.js, ktorý nám zjednodušuje implementáciu backendu. S pomocou Expressu vieme veľmi ľahko vytvoriť REST API. Napísať čo sa stane, keď používateľ pristúpi pomocou metódy GET k webovej adrese `http://mojserver.sk/pes`, by mohlo vyzerať nasledovne:

```
1        app.get('/pes', function (req, res) {
2          // nieco sa stane so psom, napriklad sa zobrazi na
               stranke
3        });
```

Zdrojový kód 4.1: Express routing

MongoDB

MongoDB sme sa rozhodli použiť kvôli dynamickej schéme, škálovateľnosti, ale aj kvôli možnosti pracovať s moongose. Moongose je nástroj, ktorý nám dáva možnosť pracovať s databázou ako s obyčajnými objektami, čo výrazne urýchľuje vývoj. Pre ilustráciu, vytvorenie nového dokumentu v databáze bude vyzerať nasledovne:

```
1        var Cat = mongoose.model('Cat', { name: String });
2
3        var kitty = new Cat({ name: 'Murko' });
4        kitty.save(function (err) {
5          if (err) // napriklad chyba v databaze
6            console.log('mnau');
```

```
7         });
```

Zdrojový kód 4.2: Vytvorenie mongoose modelu a jeho použitie

4.1.2 Front-end

Za front-end budeme považovať všetky programy a scripty, ktoré sa dostanú k užívateľovi a budú zobrazované alebo vykonávané v jeho prehliadači.

jQuery

Väčšina moderných prehliadačov vnútorne používa nejakú reprezentáciu na vykreslenie HTML, ktorú budeme nazývať objektový model dokumentu, tzv. DOM, z anglického Document Object Model. Pre prácu s DOM, používame knižnicu jQuery. Jej dokumentácia a developerská podpora je veľmi rozsiahla, čo uľahčuje jej používanie.

Knižnicu jQuery budeme tiež využívať na AJAX volania, pre ktoré nám poskytuje jednoduché API. AJAX slúži na výmenu dát medzi serverom a klientom bez nutnosti obnovovať stránku. Napríklad, keď si chceme vypýtať od servera privátny kľúč, pošleme požiadavku GET na adresu `http://server.com/getPrivKey` a dostaneme od neho odpoveď s privátnym kľúčom používateľa.

SJCL

Kryptografiu nám bude zaobstarávať knižnica Stanford Javascript Crypto Library, ktorej zdrojový kód a dokumentácia je online [9]. Táto open-source knižnica ponúka jednoduché rozhranie, pomocou ktorého vieme šifrovať a dešifrovať dáta. Podporuje symetrickú, aj asymetrickú kryptografiu, rôzne hašovacie funkcie a tiež umožňuje vytvárať a overovať podpisy. Okrem toho, že nám poskytuje všetky kryptografické primitíva, ktoré naše riešenie vyžaduje, jej hlavnou výhodou je efektívna implementácia. V rýchlosti šifrovania je v priemere štyrikrát rýchlejšia ako iné [8]. Kompatibilita medzi všetkými

modernými prehliadačmi ako Chrome, Firefox, Safari a Internet Explorer v kombinácii s rýchlosťou a jednoduchým používateľským rozhraním boli kritickými prvkami pri výbere.

4.2 Implementáciu návrhu

Túto časť venujeme prevažne zdrojovému kódu nášho riešenia. Uvedieme a vysvetlíme zdrojový kód registrácie, nahrávania a sťahovania dát a nakoniec sa pozrieme, ako funguje zdieľanie. Niektoré časti kódu pre jednoduchosť vynecháme a pokúsime sa uviesť iba tie najdôležitejšie časti.

