VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FOTOGRAFIE S ČASOVÝM RAZÍTKEM

PHOTOS WITH TIMESTAMP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATĚJ MLEJNEK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Mlejnek Matěj**
Program: Informační technologie
Název: **Fotografie s časovým razítkem**
Photos with Timestamp
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte literaturu a existující řešení na téma časová razítka a mobilní aplikace se zaměřením na časová razítka fotografií.
2. Navrhněte postup pro časové razítkování fotografií a skupin fotografií v mobilní aplikaci, buď "standalone" nebo "klient/server".
3. Vyberte vhodnou implementační platformu a způsob implementace navrženého postupu.
4. Implementujte navržený postup na vybrané platformě a demonstrujte funkčnost na vhodném příkladě.
5. Vyhodnoťte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zemčík Pavel, prof. Dr. Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce řeší způsob návrhu a implementace systému pro vytváření časových razítek k fotografiím po skupinách. Zabývá se možnostmi zpětného ověřování autentičnosti těchto vytvořených skupinových (hromadných) časových razítek. Celý systém je možné rozdělit na čtyři části. Část s mobilním zařízením, ve kterém jsou vytvářeny uživatelem fotografie. Druhá čast je server, na kterém jsou jednotlivé fotografie zálohovány a přeposílány k časovému orazítkování. Třetí část se skláda ze serveru zařizujícího vytváření hromadných časových razítek. Čtvrtá je složena z implementace uživelských nástroju pro zpětné ověření, zda byla fotografie úspěšně časově orazítkována.

Abstract

This thesis describes the design and implementation of a system for creating timestamps for photos in groups. It deals with the ways the group timestamp of multiple photos can be verified all at once. The whole system can be split into four parts. The first part is a mobile application, in which photos are taken by the user. The second part is the server on which photos are backed up and which relays the photos to a server that makes group timestamps. The third part is the server itself that creates the timestamps. The fourth part consists of user tools for retrospectively checking the existence of a valid group timestamp.

Klíčová slova

fotografie, elektronický podpis, časové razítko, certifikáty, šifrování, kryptografie, RSA, SHA, hashování, Android, serverless, Firebase, AWS.

Keywords

photo, digital signature, digital timestamp, certificates, encryption, cryptography, RSA, SHA, hashing, Android, serverless, Firebase, AWS.

Citace

MLEJNEK, Matěj. *Fotografie s časovým razítkem.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing. Pavel Zemčík

Fotografie s časovým razítkem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Dr. Ing. Pavla Zemčík, a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Matěj Mlejnek
15. května 2019

Poděkování

Chtěl bych poděkovat svému vedoucímu práce prof. Dr. Ing. Pavlu Zemčíkovi, který mě poskytl odbornou pomoc a řediteli CVIS VUT panu Ing. Jaromíru Marušinci, Ph. D. za umožnění použití časového razítka VUT v Brně.

Obsah

1 Úvod **2**

2 Shrnutí dosavadního stavu **3**
2.1 Fotografie 3
2.2 Elektronický podpis 5
2.3 Časové razítko 5
2.4 Digitální certifikáty 6
2.5 Šifrování a hashování 6
2.6 Android 7
2.7 Serverless 10
2.8 Existující aplikace 13

3 Zhodnocení dosavadního stavu a plán práce **15**
3.1 Návrh hromadného podpisu fotografii 16
3.2 Plán tvorby aplikace 17

4 Popis vlastní práce **19**
4.1 Použité technologie a služby 21
4.2 Popis komunikace, datový model 22
4.3 Mobilní aplikace 26
4.4 Zálohovací server 30
4.5 Server hromadného podpisu 31
4.6 Verifikátor a webová stránka 33

5 Testování, využití v praxi a další možnosti rozvoje **37**
5.1 Testování 37
5.2 Využití v praxi 39
5.3 Zhodnocení 40
5.4 Návrh rozšiřujících funkcí 42

6 Závěr **44**

Literatura **45**

A Obsah přiloženého pamětového média **50**

B Uživatelské rozhraní **51**

C Manuály **57**

Kapitola 1

Úvod

V současné době, kdy se rozšiřuje elektronizace jak ve veřejném, tak v soukromém sektoru se lidé čím dál častěji setkávají s právními problémy, které se týkají věcí ve virtuálním světě, jako je kontrola identity dané osoby, dokázáním, že dané dokumenty byly pořízeny daným člověkem a tudíž pak i zjišťování autentičnosti podvržených dokumentů. Tématem této bakalářské práce je kontrola pravdivosti pořízených dat z ohledu jejich časové existence, což úzce souvisí s pořizováním kvalifikovaných časových razítek. Přesněji se práce zabývá tím, jak pořídit fotografie, tak aby bylo v budoucnu možné co nejpřesněji dokázat jejich původ a hlavně tento postup co nejvíce zefektivnit.

Řešení pro vytvoření soudně prokazatelných důkazních materiálů mě jako téma zaujalo a vzhledem k zajímavému tématu a širokému prostoru pro různá zajímavá rozšíření mě zábývání se tímto tématem nadchlo natolik, že jsem si ho zvolil jako bakalářskou práci. Finálním výstupem této bakalářské práce by mělo být prostudování a sjednocení poznatků této problematiky a vytvoření demonstrační mobilní aplikace InsuranceAgent, zabývající se právě focením fotografií objektů za evidujícím účelem pro vytvoření důkazního materiálu v případě potřeby, poté shromáždění těchto fotografií a zaznamenáním jejich platnosti.

Obsah práce je rozdělen do šesti hlavních kapitol, z toho této úvodní, shrnutí dosavadního stavu, zhodnocení dosavadního stavu a plán práce, popisu vlastní práce, testování spolu s využitím v praxi a dalších možnostech rozvoje a nakonec závěru. Ve druhé kapitole Shrnutí dosavadního stavu 2 jsou podrobně popsány jednotlivé klíčové technologie a výrazy, na než je v následujících kapitolách odkazováno. Třetí část této bakalářské práce tvoří kapitola s titulem Zhodnocení dosavadního stavu a plán práce 2. Jak už název napovídá v této kapitole je podrobně rozebráno zadání a to, co jsem se rozhodl určit si za cíl funkčnosti demonstrační aplikace a potažmo i celé této práce. Čtvrtá kapitola se týká popisu vlastní práce 4. Je v ní popsán můj postup při vytváření této práce, je zde popsáno, jak byla vytvářena demonstrační aplikace a jakých zkušeností a výsledků jsem dosáhl. V páté a předposlední pasáži 5 je nejprve testování, poté využití v praxi a nakonec navázání na výsledek mé vytvořené práce, ve kterém je uvedeno mé celkové zhodnocení a popis dalšího možnosti rozvoje. Uzavírající šestá kapitola je závěr 6, která obsahuje shrnutí a zhodnocení o tom do jaké míry se podařilo naplánovaných výsledků dosáhnout. Dále jsou tam uvedeny jednotlivé body zadání, kde u kterého bodu je popsáno moje vlastní uvážení o tom do jaké míry se mě ho podařilo naplnit.

Kapitola 2

Shrnutí dosavadního stavu

V této kapitole jsou popsány klíčové technologie týkající se této bakalářské práce. Jsou zde uvedeny ty nejpodstatnější informace, které by měl čtenář znát ke správnému pochopení díla. Vzhledem k maximálnímu celkovému rozsahu práce nebylo možné popsat vše.

2.1 Fotografie

Klasické fotografie

Výsledek chemického procesu zachycení světla ze statického obrazu na plátno, například film, ze kterého se výsledný zaznamenaný obraz vyvolává, je fotografie. Zaznamenání, neboli vyfocení světla do obrazu, kterým se zachycuje vzhled objektů na něž byl objektiv fotoaparátu při focení zaměřený.

Prvotní pokusy o zachycení obrazu využívaly chemických procesů. Například vystavení na plátno potažené chloridem stříbrným, jehož světlocitlivost zachytila stíny, ale tato metoda nedokázala vytvořit výslednou fotografii, která by po krátké době nezbělala. První fotografie, kterou se podařilo pořídit, tak aby udržela obraz natrvalo je fotografie muže vedoucího koně z roku 1825 viz 2.1. [15]

Obrázek 2.1: Nejstarší fotografie na světě s motivem mladého chlapce, který vede koně do stájí z roku 1825 [15]

Do roku 2002 se o této fotografii veřejně nevědělo a za nejstarší fotografii byla považována fotografie La cour du domaine du Gras (doslova Dvůr na statku Le Gras)), vznikla asi v roce 1826 (podle některých zdrojů v roce 1827). Obě tyto fotografie byly vytvořeny ve

Francii vynálezcem Josephe Nicéphorem Niépcem. [33]. Způsob pořízení fotografie se podařilo zaznamenat pomocí jím vytvořené metody zvané Heliographie vynalezené v roce 1825. Tato metoda využívá chemickou vlastnost tvrdnutí asfaltové vrstvy, která v osvětlených částech tvrdne rychleji a tvoří v obraze bílé plochy. Ve správném okamžiku se nestvrdnutý asfalt smyje pomocí roztoku smíchaného levandulového oleje a terpentínu. Tmavé plochy fotografie (stíny) jsou tvořeny cínovou podložkou. [29]

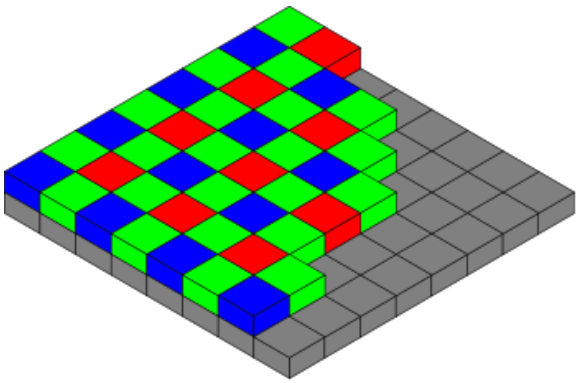
První barevná fotografie The Tartan Ribbon (Stužka) byla vytvořená James Clerk Maxwellem na principu spojení zachycení obrazu přes červený, zelený a modrý filtr a poté složení těchto filtrů dohromady. Tento princip aditivního míchání barev se i v dnešní době používá jako primární způsob zobrazení barevného obrazu na displej. [55]

Digitální fotografie

Digitální fotografie oproti klasickým používají digitální technologie pro zachycení světla do obrazu. Výsledný obraz ukládají například jako: png, jpeg, gif,psd,tiff,raw atd.

Bayerova maska

Je jednou z nejrozšířenejších variant masky filtrů. Obraz je přes čočku nasměrován do senzorů, například pomocí elektronového detektoru světla a je měřena intenzita pro jednotlivé body (pixely). V RGB variantě se na senzoru se střídají červené, zelené a modré filtry [9]. Černobílý obraz je přes ně převeden na barevný tak, že před kazdým pixelem je onen barevný filtr, který propustí pouze světlo v určitém rozsahu vlnových délek. [42]



Obrázek 2.2: Typycká Bayerova maska RGB [9]

Formát EXIF

Neboli Exchangeable image file format je specifikace pro formát metadat k fotografiím. Tyto metadata jsou přidávany do souborů digitálních fotografií přímo fotoaparáty. Ty Informace se vkládají interně do hlavičkové části souboru. Mezi podporované souborové formáty patří například JPEG, TIFF revize 6.0, PNG a RIFF WAVE. Nepodporuje ovšem soubory typu GIF ani žádné video formáty. Struktura EXIF dat je převzatá ze souborového formátu TIFF a jsou si proto velice podobné. [53]

Mezi metadata v exifu, neboli exif štítky patří například údaj o použitém fotoaparátu, GPS lokace (GPSLatitude, GPSLongtitude), nastavení fotoaparátu a další [40].

2.2 Elektronický podpis

Oproti klasickému podpisu je elektronický neboli digitální podpis vytvořen ze zdrojových dat, které znázorňují například data emailu, digitální fotografie nebo nějakého jiného elektronického dokumentu. Tento soubor dat je vytvářen pomocí šifrovacích metod (viz podkapitola Šifrování v kapitole 2.5) na základě obsahu původního podepisovaného dokumentu. Existuje zde logická vazba mezi ním a vůči původnímu souboru. Podepisující za použití svého soukromého klíče daný zaručený elektronický podpis vytvoří a pomocí veřejného klíče je možné ověřit autentičnost původu dokumentu, přesněji řečeno v okamžiku vytvoření zaručeného elektronického podpisu neboli elektronické značky zaručuje, že dojde-li k porušení obsahu datové zprávy od okamžiku, kdy byla podepsána nebo označena, toto porušení bude bezpečně možné odhalit [1].

přímé ověření - přímé ověřování původního dokumentu porovnáváním jeho celého obsahu. Přímé ověření nedává v dnešní době neustálých přenosů dat po internetu smysl [31].

nepřímé ověření - na principu vytváření otisku (hashe) dokumentu. Tato informace je ve srovnání s původním dokumentem řádově menší a často má specifickou délku. Elektronický podpis může být uložen do metadat jako součást dokumentu nebo může existovat nezávisle jako vedlejší soubor [31].

Kvalifikovaný elektronický podpis

Je podpis vytvořen na základě kvalifikovaného certifikátu podkapitola 2.4, který obsahuje věřejný klíč, a soukromého klíče. Kvalifikované elektronické podpisy jsou povinné pro veřejnoprávně podepisující (stát, samostatný správní celek, právnická osoba řízená zákonem, veřejná správa, města, obce, společnosti zřízené státem, atd.) Kvalifikovaný elektronický podpis je právně rovnocenný vlastnoručnímu podpisu a je uznáván v rámci celé EU [47] [2].

2.3 Časové razítko

Označuje čas existence dokumentu, podle certifikační autority, která musí být považována za důvěryhodnou. Certifikační autorita podepíše vlastním privátním klíčem elektronický dokument zaslaný k časovému orazítkování, vytvoří podpis do kterého vloží aktuální čas [31]. Tento výstupní soubor je samotné časové razítko, které má za úkol v budoucnosti sloužit k ověření elektronického dokumentu na základě, kterého bylo vystaveno. Při zpětném ověřování je potřeba zanést do systému podepisování tuto důvěryhodnou třetí stranu, proto aby nechodázelo k podvrhu. Časové razítko se vytvaří zpravidla k elektronickému podpisu dokumentu [47].

Z časového razítka ovšem nevyplývá to zda byl v danou dobu určitý dokument vytvořen, ale pouze to, že již existoval.

Kvalifikované časové razítko

Stejně jako kvalifikovaný elektronický podpis musí být vytvořeno na základě stejného standardu kvalifikovatelnosti. Samotné časové razítko, tak aby bylo kvalifikované, musí být vydáno kvalifikovanou certifikační autoritou.

Proces získání časového razítka

Časové razítko, o kterém se zde hovoří je to, které odpovída standardu `RFC3161` [7].

- **Query** .tsq - soubor, který uživatel vytvoří ze souboru, který chce podepsat.
- **Reply** .tsr - soubor obsahující samotné časové razítko.

Zjednodušeně řečeno pokud uživatel hodlá certifikační autoritu požádat o vystavení časového razítka vytvoří `.tsq` soubor (například pomocí knihovny OpenSSL [43]) a tento soubor odešle přes HTTPS. Certifikační autorita mu vystaví `.tsr` reply soubor samotného časového razítka.

2.4 Digitální certifikáty

Digitální certifikát je datová struktura, která obsahuje veřejný klíč (ve formátu X.509) svého majitele. Tento certifikát identifikikuje svého držitele, jelikož je vystaven certifikační autoritou [47]. Na základě obsaženého veřejného klíče je ověřován podpis podrobně viz podkapitola šifrování v kapitole 2.5. Pojem kvalifikovaný certifikát udává certifikát, který odpovídá obdobným náležitostem jako kvalifikovaný podpis [47].

Certifikační autorita

Certifikační Autorita (CA) je zpravidla nezávislý subjekt, který vydává digitální SSL (Secure Sockets Layer) certifikáty. Certifikáty vydává podepsáním souboru dat žadatele pomocí svého veřejného klíče. Veřejný klíč autority by měl být považován za důvěryhodný. Certifikační autorita se stává kvalifikovaným poskytovatelem certifikačních služeb na základě dodržení nařízení v zákoně Zákon č. 440/2004 Sb. [3], který legislativně definuje všechny požadované specifikace, které musí daný subjekt splňovat. Mezi jeden z typů certifikátů získávaných od authority patří i certifikát obsahující časový údaj vystavení, sloužící jako údaj značící časové razítko viz kapitola 2.3. Mezi certifikační autority patří například Postsignum[1], Free Timestamp Authority[2], Symantec (VeriSign)[3], GeoTrust, Thawte, RapidSSL, Comodo či TrustWave.

2.5 Šifrování a hashování

V této sekci je popsáno šifrování a hashování. Nejprve bych rád zmínil samotný rozdíl mezi těmito pojmy, jelikož šifrování vytváří soubor, z kterého lze vytvořit původní soubor naproti tomu hashování vytváří pouze hash (otisk), což je určitá komprimace dat ovšem bez možnosti původní data získat [51].

Šifrování

V moderní kryptografii se pro vytvoření šifry původního souboru používají sofistikované matematické algoritmy. Jak už bylo řečeno, tyto algoritmy vytvoří šifru, ze která se dá v budoucnu pomocí klíče původní soubor zrekonstruovat. Dělí se na symetrické a asymetrické a to podle použití klíčů pro šifrování a dešifrování původního souboru [60].

[1] http://www.postsignum.cz/index.php
[2] https://www.freetsa.org/index_en.php
[3] http://sha256timestamp.ws.symantec.com/sha256/timestamp

Symetrické šifrování

Pokud je zpráva zašifrována klíčem pak je možné i tímto stejným klíčem zprávu dešifrovat. Nastávají zde probémy s režií klíču, jelikož v případě komunikace dvou zařízení přes veřejnou síť je potřeba, aby oba objekty vlastnily stejný klíč.

Asymetrické šifrování

K podepisování se používají dva klíče: soukromého (privátního) a veřejného (public) klíče. Soukromým klíčem je podepsán soubor dat a z něj tak vyvtořený podpis. Z tohoto podpisu je možné vytvořit původní soubor pomocí veřejného klíče.

RSA Jeden z nejpoužívanejších algoritmů pro asymetrické šifrování. Princip je založen na předpokladu vysoké obtížnosti rozdělení velkého čísla na součin prvočísel. Při dostatečné délce klíče je tato metoda považována za bezpečnou (2048 bitů [10]). Použití této metody je možné i pro vytvoření elektronického podpisu [60].

PKCS8 jedná se o standardizovaný formát zapsání soukromého klíče podle normy `RFC5208` [37].

X.509 X.509 standard pro formát veřejného klíče podle normy `RFC3447` [35].

SHA256withRSA spojení hashovací metody `SHA256` spolu s šifrovací RSA metodou. Nejprve je vygenerován otisk a ten se potom podepíše.

Hashování

Hashování probíhá tak, že z původního souboru je vytvořena komprimovaná verze (zpravidla o pevné velikosti) na základě algoritmu, pomocí logických matematických funkcí, tak že tento hask (otisk) je možné pomocí implementace stejného algoritmu na jakémkoliv jiném systému replikovat. Tento princip toho, že není potřeba odeslat originální data je možné využít například při přihlašování uživatelů, tak aby hostitelský server neukládal přímo hesla jednotlivých uživatelů, ale pouze jejich otisky [52].