4.2.1 Registrácia

Aby sme vedeli s kým komunikujeme, potrebujeme najprv overiť jeho identitu. Na to sme sa rozhodli použiť protokol oAuth. S jeho pomocou dostaneme informácie o tom, s kým komunikujeme a zároveň používateľa autorizujeme k používaniu jeho GoogleDrive účtu. Následne potrebujeme vygenerovať privátny a verejný kľúč, vypýtať si od používateľa jeho univerzálne heslo a zašifrovať privátny kľúč univerzálnym heslom. Potom ho už len stačí poslať na server, kde sa uloží.

```
1  function generateKeyPair(callback) {
2      var keys = sjcl.ecc.elGamal.generateKeys(256);
3
4      //SERIALIZING KEYPAIR
5      var pub = keys.pub.get();
6      var sec = keys.sec.get();
7      var serializedKeyPair = {
8          pub: sjcl.codec.base64.fromBits(pub.x.concat(pub.y)),
9          priv: sjcl.codec.base64.fromBits(sec)
10     };
11
12     // ASKING USER FOR HIS PASSWORD
```

```
13      var passwd = prompt('Enter your new universal password.
            You will need this password to decrypt and share files
            ', 'Password here');
14      // ENCRYPTING USERS PRIVATE KEY
15      var enc = sjcl.encrypt(passwd, serializedKeyPair.priv);
16      serializedKeyPair.priv = enc;
17
18      // SENDING KEYPAIR TO SERVER
19      saveKeyPair(serializedKeyPair, callback);
20  }
21
22  function saveKeyPair(keyPair, callback) {
23      // SEND KEYPAIR TO SERVER
24      $.post('/saveKeypair', keyPair, function (res2) {
25          callback(res2);
26      });
27  }
```

Zdrojový kód 4.3: Generovanie kľúčového páru a odoslanie na server

4.2.2 Nahrávanie dát

Keď chceme uložiť ľubovolné dáta v cloude v šifrovanej forme, najprv potrebujeme načítať súbor.

```
1  function uploadFile() {
2      if ($('input#uplFile')[0].files[0] != undefined){
3          updateFileResumableGoogle({'title': $('input#uplFile'
               )[0].files[0].name + '.sc'}, $('input#uplFile')
               [0].files[0], function (uplResp) {// possible to
               add callback});
4      }
5  }
```

Zdrojový kód 4.4: Načítanie súboru

Po načítaní celého súboru vygenerujeme náhodný kľúč, ktorý použijeme na zašifrovanie dát súboru. Zo servera si vypýtame náš verejný kľúč a ním

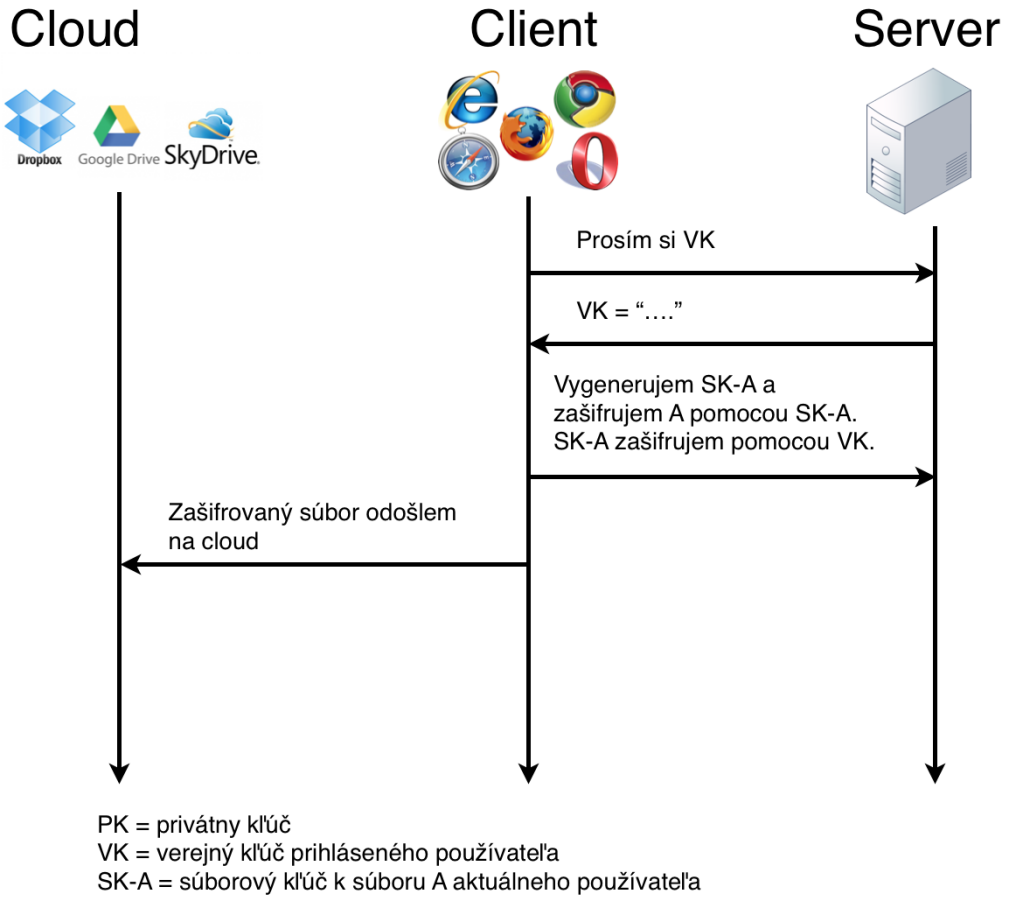
zašifrujeme kľúč použitý na šifrovanie súboru. Nakoniec odošleme súbor do cloudu a zašifrované súborové heslo na server.