SHA Se dělí na dvě základní skupiny `SHA-1` a `SHA-2` [39].

SHA-1 Dříve v digitálních podpisech, již nepoužívané [52].

SHA-2 Například SHA-256 a SHA-512 jedny z nejčastěji používaných algoritmů.

MD5 Rychlý algoritmus z roku 1997, v dnešní době považován za kompromitovaný [17].

2.6 Android

Android je open source platforma od společnosti Google, která je založená na Linuxovém jádře. Přesto že Android platforma slouží především pro mobilní zařízení (telefony, tablety, hodinky), je využívána i v chytých televizích *Android TV*, automobilech *Android Auto* nebo v mnoha dalších zařízeních [16].

Aktuální zastoupení verzí viz tabulku 2.1. Z tabulky lze vyčíst to, že pokud aplikace bude vyžadovat minimální požadavky API, naimplementované za pomocí SDK - software development kit ve verzi 16 bude dostupná na více než 99% zařízení.

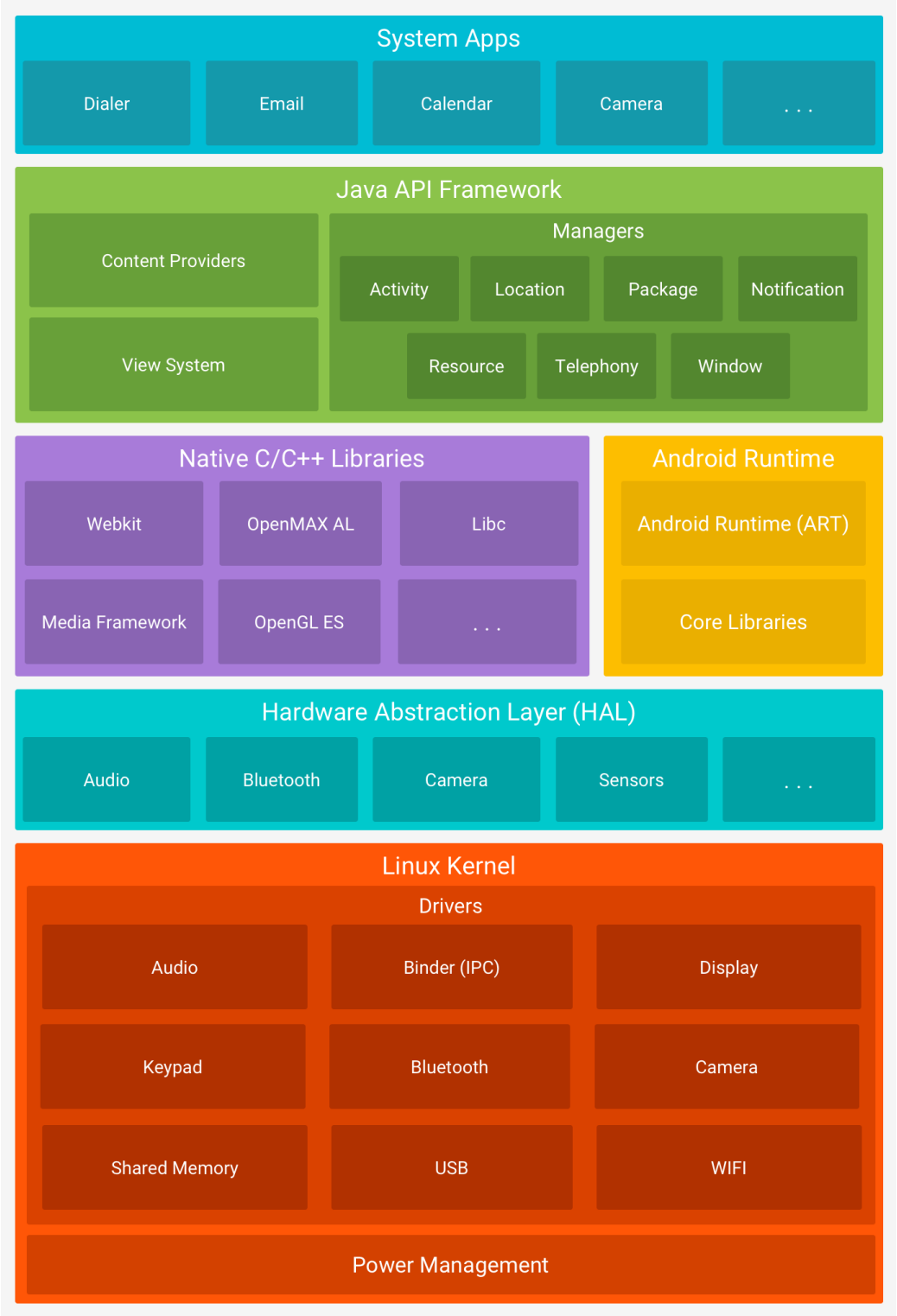
Tabulka 2.1: Zastoupení verzí androidu[4]

Verze	Jméno	API	Zastoupení
2.3.3 - 2.3.7	Gingerbread	10	0,3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0,3%
4.1.x	Jelly Bean	16	1,2%
4.2.x		17	1,5%
4.3		18	0,5%
4.4	KitKat	19	6,9%
5.0	Lollipop	21	3,0%
5.1		22	11,5%
6.0	Marshmallow	23	16,9%
7.0	Nougat	24	11,4%
7.1		25	7,8%
8.0	Oreo	26	12,9%
8.1		27	15,4%
9	Pie	28	10,4%

Android architektura

Architektura se skládá z několika vrstev a to linuxového jádra, HAL, nativních knihoven C/C++, Android Runtime, Java API frameworku a systémovým aplikacím, viz obrázek 2.3.

[4]Převzato z [27].



Obrázek 2.3: Android architektura[5]

[5]Převzato z [28].

Linux Kernel jádro celé architektury. Například Android Runtime (ART) přes jádro komunikuje s hardwarem a tam je možné programovat na velmi nízké programové úrovni (spravovat vlákna a jednotlivé procesy, low-level správa paměti a další) [28].

Hardware Abstraction Layer(HAL) Rozhraní a k němu naimplementovaná velká škála knihoven sloužící aplikacím k zjednodušení interakce s jednotlivými hardwarovými komponentami (tlačítka, gps, kamera, wifi, bluetooth a další).

Native C/C++ Libraries Knihovny napsané v nativním jazyce C nebo C++.

Android Runtime Samotné aplikace běží na této vrstvě, která se o ně stará. Od verze 5.0 (API level 21) je pro každou nově spuštěnou aplikace vytvořen samostatný proces instance ART [28]. Jednotlivé procesy běží ve virtuálních kontainerech [6].

Java API Framework Rozsáhlý soubor nástrojů implementovaných v jazyku Java. Při používání tohoto rozhraní je program vytvářen pomocí jednotlivých stavebních bloků. Vývoj pro tuto platformu je díky těmto blokům značně zjednodušen jelikož je snadné je znovu použít a jejich skládáním vzniká systém, který je modulární, díky čemuž se zde relativně lehce odhalují chyby.

Mezi základní komponenty patří:

- Systém pohledů (View System) - který slouží pro sestavení uživatelského rozhraní.
- Správce zdroju (Resource Manager) - obsluha souborů obsahující textové řetězce aplikace (názvy, dimenze rozlišení, barvy) - grafické rozložení, vektorové obrázky a další.
- Správce oznámení (Notification Manager) - díky, kterému lze k aplikaci nastavit vlastní notifikace do navigační lišty.
- Správce aktivity (Activity Manager) - spravuje životní cykly jednotlivých grafických elementů a aktivit.
- Poskytovatelé obsahu - (Content Providers) - rozhraní skrz, které mohou aplikace navzájem komunikovat (Intents, startActivityForResult)

System Apps Systémové aplikace, jako je telefon, SMS zprávy, kalendář, internetový prohlížeč a další [28].

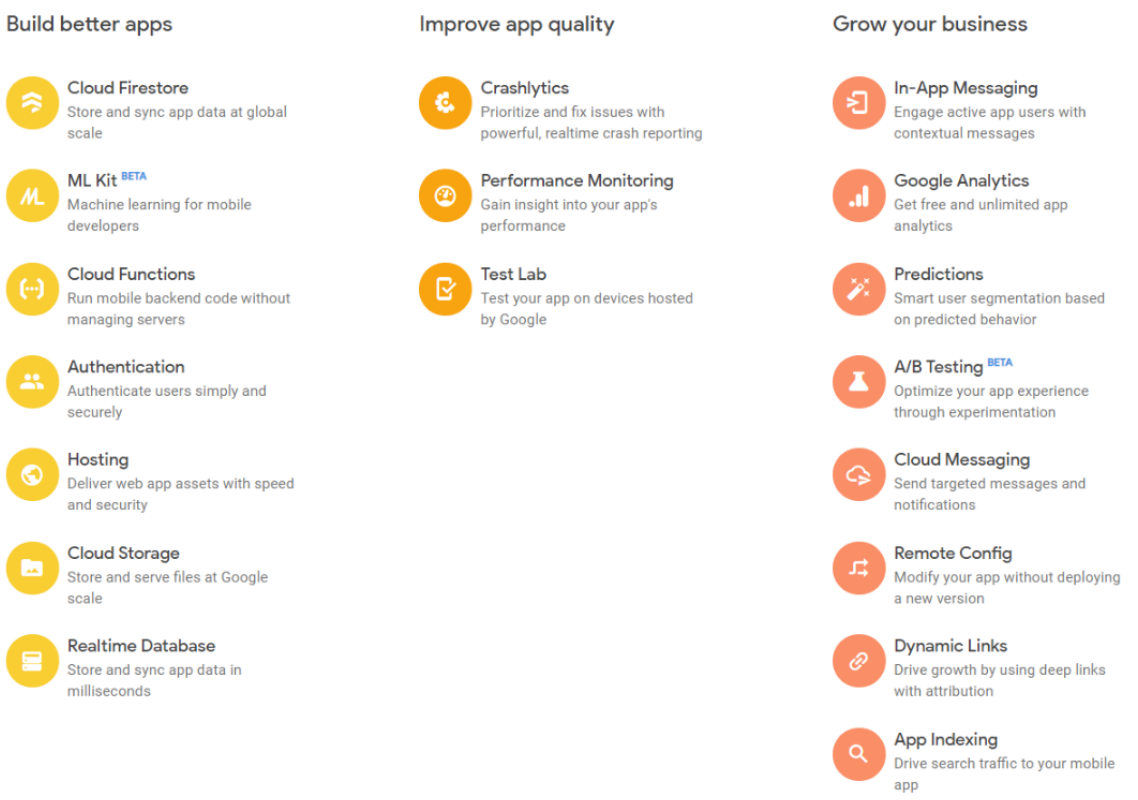
Dá se tedy také říct, že Android architektura se dělí na čtyři části: systémové aplikace (System Apps), frameworky (Java API Framework), knihovny (Android Runtime, HAL, Native C/C++ Libraries) a jádro(Linux Kernel).

2.7 Serverless

Pod názvem serverless si nepředstavujte aplikaci, která by neběžela na žádném serveru, jedná se prakticky o mikroslužbovou architekturu [19]. Výhodou je, že jednotlivé procesy (metody) zde mají svoji režii a nedochází mezi nimi ke konfliktům. Poskytovatelé těchto služeb obstarávají veškerý hardware, na kterém běží uživatelské aplikace od jednotlivých zákazníků zároveň. Výhodou je zde jednoduchá škálovatelnost, flexibilita a zpravidla téměř 100% dostupnost. Cena za služby zahrnuje například čas, po který daná aplikace „běží“, neboli spotřebovává procesorový čas, dále pak na základě využití uložiště, přenesených dat a dalších alokovaných zdrojů podle konkrétního poskotovatele [38].

Firebase

Je vývojová platforma obsahující skupinu nástrojů (služeb) pro usnadnění práce při vývoji aplikací jako jsou Android, IOS a webové aplikace. Společnost Firebase byla založena roku 2011 a v roce 2014 ji koupil Google [61]. Firebase nabízí mnoho služeb týkajících se lepšího vývoje aplikací, zvýšení jejich kvality a rozvoje podnikání [25]. Pro jednotlivé aktuálně nabízení služby viz obrázek 2.4.



Obrázek 2.4: Služby nabízené od Firebase[6]

Co se týče ceny, tak zdarma plán Spark bohatě dostačuje pro implementaci aplikace s malým počtem uživatelů a bez velkého toku dat. Zahrnuje věci jako 1GB Firebase Realtime Database uložiště, 10 GB síťové komunikace měsíčně, 10000 ověření uživatele při přihlašování měsíčně a další.

Pokud uživatel plánuje v rámci Firebase Cloud Functions komunikovat mimo Google servery je potřeba upgradovat na plán Blaze ten má stejné limity zdarma jako plán Spark, ale při překročení je místo přerušení služba zpoplatněna [4].

Firebase Realtime Database

Firebase je NoSQL databáze, takže neobsahuje žádné tabulky. Ukládáme do ní stromově strukturovaná data jako ve formátu JSON[7]. Jediným rozdílem oproti klasickému JSONu je zde to, že jsou záznamy do jednotlivých polí vkládány pod seřazenými unikátními klíči,

[6] Převzato z [25].
[7] https://www.json.org/

a tudíž je možné přes ně iterovat ve správném pořadí od vzniku [61]. V každém zařízení připojeném do databáze se nachází instance a automaticky synchronizuje jakékoliv vzniklé změny [26]. Pro čtení a zapisování dat je použito rozhraní Firebase Database REST API pro manipulaci s daty z Firebase Realtime Database pomocí HTTPS komunikace [61].

Firebase Cloud Functions

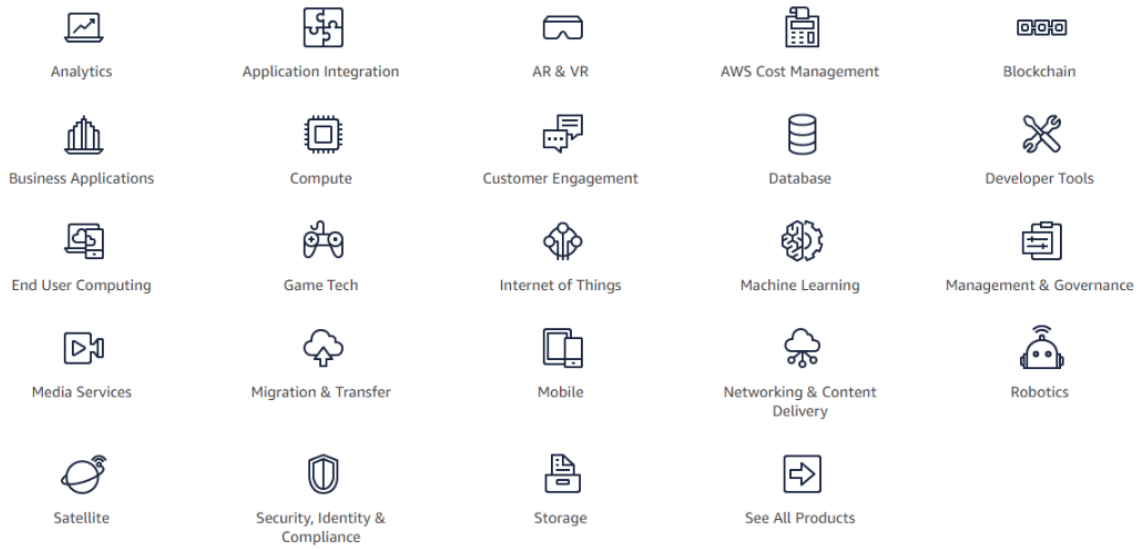
Cloud Functions for Firebase je služba, která automaticky spouští backendový kód na základě událostí jako například triggery ve Firebase Realtime Database(onCreate, onUpdate, onDelete, onWrite) nebo HTTPS requesty. Kód je umítěn na cloudovém uložišti Googlu a při spuštění běží vždy ve vlastní instanci [22].

Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) je multiplatformní notifikační nástroj, který dokáže doručit zprávy na jednotlivá zařízení. Takto se dá například odeslat notifikace z klientské aplikace a zobrazit v záhlaví mobilního telefonu zprávu sdělující uživateli konkrétní data nebo sloužící pouze jako notifikace, že aplikace vyžaduje pozornost. Tyto zprávy mohou být použité, tak aby udržovaly uživatele v obraze například při implementaci emailové aplikace okamžitě uživateli oznamovat, že mu přišel email. Tato notifikace může obsahovat až 4KB dat [23].

AWS

AWS (Amazon Web Services) je cloudová platforma, nabízející mnoho služeb, jako jsou výpočetní výkon, databázove uložiště, virtuální stroje a další, viz obrázek 2.5 [8].



Obrázek 2.5: Služby nabízené od Amazon Web Services[8]

[8]Převzato z [8].

EC2

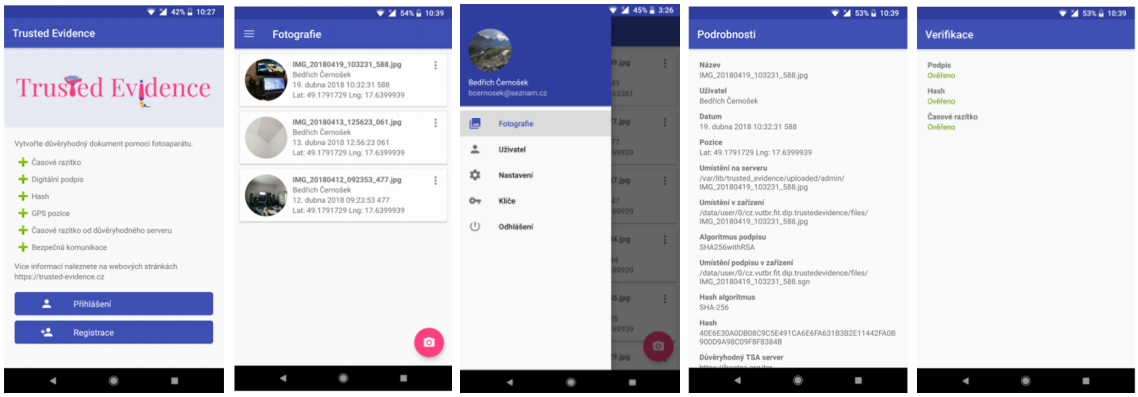
EC2 (Amazon Elastic Compute Cloud[9]) je služba poskytující virtuální výpočetní serverové instance [8].

2.8 Existující aplikace

V této sekci jsou zmíněné nejzajívavější řešení v mobilních aplikacích, které se snaží vytvořit evidenci o existenci fotografií, pak především ty pracující s časovými razítky k fotografiím nebo jiným elektronickým dokumentům.

Trusted evidence

Mobilní aplikace na platformě Android k zpracování obrazu a kryptografické ověření zdroje a času jeho vytvoření obrazu. Bezpečně pořizuje fotografie, provádí jejich zpracování a získává digitální podpis, časové razítko a GPS souřadnice [13].

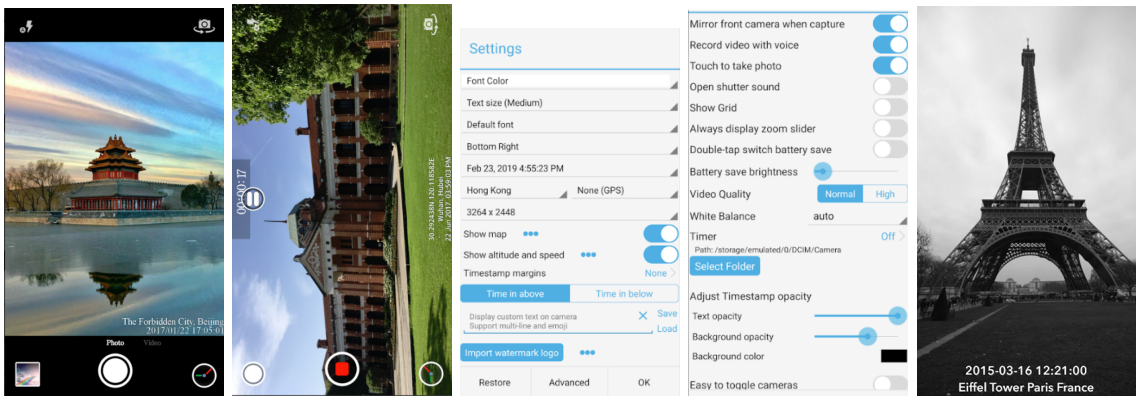


Obrázek 2.6: Obrazovky aplikace Trusted evidence[10]

Timestamp Camera Free

Je mobilní aplikace umožnující pořizování fotografií i videí, které označuje pomocí vodoznaku. Vytvořená data bohužel není možné považovat za dostatečně věrohodná, jelikož zde není použit žádný elektronický podpis, časové razítko ani žádná jiná metoda. Jediná volba ověření dané fotografie je tedy již zmíněný vložený vodoznak. Ten si uživatel pomocí rozsáhlého výběru v nastavení může nastavit podle vlastních představ například časový údaj rozšířit o vlastní text a nebo GPS souřadnice.

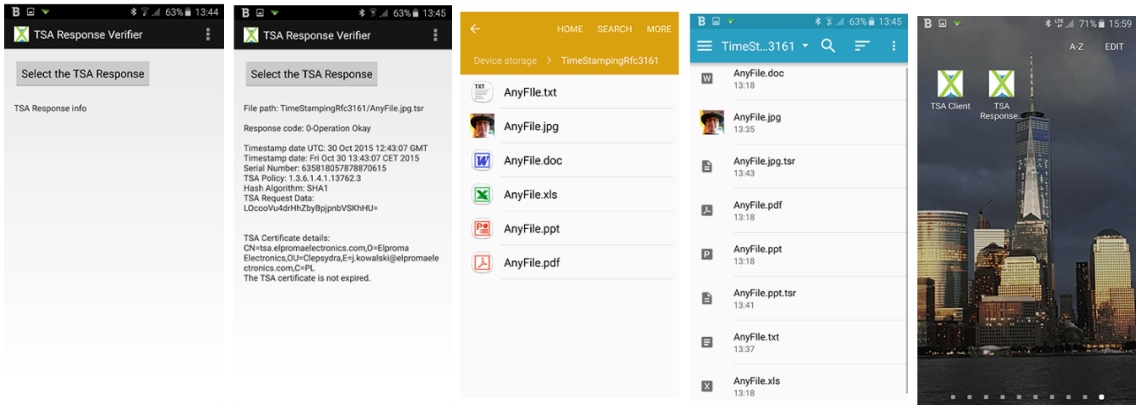
[9] https://aws.amazon.com/ec2/
[10] Převzato z [13].



Obrázek 2.7: Obrazovky aplikace Timestamp Camera Free[11]

RFC3161 TimeStamping Verifier

RFC3161 TimeStamping Verifier je aplikace pro vydání a ověření časových razítek vydaných od ELPROMA. Aplikace provádí jednoduché ověření souboru a časového razítka a je možné ji použít pro různé typy souboru jako například JPG, PNG, PDF, DOC, TXT, XLS a další [50].



Obrázek 2.8: Obrazovky aplikace RFC3161 TimeStamping Verifier[12]

[11] Převzato z [59].
[12] Převzato z [50].

Kapitola 3

Zhodnocení dosavadního stavu a plán práce

Po shlédnutí funkcionalit obdobných mobilních aplikací zabývající se tématikou podepisování a vytváření časových razítek fotograií zmíněné v přechozí kapitole 2.8 jsem vyvodil to, že je na první pohled vidět, že se aplikace dělí na dvě hlavní skupiny viz tabulku 3.1.

Tabulka 3.1: Příklady existujících mobilních aplikací

Aplikace	Ověřování	Autorita	Nastavení	Hromadný podpis
Trusted evidence	RFC3161	RFC3161[1]	rozlišení	ne
Timestamp Camera Free	vodoznak	ne	rozsáhlé	ne
RFC3161 TimeStamping V.[2]	RFC3161	ELPROMA	ne	ne

[1] Autorita RFC3161 volena uživatelem.
[2] RFC3161 TimeStamping Verifier

Jedny se snaží být co nejvíce uživatelsky přívětivé, ale většinou je zde zcela vynechán jakýkoliv mechanismus pro vytvoření časového razítka, na základě vystavení nějaké externí autority. Zato do druhé se řadí aplikace spíše techničtejšího rázu, ve kterých si člověk za to může nastavit klíče k použití, certifikační autoritu a je zde opravdu dbáno na to, aby časové razítko bylo kvalifikovaně vytvořeno a bylo ověřitelné. V žádném z aktuálních řešení, na které jsem narazil, však nebyla implementována funkcionalita vytvořit časové razítko pro více souborů zároveň. V této práci se proto zaměřím na vytvoření takového systému, ve kterém by se dalo časově orazítkovavat co největšího množství fotografií zároveň, avšak stále při zachování kvalifikovanosti a důvěryhodnosti těchto podpisů.

Jako demonstrační aplikaci, která bude reprezentovat princip mnou vytvořeného *Systému hromadného podpisu*, kde tedy „podpis“ představuje časové orazítkovaní fotografií (certifikační autorita podepisuje data a přidává k nim časovou hodnotu viz kapitola 2.3). Rozhodl jsem se vytvořit aplikaci, kterou jsem nazval InsuranceAgent. Reprezentuje možné využití mojí práce jako systém pro pojišťovnu, potažmo pojišťovací agenty, ve kterém je potřeba, aby bylo možné při vzniklé pojistné události, vytvořit fotografickou dokumentaci, která bude v budoucnu soudně průkazná. Ještě před tím než jsem začal přemýšlet o tom, jak bude samotné vytváření časových razítek k fotografiím probíhat, bylo potřeba se rozhodnout, jak bude vypadat architektura celého řešení. Na základě komplexnosti vytváření

a ověřování podpisů a razítek a nutnosti uchovávání fotografií jsem se rozhodl pro „klient/server“ síťovou architekturu. Vytváření časových razítek přímo v mobilním zařízení a uchovávání dat pouze na něm bez automatické zálohy, by nebylo příliš vhodné. Vzhledem k tomu, že v se mobilní aplikaci bude snímat fotografie, ze které se bude dále vytvářet její digitální podpis, ke kterému bude vystavováno kvalifikované časové razítko, je třeba zachovat původní soubor fotografie, tak aby bylo možné v budoucnosti zpětně bezpečně pravdivost vytvořených fotografií ověřit. Přemýšlel jsem jak bude uživatelsky nejpřívětivější tento původní soubor fotografie zachovat, aby k němu měl uživatel přístup a mohl si sám zkontrolovat jím vytvořené jednotlivé fotografie společně s doplňujícími údaji.

3.1 Návrh hromadného podpisu fotografii

V této sekci jsou popsány vlastnosti mnou navrženého principu hromadného podepisovaní fotografií, neboli vytváření časového razítka k několika souborům zároveň. Je zde popsán princip uložení v databázi, proces tvorby a následné ověření.

Návrh databáze

Na serveru hromadného podpisu, budou postupně do tabulky `Files` 3.2 databáze ukládány záznamy obsahující podepsaný soubor dat původní fotografie, spolu s SHA-256 komprimovanou verzí a odkazy na uživatele a záznam, pod který patří fotografie, do databáze Zálohovacího serveru a zatím prázdný odkaz, který bude po podpisu ukazovat na časové razítko uložené v oddělené tabulce `Packages` 3.3.

Tabulka 3.2: Návrh databázové tabulky `Files` k uložení dat k podpisu a ověřování fotografií

id	signature	compressed	user	task	package

Tabulka 3.3: Návrh databázové tabulky `Packages` hromadných časových razítek

id	timestamp

Návrh vytvoření hromadného podpisu

Hromadný podpis bude probíhat tak, že se vyhledají záznamy v tabulce `Files`, které dosud nemají přidělený řádný `package`, a tudíž nebyly časově orazítkovány. Z nalezených nepodepsaných řádků se vyčtou data ve sloupci `signature` a ty se za sebe sloučí do jednoho souboru, bude zde záviset na pořadí, které bude určeno na základě seřazení `id` atributu jednotlivých záznamů ve `Files`. Tento soubor je poté odeslán Certifikační autoritě, která k němu vytvoří časové razítko, toto časové razítko bude uloženo do tabulku `Packages` a k jednotlivým řádkům v tabulce `Files` bude přiděleno `id` nově vytvořeného řádku.

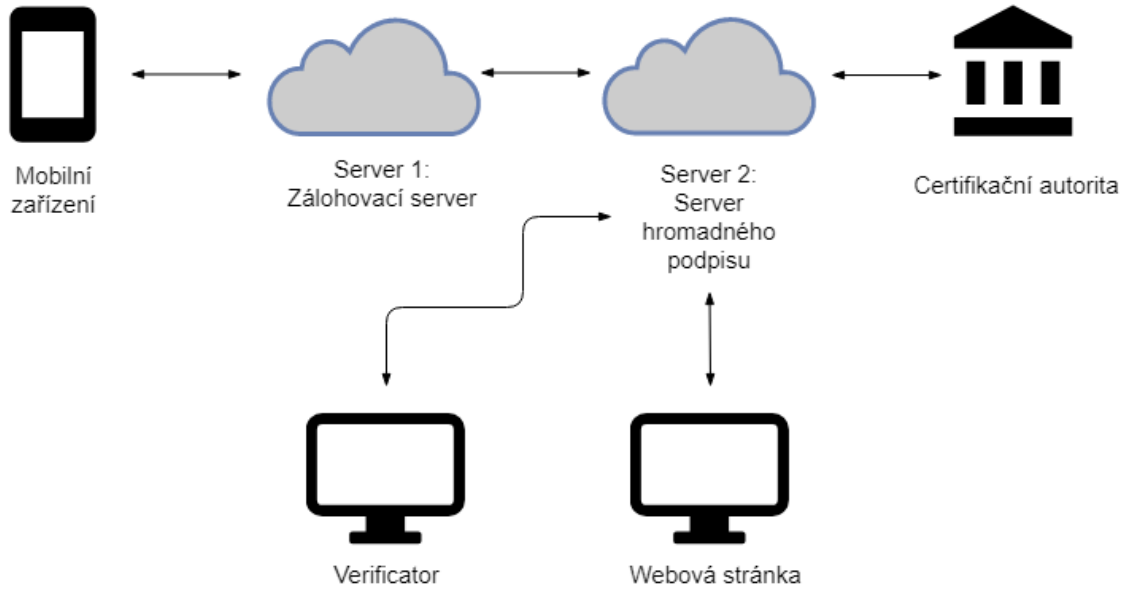
Návrh ověření časového razítka a autora z fotografie

Při kontrole, zda časové razítko náleží k fotografii, je proces ověření započat výběrem fotografie k ověření, z ní je pomocí SHA-256 vytvořena hash reprezentace, která je vyhledána

v tabulce `Files` v sloupečku `compressed`. Pokud byl nalezen řádek s touto hodnotou jsou pomocí hodnoty v sloupečku `package` vyhledány zbylé záznamy, které byly použity k vytvoření daného časového razítka. Z původních záznamů je vyselektován podpis - `signature` a znovu vytvořen soubor, který byl podepsán. Následně je tento soubor časového razítka možné ověřit vůči souboru složeném z jednotlivých původně použitých podpisů.

Pro ověření autora fotografie je potřeba vybrat správný řádek v souboru, který byl časově orazítkován a to na základě výběru správného indexu dodržením seřazovaní podpisů podle pořadí uložení v databázi.

3.2 Plán tvorby aplikace



Obrázek 3.1: Schéma navrhovaného systému hromadného podpisu.

Předsevzal jsem si vytvořit řešení, které se bude skládat z uživatelské aplikace v mobilním zařízení viz „Mobilní zařízení“ v obrázku 3.1, ze které budou jednotlivé záznamy odesílány na zálohovací server viz „Server 1: Zálohovací server“ v obrázku 3.1. Na tomto serveru budou shromažďovány data od jednotlivých uživatelů a bude sloužit jako prostředník mezi uživatelem a Serverem hromadného podpisu viz „Server 2: Server hromadného podpisu“ v obrázku 3.1. Jakmile se z mobilního zařízení požádá o časové razítko k záznamu, bude z tohoto Zálohovacího serveru odeslán příkaz na server hromadného podpisu, který bude zajišťovat shromaždování podpisů ke všem vytvořeným časovým razítkum. Zde se však budou ukládat pouze data potřebná k vytvoření časových razítek a navíc data reprezentující ukazatele do databáze zálohovacího serveru tak, aby se po úspěšném podpisu zpětně dalo notifikovat Zálohovací server a ten mohl dát vědět mobílnímu zařízení o úspěšném vystavení časového razítka. Server hromadného podpisu tedy shromažduje zaznamy k jednotlivým fotografiím a v časovém intervalu 24 hodin, je zde vytvářena žádost o vystavení časového razítka pro soubor obsahující reprezentaci dat všech nově příchozích dosud neorazítkovaných záznamů tak, aby se nemusely podepisovat zvlášt podrobněji popsáno v kapitole 3.1. Na Serveru hromadného podpisu se pak tedy uloží časové razítko a je zde uchováno, pro zpětné ověření.

Jsou zde uchovány i záznamy k fotografiím, to ovšem jen částečne, ale s veškerými potřebnými údaji k ověření původních fotografií na základě skládání těchto dat do původního časově orazítkovaného souboru, který je porovnán s vysvaveným časovým razítkem.

Ověřování bude probíhat přes webové rozhraní, uživatel bude přistupovat ve webové stránce k serveru hromadného podpisu. Nahraje fotografii a volitelně bude moct přiložit veřejný klíč autora, který fotografii vytvořil viz „Webová stránka“ obrázek 3.1.

V případě zájmu o exportovaní souborů hromadného podpisu vytvořenému k fotografii bude uživatel moci využít program Verificator „Verifikátor“ viz obrázek 3.1. Tento nástroj na základě vložení fotografie a autorova veřejného klíče dokáže soubory z podpisu (obsahující veškerá data z hromadného podpisu) uložit na lokální disk uživatelova počítače. Tyto soubory slouží k prokázání platnosti a doby existence podepsané fotografie a obsahují veškeré náležitosti potřebné k ověření hromadného podpisu v cizím systému.

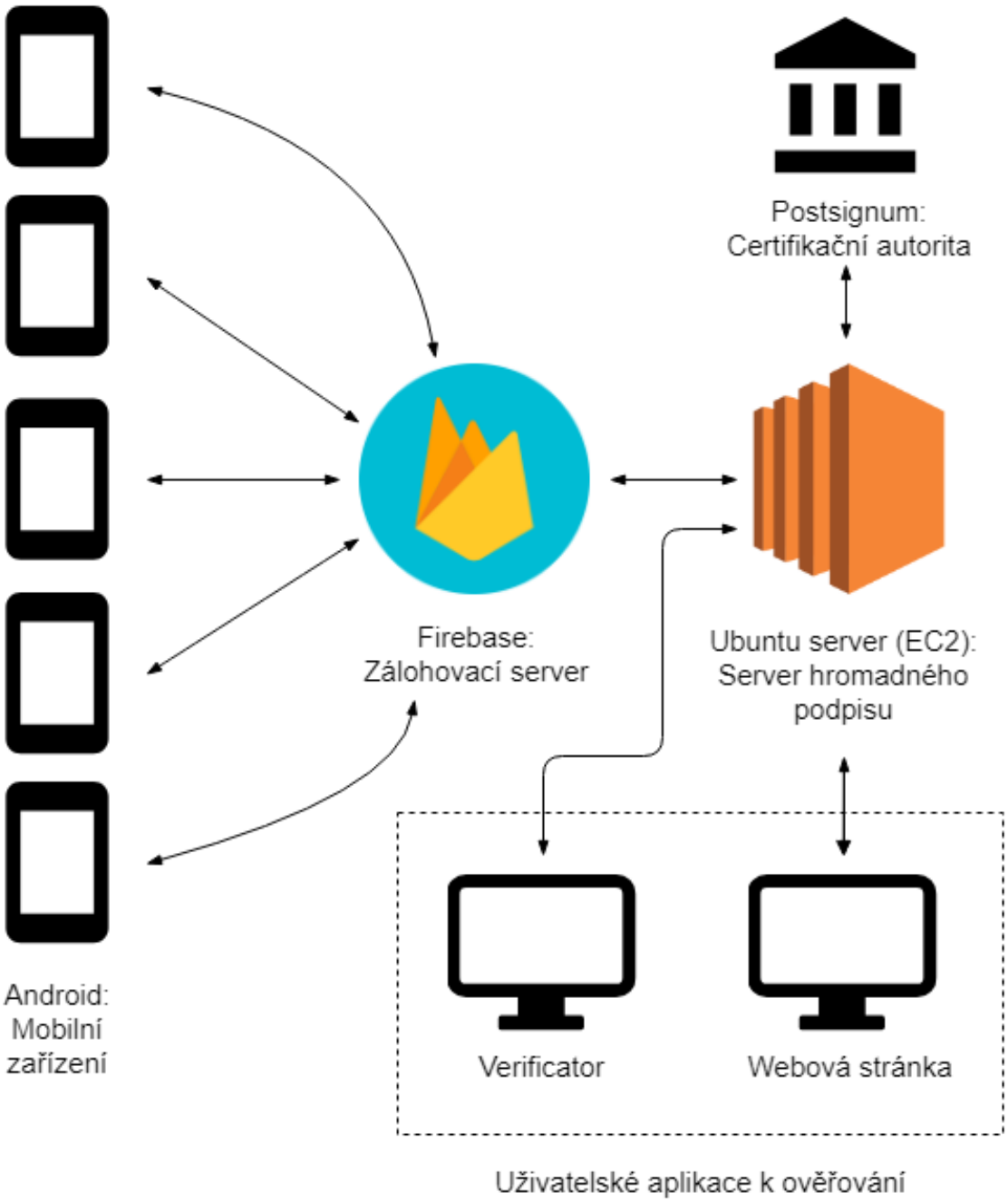
Kapitola 4

Popis vlastní práce

Mnou vytvořený systém pro vytváření hromadných podpisů obsahuje tři podsystémy a využívá externí podepisovací (certifikační) autoritu, viz kapitola 3.2.