Väčšina našich funkcií používa tzv. callback, čo je funkcia, ktorá sa zavolá po skončení predchádzajúcej funkcie. Napríklad, keď zavoláme getPubKey(email, callback), najprv sa vykoná funkcia getPubKey, a keď skončí svoju prácu, zavolá funkciu callback(pubKey). Ako parameter je vždy návratová hodnota predchádzajúcej funkcie.

```
6  function updateFileResumableGoogle(metadata, fileData,
       callback) {
7      var accessToken = gapi.auth.getToken().access_token;
8      var email = $.cookie('email');
9
10     //GET PUBLIC KEY OF LOGGED USER
11     getPubKey(email, function(pubKey) {
12
13         //GENERATE PASSWORD FOR FILE ENCRYPTION
14         createRandomString(function (passString) {
15
16             // FILE UPLOAD
17             var reader = new FileReader();
18             reader.readAsArrayBuffer(fileData);
19             reader.onload = function() {
20
21                 //FILE ENCRYPTION
22                 var bits = sjcl.codec.bytes.toBits(new
                       Uint8Array(reader.result));
23                 var crypt = sjcl.encrypt(passString, bits);
24                 var blob = new Blob([crypt]);
25
26                 //ACTUAL UPLOADING
27                 var uploader = new MediaUploader({
28                     metadata: metadata,
29                     file: blob,
30                     token: accessToken,
31                     onComplete: function(gResp) {
32
```

```
33                          //FILE PASSWORD ENCRYPTION
34                          var encPass = sjcl.encrypt(pubKey,
                                passString);
35                          gResp = JSON.parse(gResp);
36
37                          //SENDING ENCRYPTED FILE PASSWORD TO
                                SERVER
38                          saveFileKey(email, encPass, gResp.id,
                                callback);
39                      },
40                      onError: function(err) {
41                          log(err);
42                      }
43                  });
44                  uploader.upload();
45              };
46          });
47      });
48  }
```

Zdrojový kód 4.5: Upload súboru



Obr. 4.1: Komunikácia pri nahrávaní súboru na server

4.2.3 Sťahovanie dát

V prípade sťahovania súborov si klikneme na súbor vo webovom rozhraní a potom na tlačítko "Download", ktoré vykoná niekoľko akcií. V prvom rade získame zo servera privátny kľúč a rozšifrujeme ho naším univerzálnym heslom.

```
1  function getPrivKey(callback) {
2      $.get('/getPrivKey', function(res) {
3          // DECRYPTS PRIVATE KEY WITH USERS KEY
4          var userKey = prompt('Please enter your password', '
               Enter your password here');
5          var privKey = sjcl.decrypt(userKey, res);
6
7          // DESERIALIZING PRIVATE KEY
8          var sec = new sjcl.ecc.elGamal.secretKey(
9              sjcl.ecc.curves.c256,
10             sjcl.ecc.curves.c256.field.fromBits(sjcl.codec.
                   base64.toBits(privKey))
11         );
12
13         callback(sec);
14     });
15 }
```

Zdrojový kód 4.6: Získanie privátneho kľúča

Keď ho už máme k dispozícii, požiadame server o zašifrovaný kľúč k súboru a rozšifrujeme ho pomocou privátneho kľúča.

```
16 function getFileKey(fileId, callback) {
17     getPrivKey(function (privKey) {
18         $.post('/getFileKey', {'fileId': fileId}, function(
               encFileKey) {
19
20             // DECRYPTS FILE KEY WITH USERS PRIVATE KEY
21             var fileKey = sjcl.decrypt(privKey, encFileKey);
22
23             callback(fileKey);
24         });
25     })
26 }
```

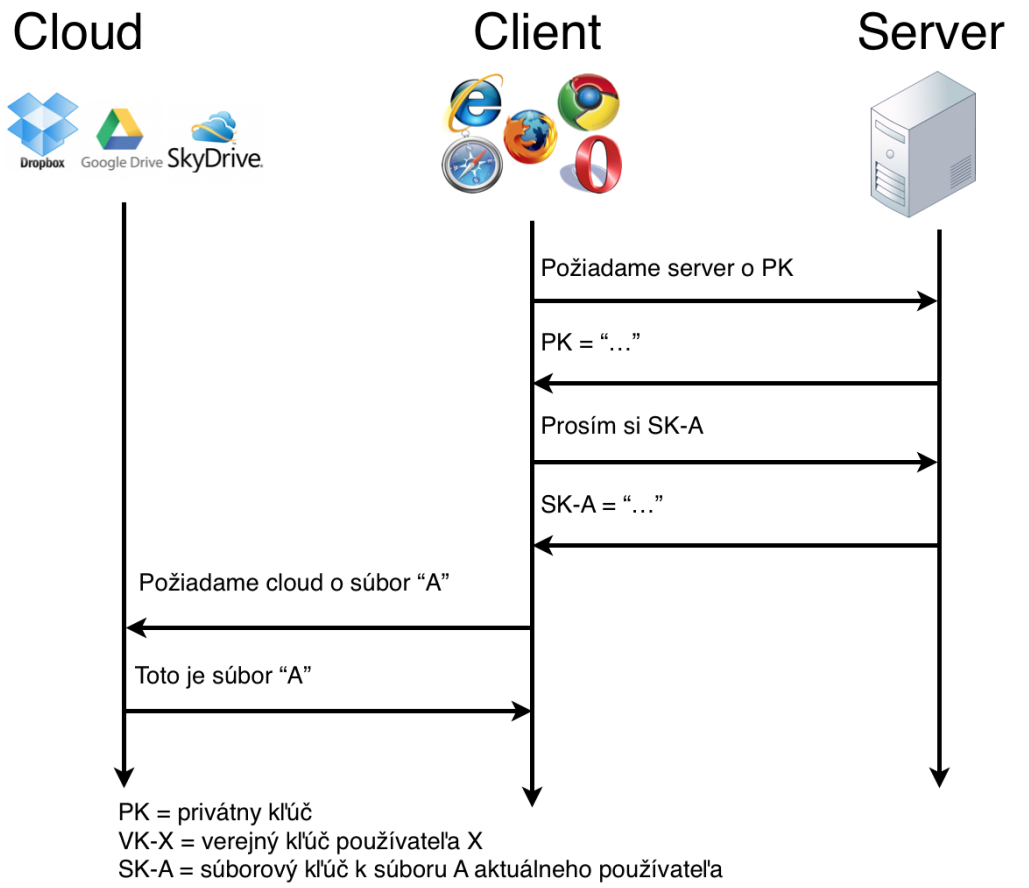
Zdrojový kód 4.7: Získanie súborového kľúča