Mobilní aplikace a zálohovací server jsou systémy, které by bylo možné nahradit či rozšířit o další systémy na stejné úrovni tak, aby se do *Serveru hromadného podpisu* dalo posílat data z více zdrojů. Dále je v této kapitole popsána komunikace mezi *Serverem hromadného podpisu* a zvolenou certifikační autoritou.

V úvodní části této kapitoly 4.1 jsou popsány jednotlivé technologie použité k implementaci Systému hromadného podpisu určenou pro demonstrační aplikaci InsuranceAgent 4.1. V rámci této kapitoly jsou dále u jednotlivých částí systému dodatečně uváděny podrobnější informace, jako například využití knihoven, které jsem se dodatečně rozhodl použít, vzhledem k požadavkům na jejich algoritmy.



Obrázek 4.1: Schéma Systému hromadného podpisu použité v demonstrační aplikaci InsuranceAgent.

Data uložená na jednotlivých mobilních zařízeních budou pro každého uživatele synchronizována ve všech zařízeních.

4.1 Použité technologie a služby

Mobilní aplikace

Při výběru platformy pro mobilní zařízení jsem se rozhodl tuto aplikaci vytvořit pro Android, jelikož má přibližně 75% zastoupení mezi mobilními operačními systémy [1] a minimální verzi SDK 16 viz 2.6. Při výběru implementačního jazyka jsem uvažoval nad Kotlinem, Javou a C#. Vzhledem k tomu, že již mám zkušenost pracovat ve vývojovém prostředí Android Studio [2], ve kterém je podpora pro Kotlin i Javu. Z nich jsem si vybral Kotlin, jelikož je modernější a díky své interoperatibilitě vzájemné s Javou si zachovám možnost používat Javové knihovny. Alternativou bylo použít jazyk C# ve vývojovém prostředí Visual studio pomocí nástroje Xamarin a .NET frameworku [3], ve kterém se sice dá vyvíjet pro Android, iOS i Windows Mobile zároveň, ale v pracnosti implementace a následného testování, by značně přesahovalo potřeby pro demostraci řešení v této bakalářské práci.

Výběr serverového hostingu

Za úkol vybrat jak a kde poběží zálohovací server a server pracující s hromadnými podpisy připadalo v potaz několik možností, nad kterými jsem uvažoval. První možnost vytvoření serveru běžícím na lokálním hardware jsem z praktických důvodů zavrhl, jelikož je potřeba komunikace mezi mobilním zařízením a servery, tudíž by musela výsledná aplikace buďto běžet pouze na lokální síti, a nebo by se musela nastavovat veřejná komunikace, což mi přišlo jako nepraktické a zbytečně vzdálené od tématu mojí bakalářské práce. Proto jsem se rozhodl zvolit si serverless infrastrukturu viz 2.7, vybíral jsem mezi AWS, Firebase, Parse, MongoDB, Back4App. A jelikož mi přišlo jako názorné pro mé řešení oba servery udělat na rozdílných infrastrukturách, aby bylo zřetelné, že se nejedná o stejný systém, zvolil jsem si, že použiji AWS a Firebase, což byly mnou již z dřívějška známé služby a v obou případech je možné přes počáteční kredit zdarma, nebo měsíční limit vybudovat aplikaci bez vzniklých nákladů. Včetně následného testovaní a použití pro demonstrační účely. Vzhledem k tomu, že s mobilním zařízením komunikuje pouze zálohovací server, vybral jsem v tomto případě použít Firebase. Pro server hromadného podpisu jsem zvolil použití AWS.

Zálohovací server

Z vybrané Firebase služby jsem pro synchronizaci dat uživatelů se serverem použil službu Firebase Realtime Ratabase viz 2.7. Pro odesílání dat využívám trigger nad databází napsaný v Javascriptu, který běží přes Firebase Cloud Functions 2.7. Javascript jsem se rozhodl použít proto, protože s TypeScriptem nemám tak velkou zkušenost. Používám zde engine Node 8 [4] a eslint ve verzi v4.12.1 [5]

Server hromadného podpisu

Server homadného podpisu poběží jako Ubuntu Server 18.04 LTS (HVM) [6] instance v rámci AWS služby EC2 popsané v kapitole 2.7. Na aplikační vrstvě běží server a podepisovací

[1] http://gs.statcounter.com/os-market-share/mobile/worldwide
[2] https://developer.android.com/studio
[3] https://visualstudio.microsoft.com/cs/xamarin/
[4] https://nodejs.org/es/blog/release/v8.0.0/
[5] https://eslint.org/docs/2.0.0/user-guide/configuring
[6] http://releases.ubuntu.com/18.04/

skript, které jsem naimplementoval v jazyce Python 3.6[7]. Data jsou na server zasílána pomocí HTTPS protokolu a ukládány do lokální MySQL databáze ve verzi 5.7.26 [8].

Verificator

Tento skript je rovněž jako **Server hromadného podpisu** napsán v jazyce Python 3.6. A komunikace s ním probíhá přes HTTPS protokol.

Webová stránka

Webová stránka bude napsána v jazyce HTML a bude komunikovat se **Serverem hromadného podpisu** přes HTTPS protokol.

Certifikační autorita

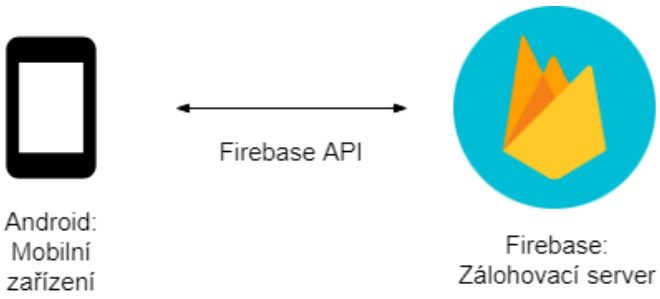
Vzhledem k tomu, že mi bylo umožněno použít časové razítko VUT v Brně zprostředkované vrámci služby Postsignum, tak jsem si tuto službu zvolil jako certifikační autoritu pro vytváření časových razítek. Z důvodu, že mi byl přidělen pouze omezený počet kvalifikovaných časových razítek od Postsignum, tak jsem si jako sekundární certifikační autoritu vybral „Free Time Stamp Authority“, přestože nesplňuje náležitosti, aby mohla vystavovaná razítka být považována za kvalifikovaná, viz kvalifikované časové razítko v kapitole 2.3. Jedná se ovšem o službu, která je bezplatná, a tudíž je pro testovací a názorný účel v rámci této bakalářské práci, napříč nevěrohodnosti použita.

4.2 Popis komunikace, datový model

V této sekci jsou popsány typy komunikace mezi jednotlivými entitami v implementaci Systému hromadného podpisu pro demonstrační aplikaci InsuranceAgent.

Mobilní zařízení a zálohovací server

Jednotlivá mobilní zařízení mají svá data synchronizovaná vůči Firebase Realtime Database 2.7, jenž je použit jako zálohovací server viz obrázek 4.2. Díky této automatické synchronizaci je zachována integrita dat a možnost aktualizace jednotlivých úkolů tj. záznamů bez zbytečné režie.



Obrázek 4.2: Schéma komunikace mezi Mobilním zařízením a Zálohovacím serverem v rámci implementace Systému hromadného podpisu použité v aplikaci InsuranceAgent.

[7] https://www.python.org/downloads/release/python-360/
[8] https://www.mysql.com/

Úkoly pro pojišťovací agenty jsou vytvářeny přímo v mobilní aplikaci a jsou uloženy ve formátu JSON viz 4.2. Kořenový uzel obsahuje element `tasks`, který obsahuje seznam `User UID` - unikátních klíčů jednotlivých uživatelu. Důvodem proč existuje tento zapouzřující element `tasks` je ponechání si možnosti pro rozšíření vytvořením vedlejšího uzlu například k implementaci rozšíření režie správy uživatelů. Vytvořením zde určitého whitelistu pro povolení zda od daného uživatele následně odesílat fotografie k podpisu viz 5.4. Ke každému uživateli dále patří úkoly s vygenerovaným unikátním klíčem. Ty už obsahují hlavní informace o záznamu ke specifické fotografii. Význam jednotlivých atributů je popsán níže.


```
{
   "tasks":{
      "hduTqxQPBwWEcl7jE4OlukhwmOS2":{
         "-LdokpLcKkgo77W1HJgr":{
            "completed":true,
            "compressed":"F7D93CC35C436E6592CADF5FB867C0B8211768DEA9...",
            "date":"1556737205433",
            "description":"crashed car",
            "hidden":false,
            "key":"-LdokpLcKkgo77W1HJgr",
            "locked":true,
            "name":"InsuranceTask1",
            "photo":"/9j/4QE8RXhpZgAATU0AKgAAAAgABwEAAAMAAAABD8AAAAE...",
            "publicKey":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQE...",
            "signature":"o8w/fvDBKvGOOKn381Cwx0/CT0yzhBLkSuJhlSpx6rs..."
         }
      }
   }
}
```


Výpis 4.1: JSON struktura uložení dat v mobilním zařízení a v Firebase

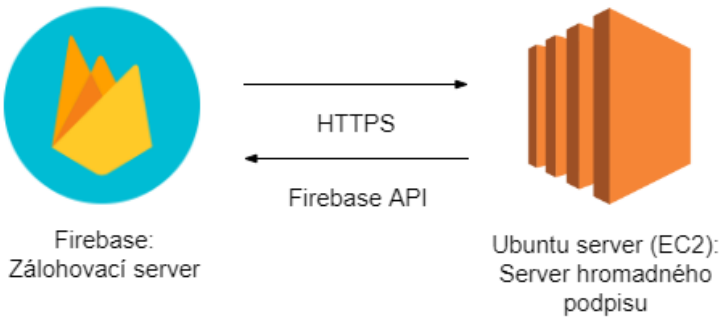
- **completed**
  označení zda byl záznam úspěšně podepsán
- **compressed**
  hexadecimální forma SHA256 vytvořená z fotografie
- **date**
  datum vytvoření časového razítka
- **description**
  volitelný atribut pro podrobnější popis záznamu
- **hidden**
  pomocný příznak, pro skrytí smazaného záznamu, bez propagace smazání do databáze
- **key**
  pomocný atribut sloužící jako hodnota třídy v mobilní aplikaci, pro snazší manipulaci se záznamem

- **locked**
  příznak, který je nastaven při žádosti o odeslání fotografie k podpisu
- **name**
  poviný atribut k snazší identifikaci fotografie uživatelem
- **photo**
  Base64 záloha původní fotografie
- **publicKey**
  Base64 záloha použitého veřejného klíče
- **signature**
  SHA256withRSA podpis 2.5

Při ručním hledání dat v databázi od konkréního uživatele a nevidíme přímo jména ani emaily konkrétních uživatelů, ale pouze `User UID`. Tento identifikátor lze naleznout v seznamu uživatelů aplikace zobrazeném ve Firebase konzoli v sekci **Authentification**.

Zálohovací server a server hromadného podpisu

Komunikace mezi těmito dvěma subjekty spočívá ve zkratce v tom, že jakmile se na zálohovacím serveru vyskytnou data připravená k podpisu jsou odeslána přes HTTPS POST request a přijata na serveru hromadného podpisu, kde jsou uložena v interní databázi. Zpětná komunikace nastává v momentu, kdy server hromadného podpisu úspěšně vytvoří k danému záznamu časové razítko, tak přes Firebase Database REST API, viz 2.7, jsou poslány hodnoty `completed` (značící příznak, že byl záznam již podepsán) a `date` (reprezentující čas vytvoření hromadného podpisu) k přiřazení do databáze k náležitým záznamům, viz obrázek 4.3.

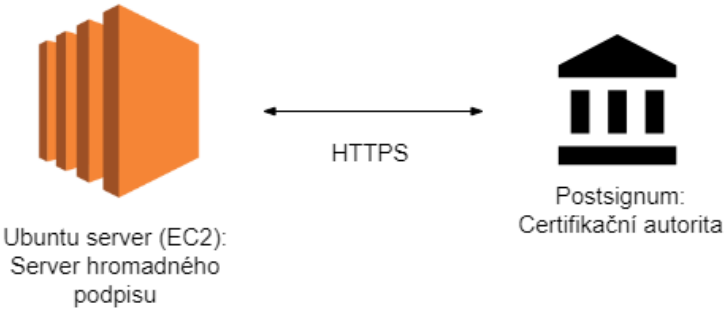


Obrázek 4.3: Schéma komunikace mezi Zálohovacím serverem a Serverem hromadného podpisu v rámci implementace Systému hromadného podpisu použité v demonstrační aplikaci InsuranceAgent.

Strukturu datového modelu v rámci zálohovacího serveru dále nebudu rozvádět jelikož byla již popsána v předchozí sekci. Co se týče struktury v interní databázi na serveru hromadného podpisu rozhodl jsem se implementovat svůj návrh databáze popsaný v sekci 3.1.

Server hromadného podpisu a certifikační autorita

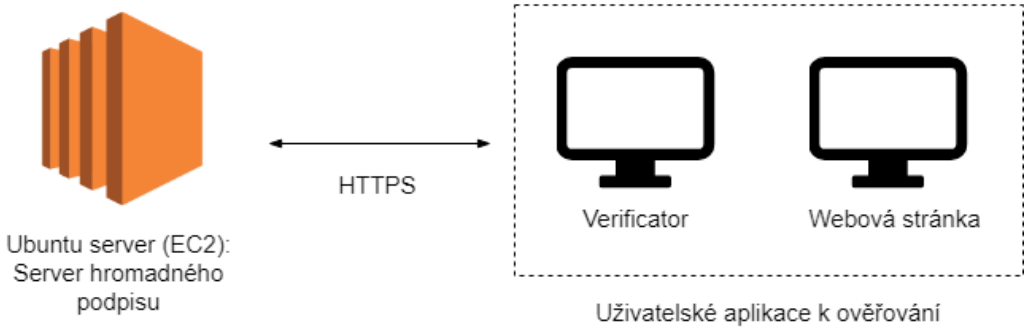
Komunikace zde probíhá skrz HTTPS připojení viz obrázek 4.4. Ze serveru hromadného podpisu se odešle tsq - „query“ soubor, na který obdrží odpověď ve tvaru tsr - „request“ souboru, který je následně uložen do databáze. Princip žádosti o časové razítko je podrobněji popsán v kapitole 2.3.



Obrázek 4.4: Schéma komunikace mezi Serverem hromadného podpisu a Certifikačí autoritou v rámci implementace Systému hromadného podpisu použité v demonstrační aplikaci InsuranceAgent.

Server hromadného podpisu a uživatelské aplikace

Uživatelské aplikace neboli webová stránka a Verifikator komunikují spolu se Serverem hromadného podpisu přes HTTPS případně HTTP komunikaci, viz obrázek 4.5.



Obrázek 4.5: Schéma komunikace mezi Serverem hromadného podpisu a uživatelskými aplikacemi v rámci implementace Systému hromadného podpisu použité v demonstrační aplikaci InsuranceAgent.

Pro úspěšnou HTTPS komunikaci je potřeba, aby uživatelé měli nainstalovaný certifikát *Serveru hromadného podpisu.*

Webová stránka

Webová stránka zobrazována uživateli je ve skutečnosti jen grafické rozhraní pro Verifikátor běžící na stejném stroji jako Server hromadného podpisu.

Verifikátor

Z verifikované fotografie je vytvořena SHA-256 komprimovaná hash hodnota, přes kterou je server dotázán. Pokud se v databázi daná hodnota znázornující fotografii nachází, pak je klientovi odeslán JSON 4.2 a dále jsou popsány náležitosti, které z nej vyplývají.

```
{
   "timestamp":"MIIFMTADAgEAMIIFKAYJKoZIhvcNAQcCoIIFGTCCBRUCAQMxCzAg...",
   "signatures":{
      "0":"o8w/fvDBKvGOOKn381Cwx0/CT0yzhBLkSuJhlSpx6rsitZKI4cf4vGWEL...",
      "1":"eDDCFw/geRjz2Rr3OxyMxEvTWfEMLBcMqFp+w6OYaPEckVmrWeqgLSZFV...",
      "2":"WF/N2IFbfCeZRGnQpqghEGwApFM23z/2pbHsLehknHkPbwUAVv5ZuRDS6..."
   },
   "signature-index":0
}
```

Výpis 4.2: JSON struktura pro přenos hromadného podpisu

Tento dotaz je možné provést i skrz webový prohlížeč, kde je uživateli zobrazen JSON nebo při neúspěchu zobrazena hláška `No match found`.

- **timestamp**
  Časové razítko ve formátu Base64
- **signatures**
  Jedná se o element obsahující atributy jednotlivých podpisů, které byly použity pro konkrétní skupinový podpis, zakódovaných v Base64 obsahující původní podpis fotografie.
- **signature-index**
  Jedná se o index, ke kterému podpisu v rámci `signatures` daná fotografie patří.

Elementy v položce `signatures` jsou číslovány podle pořadí v databázi, ovšem jsou přečíslovány od 0 (aby nebyla zveřejněna velikost databáze). Toto pořadí je velmi důležité dodržet, jelikož je stejné jako bylo při vytváření původního podepisovaného souboru.

4.3 Mobilní aplikace

V této sekci je popsán můj postup při vývoji demonstrační mobilní aplikace InsuranceAgent. Jednotlivé snímky obrazovek jsou uvedeny v příloze B.

Struktura zdrojového kódu

Zde jsou popsány nejdůležitější části mobilní aplikace a jejich význam.

- **AdapterItemInterface.kt**
  Rozhraní, přes které je zajištěno provolání metody k aktuálně zvolenému záznamu.
- **AndroidCamera.kt**
  Třída, která slouží jako prostředník k provolání metod z

`android.hardware.camera2` [9] knihovny, byla postavena na stejném principu jako oficiální demonstrační aplikace `CameraView` [12] a `Camera2Basic` [11], dále jsem značně čerpal z `Android Camera 2 api example tutorial` [30]

- **InsuranceTaskAdapter.kt**
  Toto je adaptér listu jednotlivých pojistných úkolů, který je zobrazován v rámci hlavní aktivity `MainActivity`.
- **InsuranceTask.kt**
  Třída reprezentující jednotlivé úkoly.
- **InsuranceTasks.kt**
  Singleton objekt, přes který se dá odkazovat na aktuální data v databázi a provolávat statické metody.
- **MainActivity.kt**
  Je třída, která spravuje aktivitu se seznamem jednotlivých pracovních úkolů.
- **SignInActivity.kt**
  Je třída zobrazující aktivitu, ve které si uživatel volí, pod kterým účtem se hodlá přihlásit.
- **TaskActivity.kt**
  V rámci této třídy je uživateli zobrazen konkrétní záznam pojistné události.
- **activity_main.xml**
  Grafické rozložení v hlavní aktivitě.
- **activity_sign_in.xml**
  Grafické rozložení v přihlašovací obrazovce.
- **alert_label_editor.xml**
  Vzhled dialogu při vytváření nového záznamu.
- **android_camera.xml**
  Definuje uživatelské rozhraní při snímání fotografií (částečně přejato z `Android Camera 2 api example tutorial` [30]).
- **single_item.xml**
  Znázorňující jeden záznam v `InsuranceTaskAdapter.kt`.
- **task_overview.xml**
  Detail specifického záznamu.