Teraz už môžme prejsť k samotnému stiahnutiu súboru a jeho rozšifrovaniu.

```
27 function downloadFileGoogle(file, callback) {
28     if (file.downloadUrl) {
29         var accessToken = gapi.auth.getToken().access_token;
30         var xhr = new XMLHttpRequest();
31         xhr.open('GET', file.downloadUrl);
32         xhr.setRequestHeader('Authorization', 'Bearer ' +
               accessToken);
33 
34         // ITS ENCRYPTED FILE
35         xhr.onload = function() {
36             getFileKey(file.id, function(fileKey) {
37 
38                 // DECRYPT FILE
39                 var base64decrypt = sjcl.decrypt(fileKey, xhr
                       .response, {'raw': 1});
40                 var byteArray = new Uint8Array(sjcl.codec.
                       bytes.fromBits(base64decrypt));
41 
42                 // PASS DOWNLOADED FILE TO CALLBACK
43                 callback({
44                     'title': title,
45                     'data': byteArray
46                 });
47             });
48         };
49         xhr.onerror = function() {
50             console.log("ERROR!");
51         };
52         xhr.send();
53     } else {
54         console.log("ERROR MISSING DOWNLOAD URL");
55     }
56 }
```

Zdrojový kód 4.8: Stiahnutie súboru

Keď všetko prebehne úspešne zavoláme callback funkciu, ktorej parametre budú meno súboru a jeho dáta.



Obr. 4.2: Komunikácia pri sťahovaní súboru na server

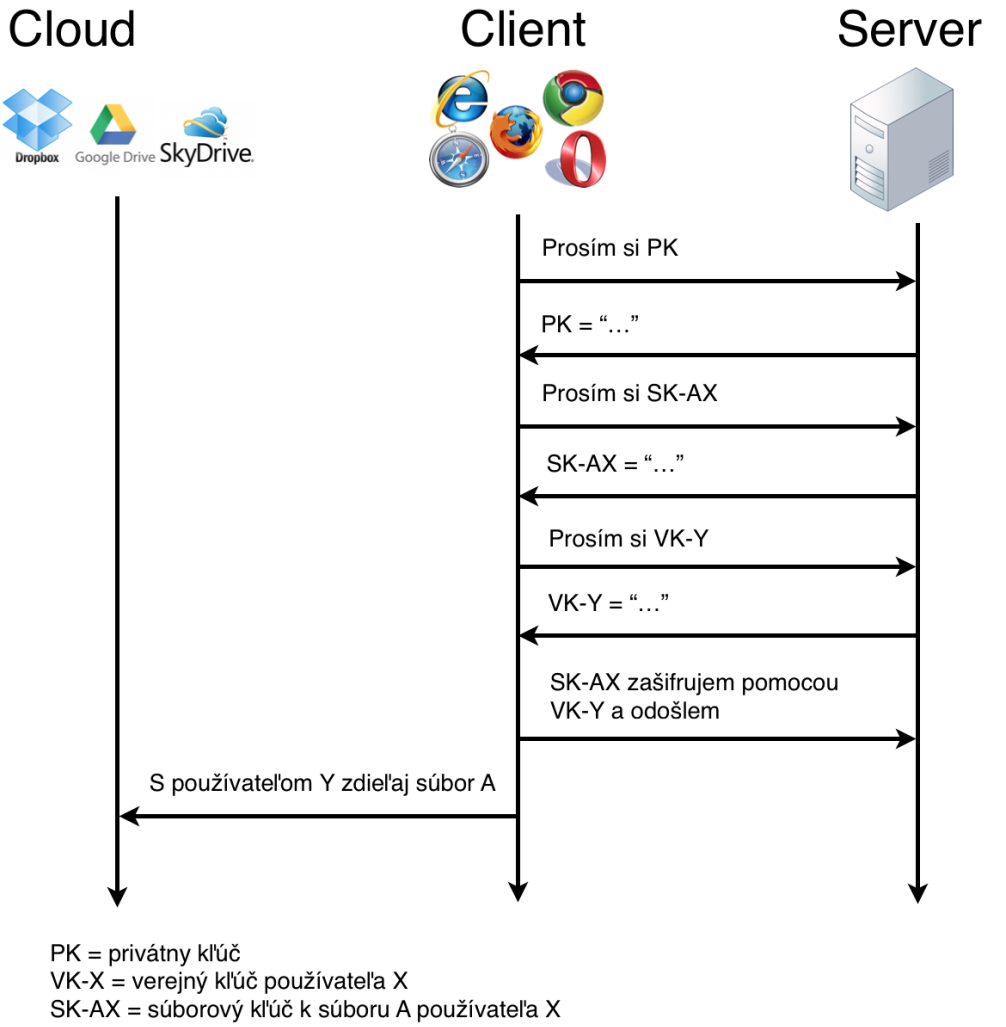
4.2.4 Zdieľanie dát

To, čo robí naše riešenie zaujímavým, je zdieľanie súborov. Pre zdieľanie súboru nám stačí zavolať funkciu, ktorej dáme štyri parametre. ID súboru, email používateľa, s ktorým chceme zdieľať, rolu a funkciu, ktorá sa má po skončení funkcie vykonať. Parameter role nám hovorí, aké právomoci má používateľ k tomu súboru. Môže byť čitateľ, čo mu umožní súbor iba čítať,

zapisovateľ, ktorý môže meniť dáta súboru, alebo vlastník, čo znamená, že môže robiť všetko ako čitateľ a zapisovateľ, ale naviac môže súbor aj zmazať. Celý proces zdieľania vyžaduje niekoľko krokov. V prvom rade musíme získať kľúč k súboru rovnako ako pri sťahovaní dát. Potom potrebujeme verejný kľúč používateľa, s ktorým plánujeme zdieľať súbor. Následne zašifrujeme súborový kľúč verejným kľúčom, ktorý sme získali a uložíme ho na server. A napokon už len stačí požiadať cloud, aby povolil prístup k súboru používateľovi, s ktorým sme sa rozhodli podeliť o naše dáta.

```
1  function shareFileGoogle(fileId, email, role, callback) {
2
3      getFileKey(fileId, function(fileKey) {
4          getPubKey(email, function (pubKey) {
5
6              //ENCRYPT FILEKEY WITH PUB-KEY OF USER I AM
                   SHARING WITH
7              var shareFileKey = sjcl.encrypt(pubKey, fileKey);
8
9              //SAVE NEW ENCRYPTED KEY
10             saveFileKey(email, shareFileKey, fileId, function
                   (saveFileKeyResponse) {
11
12                 //SHARE ON GOOGLE DRIVE (ADD PERMISSION FOR
                       USER)
13                 insertPermissionGoogle(fileId, email, 'user',
                       role, function(resp) {
14                     callback(resp);
15                 });
16             });
17         });
18     });
19 }
```

Zdrojový kód 4.9: Zdieľanie súboru



Obr. 4.3: Komunikácia pri zdieľaní súboru na server

4.2.5 Zrušenie zdieľania

Aby sme naše dáta prestali zdieľať, stačí požiadať cloud o zrušenie prístupu k súboru a aby sme zbytočne nezaplňovali databázu vymažeme aj súborový

kľúč zo servera.

```
function stopShareingFileGoogle(fileId, email, callback) {
    retrievePermissions(fileId, function(permissions) {
        for(var permission in permissions) {
            if (permissions[permission].emailAddress == email
               ) {
                var permId = permissions[permission].id;
                removePermissionGoogle(fileId, permId,
                   function(resp) {
                    callback(resp);
                });
            }
        }
    });
}
```