Implementace

Vzhledem k tomu, jak již bylo popsáno v 4.2 mobilní aplikace komunikuje se zálohovacím serverem, na kterém běží Firebase Realtime database. Synchronizaci těchto dat jsem zajistil pomocí nástrojů použitím oficiální Firebase knihovny z balíčku `firebase-database`[10]. Díky této knihovně je práce s JSON objekty v databázové struktuře velice zjednodušená, jelikož je možné je přímo načítat a přetypovávat na `InsuranceTask` třídy.

[9] https://developer.android.com/reference/android/hardware/camera2/package-summary
[10] 'com.google.firebase:firebase-database:16.1.0'

Přihlašování

Přihlašování do aplikace k ověření jednotlivých uživatelů je vyřešeno pomocí `OAuth 2.0` [11] implementovaného v rámci knihovny `firebase-auth`[12].

Práce se záznamy

Přes singletonovou třídu `InsuranceTasks.kt` přistupuji jako k objektu, přes který vždy získám aktuální seznam pracovních úkolů. V rámci `MainActivity.kt` aktivity jsem této vlastnosti využil a tento seznam zobrazuji přes `InsuranceTaskAdapter.kt`. Pro vytvoření nového záznamu je v této „hlavní“ aktivitě `FloatingActionButton`, přes které se vytváří nový záznam. Tento záznam, neboli úkol je možné otevřít, ze seznamu v `MainActivity.kt` a otevře se aktivita přes `TaskActivity.kt` s daty aktuálně otevřeného záznamu. V této aktivitě sice nejsou všechna data aktuální, ale při změně stavu dokončení podpisu se aktivita znovu překreslí. Toto překreslení probíhá i při vyfocení a podepsání fotografie.

Focení fotografií

Vyfocení fotografie je, jak už bylo zmíněno, zprostředkováno pomocí `android.hardware.camera2` uživateli je zobrazen náhled fotoaparátu a při stisknutí tlačítka „Capture“ je poslána žádost přes rozhraní knihovny, tak aby fotografie byla vytvořena se všemi náležitostmi, jako jsou typ, orientace, model fotoaparátu, výrobce. Případně šířka, výška, nebo aktuální GPS souřadnice. Vyfocená fotografie je uložena na uložiště v telefonu do složky `\InsuranceAgent\images\`. V tento moment je dále vytvořen i thumbnail fotografie uložen do složky `\InsuranceAgent\thumb\` k snadnějšímu načítání v seznamu `MainActivity.kt`. Toto řešení není sice úplně nejštastnější, ale vzhledem k tomu, že ve mne zprovozňování knihovny obsluhující hardwareový fotoaparát nevzbudilo jistotu, že vyfocená fotografie i přes nastavenou žádost pro vytvoření thumbnailu do exifu fotografie tento thumbnail vždy vytvoří, tak jsem se rozhodl vyřešit tento problém takto.

Při implementaci kamery jsem dále narazil na problém s rotací obrazovky, jenž zpravidla nastáva u telefonů značky Samsung při použití Camera2 API často, jelikož se k měnu dá na interntu najít mnoho diskuzí. Problém se mi podařilo vyřešit na základě článku [18].

Podepisování

Jakmile záznam obsahuje fotografii je možné jej podepsat privátním klíčem umístěným na uložišti mobilního telefonu ve složce `privatekey` ve formátu PKCS8, viz sekce „PKCS8“ v kapitole 2.5. Ověření, zda podpis proběhl správně je provedeno veřejným klíčem `publickey` tento veřejný klíč je ve formátu X509 popsané v kapitole 2.5. Pokud tyto klíče nejsou na zařízení nalezeny je uživateli zobrazena nabídka pro jejich vytvoření.

Fotografie ještě není uložena v databázi, jelikož je možné že bude přefocena, tudíž se čeká až uživatel vytvoří podpis pro záznam. Postup vytvoření podpisu je započat načtením soukromého a veřejného klíče spolu s fotografií. Pomocí knihovny `Signature` [34] je vytořen z fotografie podpis pomocí algoritmu SHA256withRSA 2.5. Potom je vytvořený podpis zkontrolován pomocí veřejného klíce. Pokud zpětné ověření proběhlo úspěšne je vytvořena SHA-256 komprimovaná hash reprezentace a te je ve hexadecimálním formátu spolu s fotografií (Base64), podpisem (Base64) a veřejným klíčem (Base64) uložena do databáze ve

[11] https://oauth.net/2/
[12] 'com.google.firebase:firebase-auth:16.2.1'

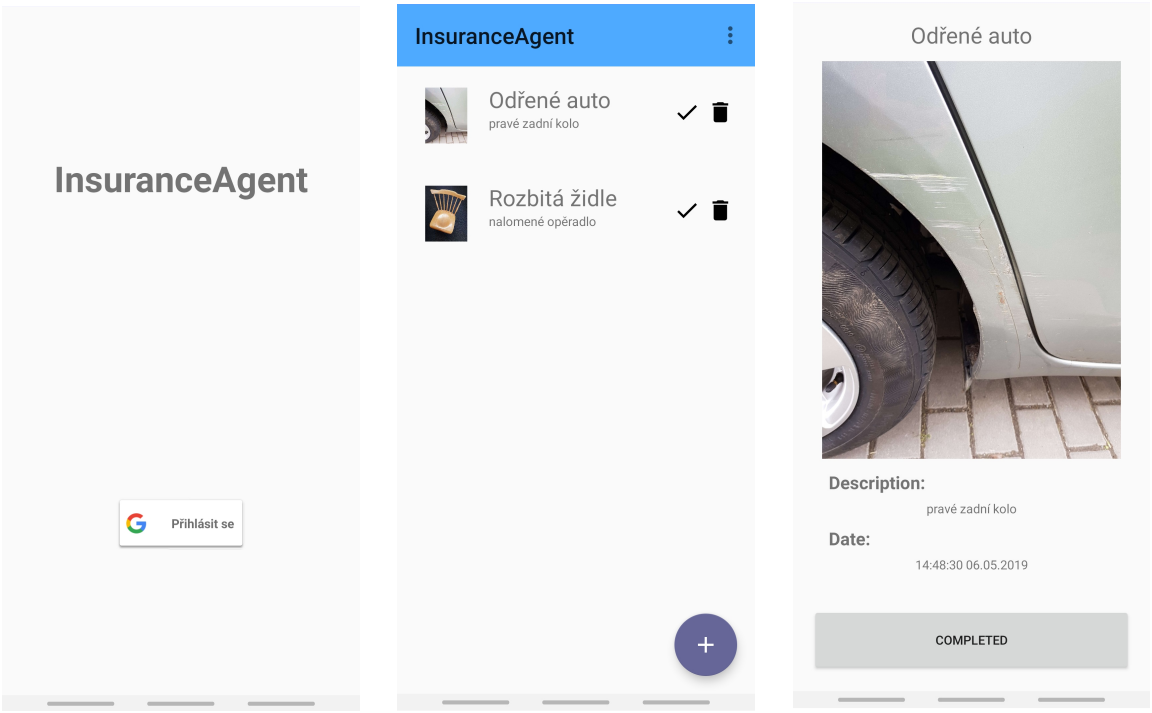
hexadecimálním formátu do náležitého záznamu. Toto odeslání veřejného klíče by mohlo být využito například při třízení dat na zálohovacím serveru, v momentě pokud by uživatel pracoval s více páry privátních a soukromých klíču.

Průběh vytvoření časového razítka

Jakmile je fotografie podepsána, je zobrazeno tlačítko s nápisem `Timestamp`, po kliknutí se vyvolá dialog pro potvrzení zda opravdu uživatel chce záznam odeslat k podpisu. Jakmile je tato akce vyvolána nastaví se v databázi příznak `locked`, který slouží k vyvolání odeslání dat do Serveru hromadného podpisu podrobněji popsáno v kapitolách 4.4 a 4.5. Jakmile je v databázi nastaven atribut `completed` a `date`, kde `completed` je příznak o tom, že byl záznam podepsán a `date` obsahuje čas, kdy bylo vytvořeno hromadné časové razítko.

Použití

V této sekci jsou pouze naznačeny možnosti aplikace a co jednotlivé obrazovky obsahují. Neslouží jako manuál ten se nachází v příloze C.



(a) Přihlašovací obrazovka (b) Menu (c) Záznam s fotografií

Obrázek 4.6: Příklady obrazovky mobilní aplikace **InsuranceAgent**

Buďto uživatel dostane přidělený soukromý a veřejný klíč, které uloží do mobilního zařízení. Nebo je možné při jejich absenci tyto klíče přímo na zařízení vygenerovat. Jelikož bez těchto klíčů by nefungovalo podepisování fotografií.

Při spuštění aplikace se otevře aktivita s názvem aplikace a tlačítkem přihlásit viz obrázek 4.6a. Po kliknutí na tlačítko přihlásit si uživatel vybere pod jakým účtem hodlá vystupovat a dostane se do další obrazovky, která obsahuje seznam jednotlivých pojišťovacích úkolů viz obrázek 4.6b.

U každého záznamu je zobrazena miniatura, titulek, description a tlačítko pro smazání. V případě, že již byl záznam podepsán je zde zobrazena značka „fajvky“. V případě rozkliknutí ikony koše, je uživateli nabídnuta dialogem nabídka k smazání nebo zrušení, či v případě nalezení existence fotografie je zde také možnost pro odstranění záznamu spolu s fotografiemi uloženými na lokálním uložišti mobilu. Pro vytváření nových záznamů se zde v právém dolním rohu nachází tlačítko s „plus“ symbolem. Při výběru záznamu ze seznamu, kliknutím na něj je otevřena detailnější obrazovka s informacemi k záznamu viz obrázek 4.6c. Uživatel po sejmutí fotografie spouští proces podpisu přes tlačítko `SIGN`. Po úspěšném podpisu je možné fotografii odeslat k časovému orazítkování tlačítkem `CREATE TIMESTAMP`

V horní části se nachází titulek, pod kterým je fotografie, níže je úmístěno textové pole pro možnost vypnění podrobnějšího popisu a položka `Date:`, ve které se po úspěšném dokončení podpisu zobrazí čas podpisu. A na konci obrazovky se zobrazí `COMPLETED`.

Pokud uživatel smaže fotografii, která byla časově orazítkována může si znovu stánout kliknutím na tlačítko `COMPLETED (DOWNLOAD PHOTO))`. Toto tlačítko se zobrazí v spodní části obrazovky místo `COMPLETED`.

4.4 Zálohovací server

V této sekci je podrobně popsáno jak jsem využil Firebase Cloud Functions, viz kapitolu 2.7, jako triggeru nad databází.

Implementace

K propojení skriptu s databází jsem použíl dvě oficiální knihovny pro práci s Firebase Realtime Database, jenž jsou firebase-admin ve verzi 7.0.0 [21] a firebase-functions ve verzi 2.2.0 [24]. Vzhledem k tomu, že funkce v nasazeném sktiptu `index.js` běží na Firebase serverech přímo v instanci InsuranceAgent projektum, bylo logické použití přístupu do databáze pomocí těchto oficiálních knihoven.

Na Firebase server je nasazena funkce, která se chová jako trigger nad Firebase Realtime Database a v momentě, kdy je v záznamu příznak „locked“, neboli uzavřen, přepnut z otevřeného na uzavřený, je volána pomocná funkce pro odeslání dat do Serveru hromadného podpisu. Pro tuto HTTPS komunikaci jsem se rozhodl využít knihovnu Request-promise ve verzi 4.2.4 [49]. Konkrétní odesílaná data neobsahují veškeré informace k záznamu, ale pouze atributy potřebné pro vytvoření časového razítka a pro zpětnou referenci do databáze, tak aby Server hromadného podpisu mohl dát zpětně vědět, které záznamy byly časově orazítkovány. Jedná se o elementy: `compressed`, `key`, `signature` a `user`, podrobně popsané v kapitole 4.2.

Použití

I přesto, že se jedná o komunikační uzel mezi jednotlivými mobilními zařízeními a Serverem hromadného podpisu, další neodmyslitelnou funkcí je zálohování. Pokud uživatel telefon ztratí jsou zde úložená data od poslední synchronizace databáze. Jednotlivé soubory jsou ovšem zakódované do Base64 formátu, což značně znepřístupňuje využití přístupu do této databáze pro konkrétní uživatele, zkušenější správce systému, by ovšem s obnovou dat neměl mít žádný problém. V momentě, kdy uživatel na svém mobilním telefonu, záznam který byl odeslaný k časovému orazítkování smaže, není z databáze na zálohovacím serveru odstraněn, je pouze skryt. U jednotlivých uživatelů se tedy v zálohovacím serveru ukládají

data všech vytvořených záznamu, ale na telefonu každého uživatele jsou zobrazeny pouze neskryté (aktivní) záznamy.

4.5 Server hromadného podpisu

V této kapitole je popsána struktura, implementace a použití Serveru hromadného podpisu zapojeném v Systému hromadného podpisu.

Struktura zdrojového kódu

Seznam skriptů, ze kterých je Server hromadného podpisu sestrojen:

- **initdb.py**
  Slouží k inicializaci interní databáze.
- **printdb.py**
  Vytiskne aktuální stav databáze na výstup.
- **response.py**
  Modul, přes který se dá nastavit způsob oznámení o úspěšně vytvořeném časovém razítku do Zálohovacímu serveru.
- **server.py**
  Hlavní skript, ve kterém je implementovan server očekávající POST a GET HTTPS a HTTP requesty, na které reaguje.
- **signer.py**
  Skript vytvářející hromadný podpis.
- **verificator.py**
  Stejná verze jako je uživatelský skript Verifikátor podrobně popsána v kapitole 4.6, která je ovšem provolávána při dotazech o verifikaci fotografií z webové stránky.

Implementace

Databáze na Serveru hromadného podpisu je určena k tomu, že nad ní v jeden moment běží pouze dva skripty a to `server.py` a `signer.py`. Struktura dat co jsou v databázi uložena je totožná s předlohou návrhu, který je podrobě popsán v kapitole 3.1.

Server

Serverový skript `server.py` obsluhuje konkrétní port, nad kterým je spuštěn. Poslouchá jednotlivé POST a GET HTTP a HTTPS requesty viz tabulku 4.1, jednotlivé atributy, které nejsou brány v potaz jsou ponechány prázdné. Dále hodnota `/SHA-256` odpovídá hashové reprezentaci dotazované fotografie. Jednotlivé případy jsou posány dále v této kapitole.

Tabulka 4.1: Rozhraní Serveru hromadného podpisu

Request metoda	Path	Content-Type	Odpověd
GET	/	-	html stránka
	/SHA-256	-	JSON
POST	/	multipart/form-data	html stránka
	/	application/json	„success“

Při přijetí dat z Firebase se rozpozná, že se jedná o POST s nálezitými hlavičkami a rozparsuje se JSON datová struktura, pokud vše odpovídá, tak jsou nově přijatá data uložena do databáze a v odpovědi na dotaz o úspěšném uložení je odeslán textový řetězec „success“ typu `Content-type: text/html`.

V případě, že se jedná o dotaz z webové stránky, neboli že se jedná o `Content-type: multipart/form-data`, jsou příjmány soubory fotografie a volitelně veřejného klíče autora, které jsou uloženy v dočasném uložišti na serveru, pomocí knihovny `tempfile` [57]. Nad těmito soubory je volán script `verificator.py` a na základě návratového kódu je uživateli zobrazená odpovídající **html** stránka.

Pokud se jedná o GET request, kde **path** requestu odpovídá hlavnímu uzlu „/“, tak je uživateli odeslána webová stránka `index.html` podrobněji v kapitole 4.6. Když se ovšem v **path** nachází textový řetězec je tento dotaz považován za `compressed` reprezentaci fotografie odeslanou z `verificator.py` a na základě této hodnoty je prohledána databáze, zda se k této fotce nevyskytuje podpis, jestliže se vyskytuje je skriptu zpět odeslán JSON s výsledkem viz sekce **Server hromadného podpisu a Verifikátor** v kapitole 4.2.

Signer

Podepisovací skript `signer.py` vždy vykoná úlohu vytvoření hromadného časového razítka a pak je proces na 24 hodin uspán.

Postup je takový, že se nejprve v lokální databázi vyhledají záznamy v tabulce `files`, které doposud nepatří do žádného hromadného podpisu z tabulky `packages`. Skutečnost, že jestě nebyly časově orazítkovány je určena na základě vyplněné, či nevyplněné relační hodnoty `packaga` v rámci jednotlivých záznamů v tabulce `files`. Jak už bylo na začátku této kapitoly uvedeno, tak datové rozložení je podrobněji popsáno v kapitole 3.1.

Načtené podpisy `signatures` jsou konkatenovány do jednoho souboru ve formátu `.txt`. Jednotlivé podpisy jsou umístěny vždy na zvláštní řádek, který je zakončen `n` symbolem a to i včetně posledního řádku. Pořadí podpisů je předurčeno pořadím v databázi na základě seřezení podle indexu `id`.

Jakmile je soubor `.txt` s podpisy připraven, je z nej vytvořen soubor **`.tsq`**, neboli `query` soubor, jenž bude časově orazítkován. Vytvoření `query` souboru probíhá pomocí knihovny OpenSSL [43] z této knihovny je pro vytváření časových razítek použit nástroj `Time Stamping Authority tool (client/server)`, který je implementován podle **RFC 3161** specifikace. [44].

Jakmile je `query` soubor po vytvoření odeslán pomocí nástroje curl ve verzi 7.58.0 [54]. Jedná se o HTTPS, kde je v headeru uveden `Content-Type: aplication/timestamp-query` a odeslání binírních dat `.tsq` souboru, poté příjde odpověd s hlavičkou obsahující `Content-Type: application/timestamp-reply` a binárními daty `reply` neboli `.tsr` souboru [7]. Tento soubor je již samotným časovým razítkem, ještě před uložením a přiřazením v databázi proběhne kontrola přes `openssl` [44]

a to buďto pomocí příkazu `-verify` a nebo pomocí zobrazení v případě použití Postsignum certifikační autority je třeba použít `-reply`. Problém je v tom, že časové razítko od Postsinum obsahuje atributový certifikát a knihovna OpenSSL s ním nedokáže pracovat [32] při verifikaci, přes onen zmíněný příkaz `-verify`. [41]

Význam `query` a `request` souborů je popsán v sekci **Proces získání časového razítka** v kapitole 2.3.

Pokud tedy vše proběhlo úspěšně je v databázi uložena vazba jednotlivých `files` záznamů k nově vytvořenému záznamu v `package`, do kterého se uloží soubor časového razítka `.tsr` v Base64 formátu.

Pokud nastane kdekoliv v procesu pořizování chyba například při chybné komunikaci HTTPS s certifikační autoritou, jelikož nelze zaručit, že bude vždy dostupná, tak se databáze v takovémto případě neupraví, vrátí do původního stavu a nepodepsané fotografie se podepisovací skript pokusí znovu časově orazítkovat v dalším průběhu.

Poslední krok, který po vykonání podpisu proběhne je provolání `response.py` skriptu, aby dal vědět Firebase serveru, které záznamy byly úspěšně odeslány.

Response

Jakmile je `signature` patřící ve Firebase databázi na zálohovacím serveru podepsán. Pak je potřeba tento Firebase server notifikovat a vzniklých změnách a k jednotlivým časově orazítkovaným záznamům přidat položky `completed` a `date`, kde `completed` značí příznak, že byl podpis `signature` časově označen a hodnota `date` je čas, kdy se na Serveru hromadného podpisu podařilo úspěšně vytvořit a ověřit vytvořené časové razítko.

Pro komunikaci jsem si vybral využít Firebase API a tentokrát jelikož se nacházíme v jazyce Python jsem zvolil použit knihovnu Pyrebase [58]. Přes vygenerovaný administrátorský klíč z webového rozhraní Firebase [13] je nastavení přístupu do databáze velice intuitivní a chytře vymyšlené.

Použití

Nejprve před prvním spuštěním skriptů `server.py` a `signer.py` je potřeba spustit skript `initdb.py`, který smaže veškeré záznamy v tabulkách **files** a **packages**, pokud nějaké existovaly a vytvoří tyto tabulky nové a prázdné. Jakmile je databáze připravena, je možné libovolně spouštět „serverový“ a „podepisovací“ skript, tyto skripty jsou na sobě nezávislé. Proto pokud běží „server“ není nutné nově příchozí záznamy časově orazítkovávat, ovšem z hlediska logiky aplikace je to vysoce vhodné. Pokud by někdo chtěl tento zdrojový kód využít ve své aplikaci je nutné upravit `response.py`, `verificator.py` a `signer.py`.

Buďto přeprogramovat `response.py` a místo Pyrebase použít například některou HTTPS knihovu pro notifikaci serveru a na Zálohovacím serveru naprogramovat listener.

4.6 Verifikátor a webová stránka

Jak už bylo v kapitole 3.2 zmíněno, tak Verifikátor je oproti webovému rozhraní mnohem podrobnější a umožnuje uživateli export dat pro přenos a možnosti ověření v jiném systému. Webová stránka má za úkol uživateli hlavně ukázat, zda byla fotografie úspěšně podepsána, či ověřit kdo byl autorem. Podle tohoto návrhu jsem také postupoval při implementaci.

[13] https://console.firebase.google.com/

Struktura zdrojového kódu

Verifikátor se skládá pouze z jednoho souboru `verificator.py` a webová stránka se skládá z Html souborů obsahující jednotlivé stránky.

Implementace

V této sekci je popsána implementace uživatelských programů sloužících k zpětnému ověření, zda byla fotografie správně časově orazítkována. Webová stránka je ve skutečnosti především grafické rozhraní pro Verifikátor. Vzhledem k tomu, že webová stránka je odesílána uživateli na základě dotazů na server, kde běží Server hromadného podpisu, je v této kapitole popsáno pouze sestrojení dané webové stránky a je zde vynechána komunikace. Pokud se chcete dočíst podrobněji, jak je naprogramována komunikace přes webovou stránku, tak si o ní můžete přečíst v kapitole Serveru hromadného podpisu 4.5.

Verifikátor

Z vložené fotografie je vytvořena SHA-256 komprimovaná hash reprezentace a na základě ní je dotázán Server hromadného podpisu přes HTTPS/HTTP **GET** request, kde se ve zkratce na základě této hash reprezentace prohledá tabulka `files` zda k fotografii neexistuje záznam. Pokud existuje, tak se zkontroluje jestli je vyplněn sloupeček `package` referencující záznam z tabulky `packages`. Existence této vazby značí existenci časového razítka k dané fotografii. Pokud je tedy nalezena jsou časové razítko spolu se seznamem originálních podpisů a indexem podpisu, které k fotografii náleží, odeslány v JSON formáto zpět uživateli na počítač, ze kterého byl onen **GET** dotaz položen. Podněji popsáno v kapitole *Popis komunikace a datový model* 4.2 v sekci *Server hromadného podpisu a Verifikátor*.

Na základě dat vyčtených z odpovědi se dekóduje z Base64 formátu časové razítko na soubor a ze seznamu časově orazítkovaných podpisu `signatures` je zpětně vytvořen původní podepsaný soubor. Je zde potřeba dbát na fakt, že jednotlivé podpisy jsou seřazeny podle indexu v databázi a že za každým podpisem následuje znak nového řádku.

Tyto vytvořené soubory jsou znovu uloženy do formátů `.tsq` a `.tsr` a je použito stejné vytvoření `query` souboru a následné ověření přes OpenSSL[44] jako ve skriptu `signer.py` který běží na Serveru hromadného podpisu 4.5 a daný hromadný podpis vytvořil.

Při ověřování autora je z podpisů vybrán podpis `signature` patřící k fotografii, na základě indexu `signature-index`. Tento podpis je ověřen s původními daty, neboli s SHA256 kompresí na základě vloženého veřejného klíče.

Vzhledem k tomu, že má uživatel na výběr zvolit pro vytvoření výstupní prefix pro pojmenování souboru dat z aktuálního ověřování, tak při jeho vynechání je potřeba vytvořit dočasné soubory, tak aby bylo možné použít externí knihovnu OpenSSL k ověření. Pro vytvoření techto dočasných souborů používám knihovnu pro dočasné `tempfile` [57].

Webová stránka

Zpráva v jednotlivých webových stránkách po odeslaní dotazu k fotografii je vložena do odpovídajícího `.html` souboru k zobrazení. A to tak, že proběhne načtení `.html` souboru a nahrazení hodnoty `OUTPUTTEXTHOLDER` správným popiskem, vhodním k aktuální situaci. Toto nahrazování a implementace veškeré komunikace, jak již bylo v úvodu této kapitolym zmíněno, probíhá na Serveru hromadného podpisu.

Použití

V této sekci je zevrubně popsáno jak uživatel pracuje s Verifikátorem a Webovou stránkou. Nejedná se ovšem o manuál ten je obsažen v příloze C.

Verifikátor

Nástroj Verifikátor je spustitelný z příkazové řádky a k jeho správné funkčnosti je zapotřebí mít na systému nainstalovaný nástroj `OpenSSL ts` [44]. Při jeho spuštění má uživatel na výběr použít následující parametry:

- **inputfile**
  Fotografie k ověření.
- **authorcert**
  Veřejný certifikát autora, o kterém je zjišťováno zda se jedná o autora podpisu.
- **details**
  Slouží k povolení zobrazení detailních výpisů na výstup.
- **outputfile**
  Prefix názvu pro vytvoření souborů k exportu.

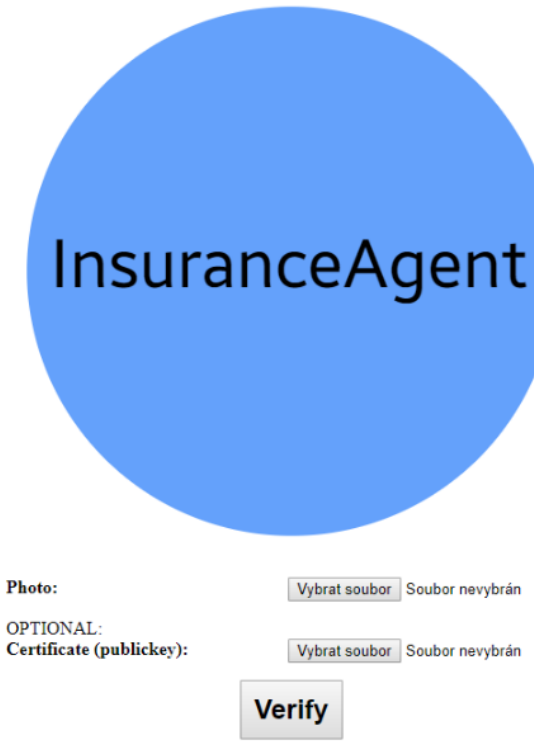
Jak je v předchozí sekci naznačeno k ověřované fotografii je po přenosu ze Serveru hromadného podpisu na uživatelském počítači časové razítko znovu vytvořeno a ověřeno vůči reprodukovanému původnímu souboru s podpisy, který byl podepsán. Uživatel si určuje, prefix pod kterým budou uloženy do souborového systému na jeho počítači. Toto může být zapotřebí v případě nutnosti exportu dat. Při použití parametru `outputfile` se tedy vytvoří následující soubory:

- `.txt`
- `.tsq`
- `.tsr`
- `.signature`

Jak už bylo v implementační sekci popsáno `.txt` obsahuje `signatures` a znázorňuje původní soubor dat. Query `.tsq` soubor je formát souboru ve kterém byla dotazována certifikační autorita. Reply `.tsr` je soubor, který přišel jako odpověd, neboli samotné časové razítko hromadného podpisu. Podpis `.signature` je soubor, který obsahuje řádek z `.txt` souboru ovšem ne už ve formě Base64 reprezentace, ale jako celý původní řetezec dat, vytvořený z fotografie přes privátní klíč na mobilním zařízení. Všechny soubory mimo soubor `.signature` jsou vytvářeny při každém běhu programu, soubor s podpisem je vytvářen pouze při práci s veřejným klíčem uživatele, neboli použitím parametru `authorcert`.

Webová stránka

Obrazovky webových stránek jsou k dispozici k nahlédnutí v kompletním seznamu v příloze B. Pro ilustrační účel zde vkládám úvodní obrazovku, viz obrázek 4.7.



Obrázek 4.7: Úvodní obrazovka webové stránky zobrazená uživateli přes webový prohlížeč.

Při GET HTTP/HTTPS dotazu bez specifikované cesty, neboli když `path` = / na Server hromadného podpisu je uživateli zobrazena úvodní `index.html` stránka. Na této stránce se nachází obrázek s logem aplikace InsuranceAgent, dále dvě výběrové tlačítka spolu s popisky, které slouží pro zvolení fotografie k ověření a případně přiložení veřejného klíče autora a tlačítko „Verify“ pro spuštění ověření vybraných souborů. Uživatel vybere na počítači fotografii, nebo navíc volitelně přídá i zmíněný veřejný klíč, a zvolené soubory odešle na Server hromadného podpisu. Ten na základě volání **Verifikátoru** umístěném na stejném stroji, rozhodne zda odešle uživateli jednu z následujících tří stránek.

Při úspěšně ověřeném podpisu, neboli pokud byl vložen a úspěšně ověřen veřejný klíč, se uživateli pošle `success.html`.

V případě, že časové razítko existuje a odpovídá, ale vložený soubor veřejného klíče je chybný je uživateli zobrazena stránka `warning.html` s upozorněním.

Pokud kdekoliv v průběhu podpisu nastane chyba je zobrazena `error.html` webová stránka.

Vzhledem k tomu, že se jedná o webovou stránku, tak uživatel může ověřování provádět rovnou z mobilního zařízení.

Kapitola 5

Testování, využití v praxi a další možnosti rozvoje

Nedílnou součástí životního cyklu softwaru je také testování. Pro správné otestování systému určeném k vytváření důkazního materiálu toto pravidlo platí dvojnásob. Je potřeba brát v potaz nejenom správnou funkčnost z hlediska grafického rozhraní uživatelů, ale především na zachování integrity dat. Pokud by mohlo dojít k jakémukoliv úniku citlivých dat nebo zakázané manipulace s daty v databázi, tak se nedá výslednému řešení důvěřovat.

5.1 Testování

V konkrétním případě této bakalářské práce, kde se jedná o demonstrační aplikaci, která se zabývá vytvářením a zpětným ověřováním hromadných časových razítek, je tedy potřeba, aby byla zajištěna integrita dat při pořizování fotografií spolu se všemi náležitostmi vedlejších souborů určených k ověření.

Jednotliví uživatelé vlastní soukromé a veřejné klíče, pokud by došlo k úniku privátního klíče, tak je sice v otázce autenticita zdroje jím pořízených záznamů, ale ostatní uživatelé by neměli být jakkoliv dotknuti. Jakmile by ovšem mohlo dojít k chybám přenesených záznamů a možnosti napadení komunikace nebo přímo některé databáze, tak by se aplikace nedala považovat za důvěryhodnou.

Průběh testování spočíval zejména v tom, že jsem *Systém hromadného podpisu* programoval postupně od aplikace pro mobilní zařízení **InsuranceAgent**, přes *zálohovací server* běžícím na Firebase, poté *Server hromadného podpisu*, až po komunikaci s certifikační autoritou *Postsignum*. Jednotlivé změny v kódu a nově vytvořený kód byl proto prvotně otestován již krátce po svém vytvoření. Testování jsem rozdělil do čtyř kapitol odpovídající jednotlivým částem systému hromadného podpisu a komunikací s externí certifikační autoritou.

Mobilní aplikace

InsuranceAgent mobilní aplikace se byla vyvinuta pouze jako demonstrační uživatelská aplikace se značně jednoduchým uživatelským rozhraním. Nekladl jsem proto na testování mobilní aplikace tak velký důraz, jako by si při implementaci do reálného systému zasloužila. Správnou funkčnost aplikace jsem si ověřil na zařízeních vypsaných v tabulce 5.1 a u všech se mi podařilo bez problémů vytvořit, podepsat a časově orazítkovat fotografie.

Tabulka 5.1: Testování, na kterých byla ověřena správná funkčnost aplikace

Zařízení	Android verze
Samsung Galaxy S8	9.0
Samsung Tab A (2016)	8.1.0
Xiaomi Redmi Note 3	6.0.1
Panasonic Toughpad FZ-X1	5.1.1
ASUS ZenPad 10 (Z300CL)	5.0.1

Jsem si vědom, že se v aplikaci vyskytují malé chyby, ale aplikace jako taková funguje a svůj demonstrační účel plní bez problémů. Vzhledem k tomu, že mobilní aplikace má být použita v konkrétní implementaci serveru hromadného podpisu, tak v případu demonstační mobilní aplikace **InsuranceAgent**, kde se jedná pouze o hypotetickou pojišťovací instituci, nebyl příliš kladen důraz na uživatelské testování. Pro nasazení do reálného systému by bylo vhodné aplikaci předělat vůči pořadavkům konkrétního zákazníka a provést testování nad větším vzorkem uživatelů (jeho zaměstnanců), kde by se nalezené nedostatky dle přání daly vyřešit a některé vlastnosti a chování předělat, odstranit nebo naopak doplnit další funkce.

Zálohovací server

Jedná se o první část systému, ve které jsem se musel zabývat zapojením komunikace přes veřejnou sít. Tento problém, jak už bylo zmíněno, jsem vyřešil přes Firebase API konkrétně synchronizací Firebase Realtime Database 2.7, kteryžto je značně otestovaný komunitou využívající tyto funkce. Komunikace s mobilním klientem, tedy nebylo zapotřebí tolik testovat.

Oproti tomu při testování komunikace se serverem hromadného podpisu jsem narazil na chybu, která nastane při nezdařeném odeslání zvoleného záznamu (podpisu fotografie) k časovému orazítkování. Pokud tato komunikace selže, tak neexistuje žádný podpůrný mechanismus v rámci Firebase Cloud Functions, který by se neodeslaný záznam znovu pokusil odeslat. Na tuto chybu jsem narazil při procházení logových hlášek v momentě kdy jsem zjistil, že serverová instance v EC2 2.7 **AWS** 2.7 Server hromadného podpisu mění svoji DNS a IP adresu po každém vypnutí a zapnutí. Jako řešení by místo **EC2-Classic** bylo možné použít **EC2-VPC** [56] a tím tento problém vyřešit.

Server hromadného podpisu

Testování v této části systému probíhalo zejména nejenom při implementaci této části samotné, ale v rámci jednotlivých iterací jakékoliv dodatečné změny jiné části systému, jelikož se jedná o „srdce“ systému. Co se týče úspěšnosti vytvářených časových razítek byla tato část při mém testování bezchybná, jelikož se ovšem nejednalo o implementaci v reálném systému, tak nebylo možné i za použití více demonstačních uživatelů systém dostatečně otestovat. Tudíž v případě mých dostupných prostředků, jakožto časových, tak i finančních jsem byl nucen po vlastním zevrubném testování tuto část systému prohlásit za funkční pro konkrétní implementaci a potřebu funkčnosti v rámci této demonstrační aplikace.

Certifikační autorita

Při zapojování komunikace s certifikační autoritou **Postsignum** [1] jsem narazil na zásadní problém, že příkaz `-verify` z knihovny OpenSSL TS[44], který je v případě jakékoliv jiné mnou vyzkoušené certifikační autority používající protokol **RFC316** [7] fungoval, tak v případě **Postsignum** vůbec nefungoval. Tento problém se mi po dlouhém hledání řešení na internetu podařilo vyřešit v konzultaci s technickou podporou [2]. Vzhledem k tomu, že mimo kořenový certifikát Postsignum přidává do časového razítka rovněž, atributový certifikát, se kterým knihovna OpenSSL nebyla schopná pracovat, bylo mi porazeno pro ověření použít příkaz `-reply` a nebo vytvořit `query` soubor bez žádosti o tento atributový certifikát. Další testování jsem, co se týče certifikační autority neprováděl, jelikož od zapojení jsem provedl desítky časových razítek a v žádném případě zde nedošlo k chybě.

5.2 Využití v praxi

Nejprve bych chtěl pohovořit o možném využití *Systému hromadného podpisu* v praxi a potom stručně shrnout to, jak by se dala moje vytvořená demonstrační aplikace využít. Kde při jejím používání figuruje uživatel, jakožto v rámci **InsuranceAgent** mobilní aplikace, ale také webového rozhraní i při použití nástroje **Verifikátor**.

Systém hromadného podpisu

Algoritmus vytvářející hromadné podpisy pracuje v demonstračním řešení přímo s podpisy daných fotografií, tudíž by bylo velice jednoduché systém využít například pro časové orazítkovávání podpisů kterýchkoliv jiných elektronických dokumentů v různých systémech, kde je potřeba vytvářet časové razítka a je žádáno o ně u certifikační autority. Server může být využit jako prostředník mezi tímto stávajícím systémem a samotnou autoritou, a začít vytvářet časové razítka hromadně.

Systém hromadného podpisu, jako takový může být v některé z jeho částí nahrazen za jinou entitu a k navázání s ostatními částmi by nemělo být nějak zásadně náročné (samozřejmě záleží na zvolených technologiích), zároveň by bylo možné systém zkrátit a místo konkrétních mobilních zařízení a zálohovacího serveru komunikovat se **Serverem hromadného podpisu** přímo v jednotlivých zařízeních. Systém je tedy dostatečně modulární a například **Server hromadného podpisu** by mohl komunikovat s více zálohovacími servery zároveň, tudíž by se mohla vytvořit určitá certifikační „podautorita“, která by zvládla vytvářet časové razítka ke stovkám podpisu zároveň z několika zdrojů. Toto seskupení zdrojů si můžete představit jako skupinu firem, které by zároveň odesílaly podpisy a časově orazítkovávávaly data na jednom systému současně a tím mohly ještě více ušetřit. Princip vytváření a ověřování hromadných časových razítek nepracuje s původními daty, ale pouze s jednotlivými podpisy. Na serveru se ani nějak neukládají data o veřejném klíči, tudíž je vše dostatečně zanonymizováno.

V databázi na **Serveru hromadného podpisu** se nachází čtyři podstatné údaje, které na něj byly odeslány ze **Zálohovacího serveru**. První dva údaje `user` a `task` referencují do původní databáze Firebase a v rámci mé demonstační aplikace jsou použity ke zpětné notifikaci, ovšem v reimplementaci by zde například bylo možné uvést navíc IP adresu, ze kterého data přišly a posílat hodnoty `user` a `task` přímo v HTTPS POST requestu na

[1] http://www.postsignum.cz/index.php
[2] servicedesk@cpost.cz

originální „zálohovací server“. Zbylé dvě hodnoty, které jsou na serveru v databázi uloženy, které se dají logicky propojit s původními daty jsou `SHA-256` komprimovaná reprezentace a podpis `signature SHA256withRSA` (viz sekce Šifrování v kapitole 2.5) obě byly vytvořeny jak z názvy vypovída přes SHA256 (viz sekce Hashování kapitola 2.5). V tomto obráceném směru se ovšem z těchto dat původní data získat nedá, jelikož se nejedná o šifrovací algoritmy, ale hashovací algoritmy.

Princip využití demonstrační aplikace

Konkrétní kroky pro práci s s aplikací jsou popsány v manuálech viz příloha C. Jednotliví uživatelé, zejména pojišťovací agenti, jelikož pro ty je aplikace zaměřena, si v mobilní aplikaci vyfotografují fotografii, která je následně časově orazítkována, tak aby bylo v budoucnu možné dokázat její existence v danou dobu podpisu. Prvotní nastavení je poněkud složitější, jelikož je potřeba vygenerovat soukromý a veřejný klíč, aby uživatel mohl vytvářet fotografie. Nejedná se o žádný složitý proces, který by nedokázal inteligentný člověk za pomoci návodu vykonat. V rámci mobilní aplikace jsou přihlašováni uživatelé pomocí Google účtu, přes který je potvrzována jejich totožnost. V aplikaci mají seznam, zda byly fotografie již časově orazítkovány. Pokud si chtějí ověřit některou z fotografií, můžou ji například stáhnout z mobilního uložiště na počítač a tam si dodatečně zkontrolovat, zda byla opravdu podepsána. Pokud by bylo potřeba exportovat hromadně vytvořený podpis, může uživatel použít program **Verifikátor**, přes kterou vygeneruje veškeré potřebné soubory na základě původní fotografie a veřejného certifikátu, kterým byla daná fotografie podepsána.

5.3 Zhodnocení

V rámci minulé sekce jsem již zmínil několik objevených problémů odhalených při testování. Jako další problém, který je nutné zmínit je to, že vytvořená demonstrační implementace Systému vytvářející hromadný podpis má špatně nastavený cerfitikát pro HTTPS komunikaci. Tento problém, jenž nastává při komunikaci mezi instancí Firebase a Serverem hromadného podpisu. Bylo problém vytvořený certifikát na serveru `Firebase` nainstalovat, jelikož se jednálo o vlastnoručně podepsaný certifikát, který jsem si já sám vytvořil, bez použití externí autority pro vydání SSL certifikátu. Tento problém nastává v komunikaci pouze ve směru z Firebase do Serveru hromadného podpisu, a to konkrétně při odesílání záznamu. Jednoduché řešení by bylo řešení požádat o vystavení tohoto certifikátu například na doméně SSLMate [3], ale na to by se museli vynaložit další finanční prostředky, což jsem nepovažoval pro demonstrační účel mého řešení v této bakalářské práci za nezbytné. Alternativní řešení se nabízí nastavením „whitelistu“ na obou serverech a navzájem považovat tuto komunikaci za jedinou povolenou. V implemetaci, která by ovšem fungovala nad reálnými daty ve skutečně nasazeném systému, by bylo například možné tyto servery (Server hromadného podpisu a Zálohovací server) sjednotit, aby fungovaly v rámci jedné lokální sítě. Při použití jiného externího serveru nebo zachování použití Firebase by tedy bylo potřeba striktně nastavit pravidla z jaké IP adresy bude možné záznamy k časovému orazítkování přijímat.

Jako další věc považuji za nutné uvést, že jsem zvýšil minimální požadovanou verzi Android z 4.0.4 (SDK 16) na Android 5.0 (SDK 21), pro použití knihovny [12] obsluhující fotoaparát, jelikož v starší verzi nastávala chyba při zobrazovaní náhledu kamery.

[3] https://sslmate.com/

Vzhledem k neustálému testování jsem přešel na **FreeTSA** [4], jelikož jsem měl na školním účtu **Postsignum** [5] omezený počet časových razítek.

V rámci **Firebase** jsem přešel ze základního platebního plánu **Spark** na plán **Blaze**, jelikož základní nenabízí možnost komunikace s externími servery v metodách běžících přes **Firebase Cloud Functions** [4] [5].

Použitelnost mnou vytvořeného Systému hromadného podpisu bych dle svého názoru zhodnotil za úspěšně dosaženou. Systém je připraven na to být implementován v reálném systému a dle mého usouzení ho považuji za důvěryhodný. I kdyby se použil jen v rámci velkého korporátu, pro správu souboru, tak časové razítkování přímo přes autoritu by bylo finančně nesmyslné. Vzhledem k tomu, že při použití některých certifikačních autorit pro vystavení kvalifikovaného časového razítka se cena při využití paušálního ceníku pohybuje okolo 1 Kč až 4 Kč v závislosti na objemu předplacených podpisů.

Ovšem při reálném použití tento paušál není naplno využit a zákazník stále platí za objednaný počet. Tudíž reálně při využití jednoho podpisu denně, neboli kolem třiceti podpisů mesíčně se může cena pohybovat při měsíčním zůčtování až kolem 400 Kč viz tabulky 5.3 a 5.2. Přepočteno na počet razítek vychází cena jednoho přibližně na 13 Kč.

Nebo například při využití pěti časových razítkách denně, tedy přibližně stopadesáti mesíčně se cena v horším případě může vyšplhat až k 1000 Kč viz tabulku 5.2. Po přepočtu za razítko je tedy cena přibližně 7 Kč za razítko.

Tabulka 5.2: Postsignum Ceník časových razítek[46]

Počet	Cena [Kč]	Cena jednoho razítka [Kč]
30	121	4,03
100	363	3,63
350	1028,5	2,94
1 000	2 420	2,42
3 500	6 050	1,73
10 000	15 125	1,51
35 000	42 350	1,21
100 000	90 750	0,91
250 000	181 500	0,73
více	242 000	-

Tabulka 5.3: První certifikační autorita, A.S. Ceník časových razítek[48]

Počet	Cena [Kč]	Cena jednoho razítka [Kč]
100	423,5	4,24
300	726	2,42
500	907,5	1,82

[4] https://www.freetsa.org/index_en.php
[5] http://www.postsignum.cz/index.php

Z nabytých znalostí z tabulek 5.2 a 5.3 vyvozuji, že při použití mého *Systému hromadného podpisu* v praxi za hromadmného podepisování každých 24 hodin (30 krát měsíčně) bude efektivita vypadat takto, viz tabulku 5.4.

Tabulka 5.4: Cena jednoho razítka v *Systému hromadného podpisu* oproti Postsignum a Prvnícertifikační autoritě, A.S. (při podpisu 30 podpisů měsíčně tedy jednou za 24 hodin)

Počet	Cena PS[1] [Kč]	Cena SHP+PS[2] [Kč]	Cena PCA[3] [Kč]	Cena SHP+PCA[4] [Kč]
30	4,03	**4,03**	14,12	**14,12**
50	7,26	**2,42**	8,47	**8,47**
100	3,63	**1,21**	4,24	**4,24**
200	5,14	**0,61**	3,63	**2,12**
300	3,42	**0,40**	2,42	**1,41**
350	2,94	**0,35**	2,60	**1,21**
500	4,84	**0,24**	1,82	**0,85**
1000	2,42	**0,12**	1,82?	**0,42**

[1] Postsignum ceník [46]
[2] Server hromadného podpisu za použití ceníku Postsignum [46]
[3] První certifikční autorita, A.S. ceník [48]
[4] Server hromadného podpisu za použití ceníku První certifikační autority, A.S [48]

Pokud by bylo žádoucí vytvářet časová razítka v kratších časových intervalech a byl by malý počet uživatelů, poměr ceny hromadného razítka vůči ceně autority bude vždy v nejhorším případě jedna ku jedné.

5.4 Návrh rozšiřujících funkcí

Pojednávám zde o tom, jak by se dal *Systém hromadného podpisu* rozšířit. Pak na základě získaní zkušeností načerpaných při vytváření demonstrační aplikace InsuranceAgent zde uvádím výčet možných rozšíření, jenž mají za účel primárně ujednodušit práci uživatele. Jak už bylo v této bakalářské práci popsáno, tak demonstrační aplikace je zamýšlena k použití v systému, pro k pořizování fotografií k pojistným událostem, a proto jsem se také zamýšlel nad tím, jak by se dala tato činost vylepšit, zefektivnit a systém lépe zabezpečit.

Ověření lokace

Bylo vhodné v procesu ověřování doby existence fotografie zapojit i ověření místa vytvoření fotografie pomocí GPS souřadnic. Zde se ovšem znovu setkáváme s dalším údajem, který by mohl být podvrhnut. Tento problém je popsán například v *Fake GPS Defender: A Server-side Solution to Detect Fake GPS* [14]

Autenticita doby vzniku fotografie

Vyřešit, jak ověřit, že fotografie byla opravdu vytvořena daným zařízením ve specifickém časovém úseku.

- Fotoaparát vytvářející šum ve svých fotografiích. Řešení této problematiky rozpoznání původního zařízení na internetu existuje mnoho, jako například *Source camera identification using noise residual* [36], *An Inception-Based Data-Driven Ensemble Approach to Camera Model Identification* [20] a nebo publikace ze skupiny v projektu **PIZZARO**, která se zabývá vývojem softwarového produktu pro podporu identifikace a autentifikací obrazových záznamových zařízení a materiálu a rekonstrukce zachycené obrazové informace [45].

- Reálné objekty umístěné do snímku fotografie. Pořízené snímky potom obsahují objekty, které pokud se nám podaří obhájit, že nemohly existovat dříve než v určitý čas poté máme k zaručené době existence z časového razítka druhý údaj o nejdříve možné době existence. Například noviny, nebo v případě komerčního řešení vytvořit systém, který by dal uživateli ráno možnost ke stažení obrázek například QR kód a ten si ho vytiskl a do fotografií v ten den umísťoval.

V rámci Systému hromadného podpisu by bylo velice jednoduché ho doplnit o uživatelský program, který by takto generoval QR kódy.

Více fotografií k jedné pojistné události

V aktuální logice databáze mobilní aplikace se k jednomu záznamu, neboli pracovnímu úkolu, dá pořídit pouze jedna fotografie. Bylo by vhodné předělat její strukturu tak, aby se dalo pořizovat k jednomu záznamu fotografií více.

Autentizace přes whitelist

Nastavení firemní správy uživatelů a nastavit systém tak, aby nechával časově orazítkovávat pouze záznamy od uživitelů s pozvánkou. Případně vyřešit přes povolení pouze pro uživatele s emailovou adresou patřící pod konkrétní doménu firmy.

Zadávání úkolů

V rámci mobilní aplikace by jednotliví uživatelé dostávali přidělené pracovní úkoly, ke kterým by měli za úkol přijet. Například skrz Firebase přes FMC(Firebase Cloud Messaging) notifikace (viz kapitola Firebase 2.7) by se dali jednoduše pracovníci notifikovat o nově přiděleném pracovním úkolu.

Dále by se pak do jednotlivých záznamů daly vkládat GPS souřadnice, kde se pracovní úkol nachází a seřazovat je od nejbližšího a vypočítávat pro pracovníky pořadí zpracování na základě výběru nejefektivnější trasy.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat mobilní aplikaci, přes kterou bude uživatel schopen vytvářet fotografie s možností zpětného ověření doby jejich vzniku. Tento cíl byl splněn.

Z pohledu zadání jsem měl v prvním bodě za úkol prostudovat literaturu a zanalyzovat obdobné mobilní aplikace se zaměřením na časová razítka fotografií. V sekci 2.8 jsem rozebral jednotlivé aplikace, které stály za zmínku a reprezentovaly odlišný pohled na tuto problematiku. Jejich vlastnosti jsem potom porovnal v kapitole 3.

Druhý bod zadání jsem zpracoval tak, že po uvažování nad tím, v čem by výstup této bakalářská práce mohl být užitečný a inovativní, jsem se rozhodl soustředit na konkrétní postup časového razítkování skupin fotografií, jelikož žádná mobilní aplikace tuto možnost nenabízela, viz kapitola 3.

Třetí bod zadání se týká výběru implementační platformy a samotné implementace. Při výběru implementační platformy pro mobilní zařízení jsem zvolil operační systém Android, pro zálohovací server jsem zvolil Firebase a pro server hromadného podpisu jsem zvolil Ubuntu běžícím jako instance v AWS. Podrobné rozebrání, proč jsem se rozhodl zvolit tyto platformy a způsob implementace navrhovaného podpisu je zpracováno v kapitolách 3 a 4.

Ve čtvrtém bodě zadání jsem měl za úkol implementovat navržený postup a demonstrovat jeho funkčnost. Řešení je popsáno v kapitole 4.

V posledním pátém bodě o vyhodnocení dosažených výsledků a možnosti dalšího pokračování pojednávám podrobně v kapitole 5.3. Ve zkratce řečeno systém, který se mi podařilo zprovoznit tak, že úspěšně využívá implementace hromadného podpisu. Navíc jsem prokázal, že může být použit k vytvoření kvalifikovaných časových razítek a implementován do systémů, ve kterých není potřeba pouze práce s časovými razítky k fotografiím, ale při vytváření jakýchkoliv elektronických dokumentů. Pokud by tedy firma potřebovala časově podepsat existenci tisíce souborů, je možné použít jediný podpis místo jednotlivého podepisování každého souboru zvlášť, což markantně sníží náklady spojené s vytvářením časových razítek.

Co se týče toho, jak by se dal v budoucnu *Systém hromadného podpisu* rozšířit, jsem již podrobně rozebíral v kapitole 5.3, tam si můžete přečíst o veškerých zajímavých rozšířeních, která mě napadla. Jako příklad bych zde zmínil inplementaci schopnosti prokázání doby a místa existence dané fotografie a dále pak možné rozšíření demonstrační aplikace o přidání režie zadávání pracovních úkolů, tak aby se v rámci firemní hierarchie jednotliví pracovníci pohybovali v terénu a jejich vedoucí by mezi ně mohl rozdělovat pracovní úkoly.

Literatura

[1] Zákon č. 227/2000 Sb. Zákon o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu). Tiskárna Ministerstva vnitra, p. o, Praha, CZ, 2000 [Online; navštíveno 18.1.2019].
URL https://www.zakonyprolidi.cz/cs/2000-227

[2] Zákon č. 297/2016 Sb. Zákon o službách vytvářejících důvěru pro elektronické transakce. Tiskárna Ministerstva vnitra, p. o, Praha, CZ, 2016 [Online; navštíveno 18.1.2019].
URL https://www.zakonyprolidi.cz/cs/2016-297

[3] Zákon č. 440/2004 Sb. Zákon, kterým se mění zákon č. 227/2000 Sb., o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu), ve znění pozdějších předpisů. Tiskárna Ministerstva vnitra, p. o, Praha, CZ, 2014 [Online; navštíveno 18.1.2019].
URL https://www.zakonyprolidi.cz/cs/2004-440

[4] Cloud Functions Pricing plans - Products. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 12.4.2019].
URL https://firebase.google.com/pricing/?hl=ru

[5] Cloud Functions - Pricing. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 12.4.2019].
URL https://cloud.google.com/functions/pricing?hl=ru

[6] Adamec, M.: *Chcu víno! - mobilní aplikace pro komunikaci zákazníka s vinařem*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2018.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=20509

[7] Adams, C.; Entrust; Cain, P.; aj.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). Technická zpráva, RFC Editor, Reston, Virginie, USA, 2001.
URL https://www.ietf.org/rfc/rfc3161.txt

[8] Amazon Web Services. Amazon, Seattle, Washington, USA, 2019 [Online; navštíveno 2.5.2018].
URL https://aws.amazon.com/?nc2=h_lg

[9] ATEsystem: Bayerova maska. Ostrava, CZ, 2016 [Online; navštíveno 1.5.2019].
URL http://kamery.atesystem.cz/know-how/slovnik-pojmu-ve-strojovem-videni/bayerova-maska/

[10] Barker, E.; Dang, Q.: *Recommendation for Key Management - Part 3: Application-Specific Key Management Guidance*. Gaithersburg, Maryland, USA: National Institute of Standards and Technology, 2015, ISBN 9781494939045, 12 s.
URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

[11] Android - Camera 2 Basic. [Online; navštíveno 12.12.2018].
URL https://github.com/googlesamples/android-Camera2Basic

[12] CameraView. [Online; navštíveno 12.12.2018].
URL https://github.com/google/cameraview

[13] Černošek, B.: *Klient-server mobilní aplikace se zpracováním obrazu*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2018.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=17951

[14] Chang, Y.-H.; Hwang, Y.-L.; Hu, C.-L.; aj.: Fake GPS Defender: A Server-side Solution to Detect Fake GPS. 2018, ISBN 978-1-61208-658-3, s. 36–41.

[15] Chronologie fotografie - Wikiwand. [Online; navštíveno 15.12.2018].
URL http://www.wikiwand.com/cs/Chronologie_fotografie

[16] Chvála, J.: *Android IP kamera*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2015.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=17905

[17] Conklin, W. A.; White, G.; Williams, D.; aj.: *CompTIA Security+ All-in-One Exam Guide, Fifth Edition (Exam SY0-501)*. New York, New York, USA: McGraw-Hill Education, Leden 2018, ISBN 1260019322, 453 s.
URL https://www.xarg.org/ref/a/1260019322/

[18] Cook, B.: Solving image rotation on Android using Camera2 API. [Online; navštíveno 1.5.2019].
URL https://medium.com/\spacefactor\@m{}kenodoggy/solving-image-rotation-on-android-using-camera2-api-7b3ed3518ab6

[19] Demian, J.: AWS vs Firebase - Is Even a Fair Fight? [Online; navštíveno 26.2.2018].
URL https://dashbird.io/blog/aws-lambda-vs-firebase/

[20] Ferreira, A.; Chen, H.; Li, B.; aj.: An Inception-Based Data-Driven Ensemble Approach to Camera Model Identification. 12 2018, s. 1–7, doi:10.1109/WIFS.2018.8630774, 2018 IEEE International Workshop on Information Forensics and Security (WIFS), Hong Kong 2018.

[21] Firebase Admin Node.js SDK. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 20.3.2019].
URL https://github.com/firebase/firebase-admin-node

[22] Cloud Functions for Firebase. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 2.5.2019].
URL https://firebase.google.com/docs/functions

[23] Firebase Cloud Messaging. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 12.5.2019].
URL https://firebase.google.com/docs/cloud-messaging

[24] Firebase SDK for Cloud Functions. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 20.3.2019].
URL https://github.com/firebase/firebase-functions

[25] Firebase - A comprehensive mobile development platform. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 2.5.2019].
URL https://firebase.google.com/

[26] Firebase Realtime Database. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 2.5.2019].
URL https://firebase.google.com/docs/database

[27] Google: Distribution dashboard. [Online; navštíveno 10.5.2019].
URL https://developer.android.com/about/dashboards

[28] Platform Architecture. Google, Mountain View, Kalifornie, USA, 2019 [Online; navštíveno 10.5.2019].
URL https://developer.android.com/guide/platform

[29] Heliografie – Wikipedie. [Online; navštíveno 15.12.2018].
URL https://cs.wikipedia.org/wiki/Heliografie

[30] Henry: Android Camera 2 API example tutorial. [Online; navštíveno 12.12.2018].
URL
https://inducesmile.com/android/android-camera2-api-example-tutorial/

[31] Hloušková, V.; Lubas, J.; Rosol, I.; aj.: Důvěryhodný digitální dokument. [Online; navštíveno 10.2.2019].
URL http://www.ipsd.cz/wp-content/uploads/2017/03/2014-03-25-ictu-duveryhodny-dokument-final-12-ds.pdf

[32] Imrich, J.: Possible Bug in OpenSSL - rfc 3161 - TSA service. [Online; navštíveno 14.2.2019].
URL http://openssl.6102.n7.nabble.com/possible-Bug-in-OpenSSL-rfc-3161-TSA-service-td43128.html

[33] Introduction to Photography. [Online; navštíveno 15.12.2018].
URL https://photographylife.com/what-is-photography

[34] Java security library Signature. [Online; navštíveno 2.5.2019].
URL
https://docs.oracle.com/javase/7/docs/api/java/security/Signature.html

[35] Jonsson, J.; Kaliski, B.: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RSA Laboratories, Bedford, Massachusetts, United States, 2003 [Online; navštíveno 12.3.2019].
URL https://tools.ietf.org/html/rfc3447

[36] K R, A.; Anitha, H.; A K, K.; aj.: Source camera identification using noise residual. Bangalore, India: IEEE, Květen 2016, s. 1080–1084, doi:10.1109/RTEICT.2016.7807997.
URL https://ieeexplore.ieee.org/document/7807997

[37] Kaliski, B.: Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2. RSA Laboratories, Bedford, Massachusetts, United States, 2008 [Online; navštíveno 12.3.2019].
URL https://tools.ietf.org/html/rfc5208

[38] Kubica, T.: TServerless - existují aplikace bez serverů? [Online; navštíveno 26.2.2018].
URL https://www.tomaskubica.cz/post/2017/serverless-existuji-aplikace-bez-serveru/

[39] Lynch, V.: Re-Hashed: The Difference Between SHA-1, SHA-2 and SHA-256 Hash Algorithms. 2018, [Online; navštíveno 8.4.2019].
URL https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/

[40] Mareš, J.: *Doplnění geografických souřadnic do fotografií podle záznamu trasy*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2018.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=21259

[41] Nikkel, B.: *Practical Forensic Imaging*. San Francisco, Kalifornie, USA: No Starch Press, 2016, ISBN 9781492018049, 201 s.
URL https://books.google.cz/books?id=7zRZAQAACAAJ

[42] NOVÁČEK, B. P.: *MODERNÍ PROSTŘEDKY PRO DIGITÁLNÍ SNÍMÁNÍ SCÉNY*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2015.
URL
https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=99967

[43] OpenSSL knihovna implementující SSL a TLS protokoly. [Online; navštíveno 15.1.2019].
URL https://www.openssl.org/

[44] Nástroj pro práci s časovými razítkami v rámci knihovny OpenSSL. [Online; navštíveno 15.1.2019].
URL https://www.openssl.org/docs/manmaster/man1/ts.html

[45] Pizzaro, T.: Projekt PIZZARO. Praha, CZ, 2014 [Online; navštíveno 10.5.2019].
URL http://pizzaro.utia.cas.cz/index.php/cz/

[46] Postsignum ceník. Česká pošta, Praha 1, Praha, 2019. [Online; navštíveno 1.4.2019].
URL http://www.postsignum.cz/casova_razitka.html

[47] Postsignum slovník. Česká pošta, Praha 1, Praha, 2019. [Online; navštíveno 20.3.2019].
URL http://www.postsignum.cz/slovnicek_pojmu.html

[48] První certifikační autorita, A.S. ceník. [Online; navštíveno 1.4.2019].
URL https://www.ica.cz/Cenik

[49] Request - Request promise. [Online; navštíveno 22.1.2019].
URL https://github.com/request/request-promise

[50] RFC3161 TimeStamping Verifier. [Online; navštíveno 2.11.2018].
URL https://play.google.com/store/apps/details?id=com.clepsydratime.timestampverifier

[51] What is the difference between hashing and encrypting. 2016, [Online; navštíveno 4.4.2019].
URL https://www.securityinnovationeurope.com/blog/page/whats-the-difference-between-hashing-and-encrypting

[52] Siska, J.: *Demonstrační aplikace hashovacích algoritmů SHA-1 a SHA-2*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2012.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=12411

[53] Sklenář, Z.: *Vyhledávání duplicitních fotografií*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2018.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=21432

[54] Stenberg, D.: *Everything curl*. Emeryville, Kalifornie, USA: Alibris, 2018, ISBN 978-91-639-6501-2.
URL https://curl.haxx.se/book.html

[55] Stužka – Wikipedie. [Online; navštíveno 15.12.2018].
URL https://cs.wikipedia.org/wiki/Stu%C5%BEka

[56] Svanlund, D.: how to get a Static DNS name for an instance API. [Online; navštíveno 6.4.2019].
URL https://forums.aws.amazon.com/thread.jspa?threadID=139099

[57] tempfile – Generate temporary files and directories. [Online; navštíveno 13.3.2019].
URL https://docs.python.org/3/library/tempfile.html

[58] thisbejim: Pyrebase - A simple python wrapper for the Firebase API. [Online; navštíveno 14.3.2019].
URL https://github.com/thisbejim/Pyrebase

[59] Timestamp Camera Free. [Online; navštíveno 2.11.2018].
URL https://play.google.com/store/apps/details?id=com.jeyluta.timestampcamerafree

[60] Vícha, T.: *Bezpečná komunikační aplikace pro Windows Phone*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, CZ, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19965

[61] Žára, O.: Firebase: krátké seznámení. [Online; navštíveno 1.5.2019].
URL https://www.zdrojak.cz/clanky/firebase-kratke-seznameni/

Příloha A

Obsah přiloženého pamětového média

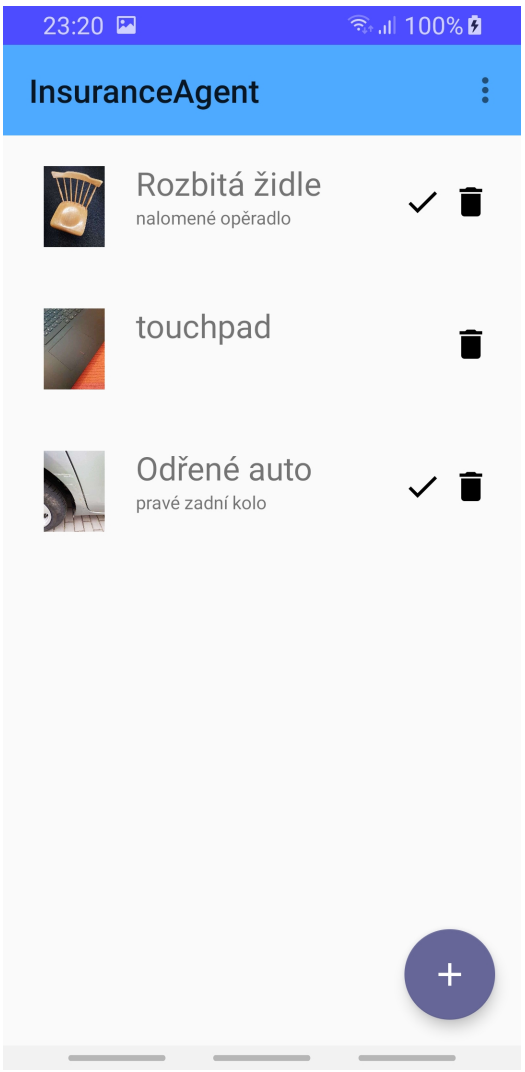
- `\InsuranceAgent\Android` - zdrojové soubory mobilní aplikace
- `\InsuranceAgent\Firebase` - zdrojové soubory zálohovacího serveru
- `\InsuranceAgent\Server` - zdrojové soubory serveru hromadného podpisu
- `\InsuranceAgent\Res` - html zdrojové kódy a obrázky webové stránky
- `\InsuranceAgent\verificator.py` - uživatelský ověřovací nástroj
- `\Thesis` - latex zdrojové kódy a obrázky použité v této práci
- `\thesis.pdf` - text práce ve formátu pdf

Příloha B

Uživatelské rozhraní

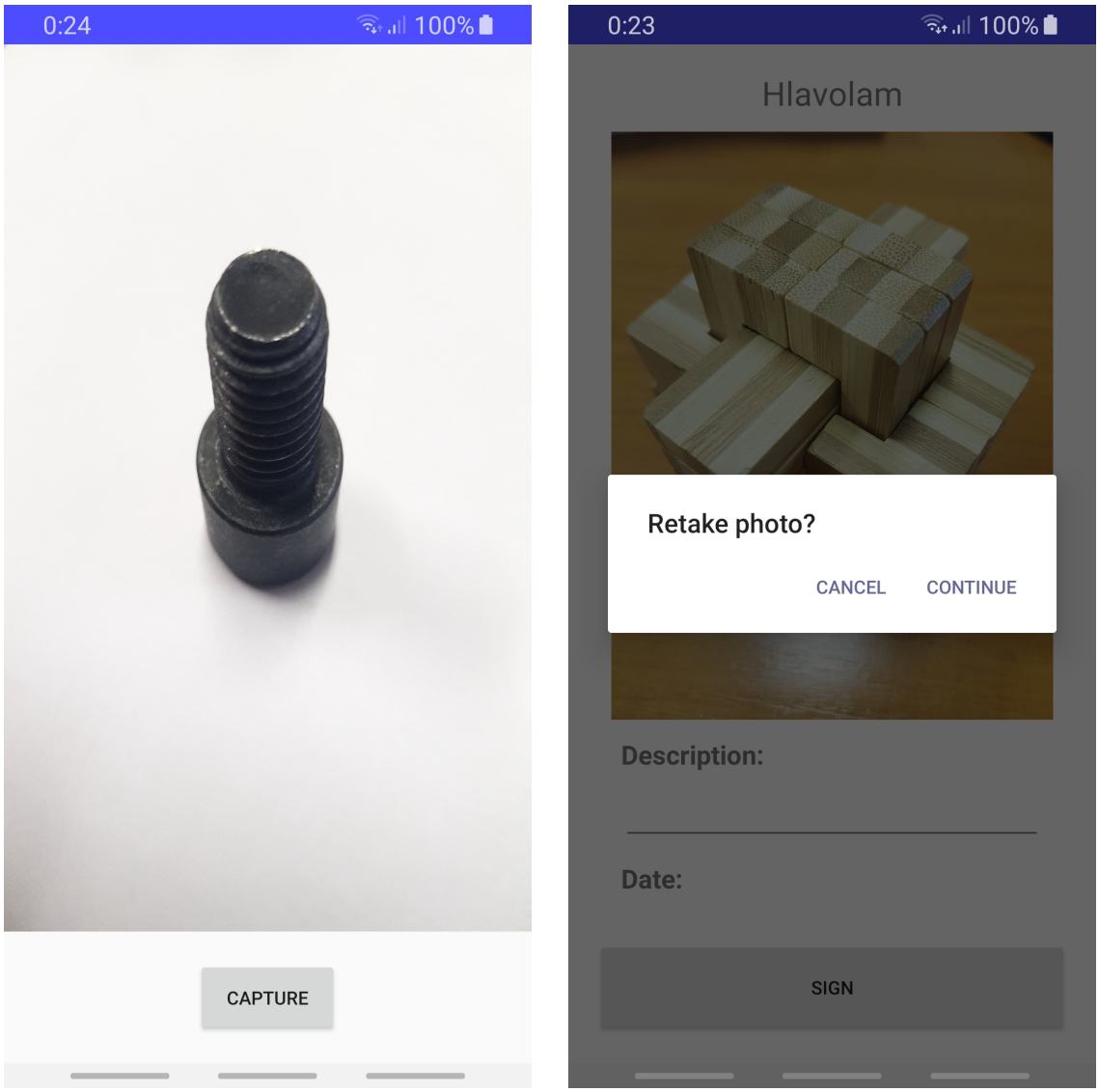
Mobilní aplikace

(a) Přihlašovací obrazovka



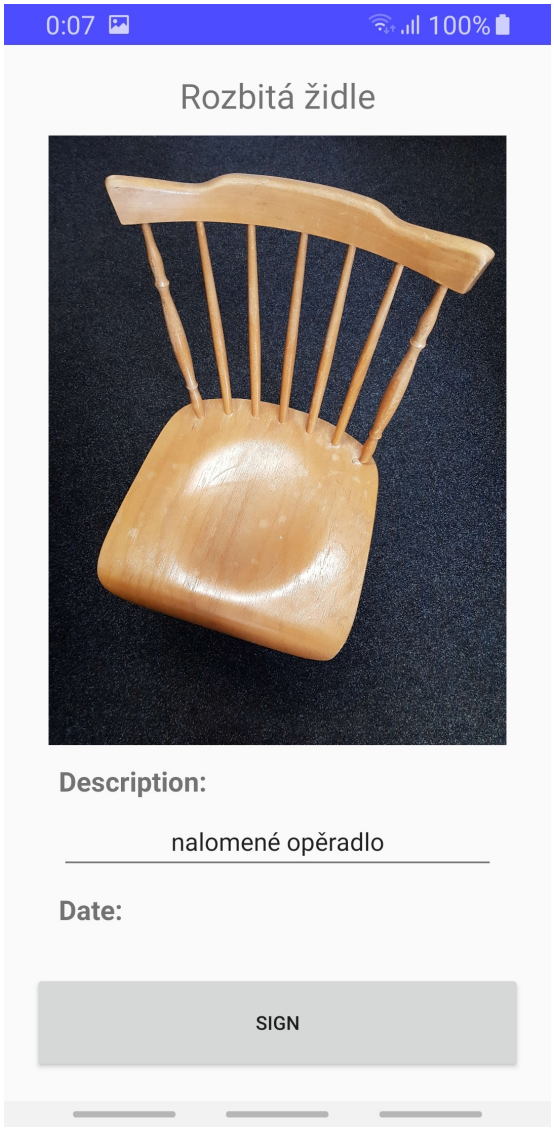
(b) Menu

Obrázek B.1: Obrazovky mobilní aplikace č.1

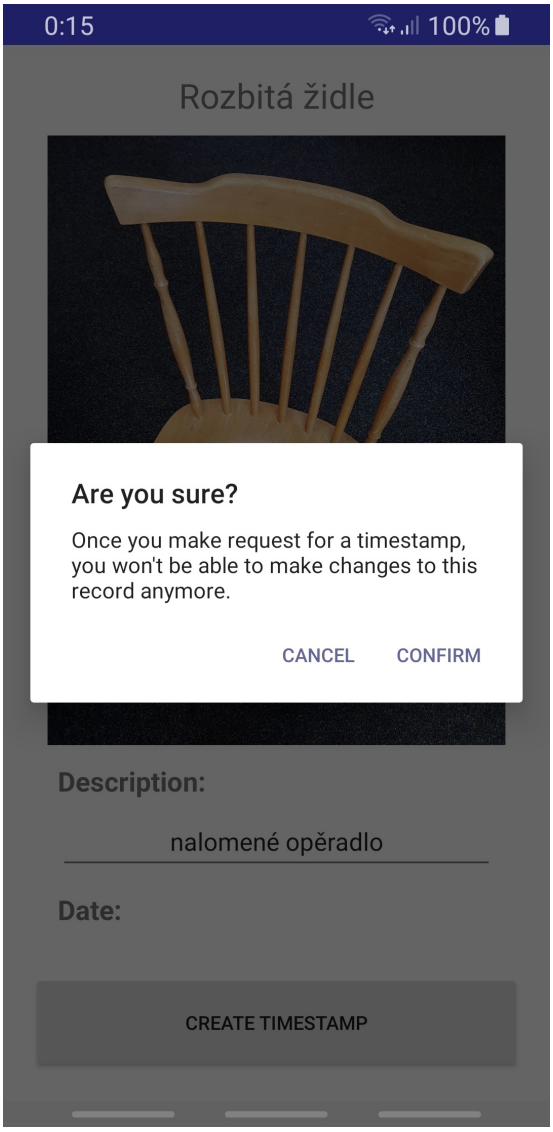


(a) Vzhled kamery (b) Dotaz zda je uživatel chce fotografii přefotit.

Obrázek B.2: Obrazovky mobilní aplikace č.2

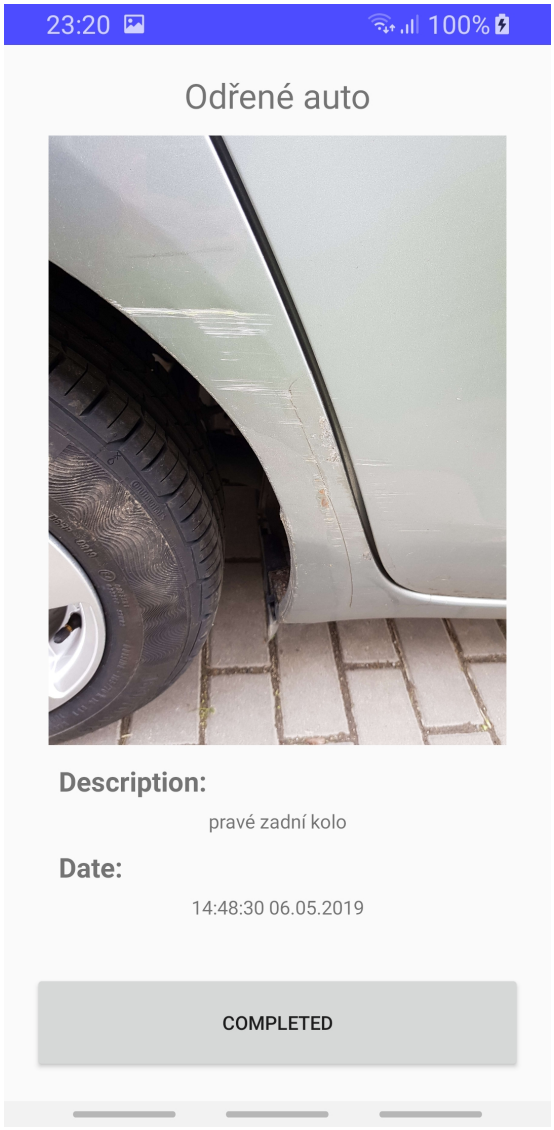


(a) Obrazovka po vyfocení fotografie