Zdrojový kód 4.10: Zrušenie zdieľania súboru

Záver

Práca sa zaoberala implementáciou webovského rozhrania, ktoré poskytuje šifrovanie dát v cloude. V úvode sme sa venovali kryptografii, popísali sme jej základné pojmy a typy šifrovania. Ďalej sme predstavili zoznam existujúcich riešení pre bezpečné zdieľanie dát. Prvú skupinu riešení tvoria služby využívajúce vlastné cloudové riešenia, napr. úložisko Mega, do druhej skupiny sme zaradili služby využívajúce už dostupné cloudy (Google Drive, DropBox a iné). Nevýhodou prvej skupiny je nutnosť prejsť na neznáme cloudové prostredie a nevýhodou druhej je potreba inštalácie dodatočného softvéra.

My sme zvolili riešenie, ktoré tieto dva nedostatky eliminuje a kombinuje výhody oboch prístupov. Vytvorili sme službu SecureCloud, ktorá pre použitie vyžaduje iba moderný prehliadač, internetové pripojenie a účet populárneho cloudu Google Drive. Pre integráciu s inými cloudovými úložiskami by stačilo implementovať časti, ktoré pracujú s novým cloudom a napojiť ich na naše webové rozhranie.

V tretej kapitole sme navrhli spôsob fungovania SecureCloudu. Popísali sme jeho funkcionalitu, ktorej prednosťou je, že nevyžaduje zdieľanie informácií v otvorenom tvare so serverom ani cloudom. V službe Mega musí používateľ sprostredkovať svojmu adresátovi heslo na odšifrovanie súborov, kým v našom riešení je potrebné len zadať e-mail adresáta a zvoliť súbor na zdieľanie. To umožňuje vyhnúť sa distribúcii hesla po potenciálne nezabezpečených kanáloch.

V poslednej kapitole sme podrobne popísali implementáciu nášho webovského rozhrania, uviedli sme použité technológie, časti zdrojového kódu

SecureCloudu a vysvetlili sme ich fungovanie.

Na prácu je možné nadviazať implementáciou ďalších funkcionalít, ako je napríklad šifrovaná kolaborácia v reálnom čase alebo podpora pre zdieľané tajomstvo. Ďalším dôležitým aspektom je používateľská prívetivosť, na ktorú sme sa v tejto práci nesústredili.

Literatúra

[1] Alfred J. Menezes - Paul C. van Oorschot - Scott A. Vanstone, 1996, Handbook of Applied Cryptography, CRC Press

[2] Mega, Februar 2015, [online] Dostupné na internete: https://mega.co.nz/#doc

[3] Viivo, Februar 2015, [online] Dostupné na internete: https://viivo.com/

[4] BoxCryptor, Februar 2015, [online] Dostupné na internete: https://www.boxcryptor.com/en

[5] Wuala, Februar 2015, [online] Dostupné na internete: https://www.wuala.com/en/learn/technology

[6] SpiderOak, Februar 2015, [online] Dostupné na internete: https://spideroak.com/

[7] Grolimund D. - Meisser L. - Schmid S. - Wattenhofer R., Cryptree: A Folder Tree Structure for Cryptographic File Systems, [online] Dostupné na internete: http://dcg.ethz.ch/publications/srds06.pdf

[8] Stark E. - Hamburg M. - Boneh D., Symetric Cryptography in Javascript, 2009, Annual Computer Security Applications Conference, [online] Dostupné na internete: https://bitwiseshiftleft.github.io/sjcl/acsac.pdf

[9] SJCL, Stanford Javascript Crypto Library, April 2015, [online] Dostupné na internete: https://github.com/bitwiseshiftleft/sjcl/

[10] SJCL, Stanford Javascript Crypto Library, April 2015, [online] Dostupné na internete: https://github.com/bitwiseshiftleft/sjcl/wiki

[11] NIST, Processing Standards Publication 197 - AES, April 2015, [online] Dostupné na internete: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf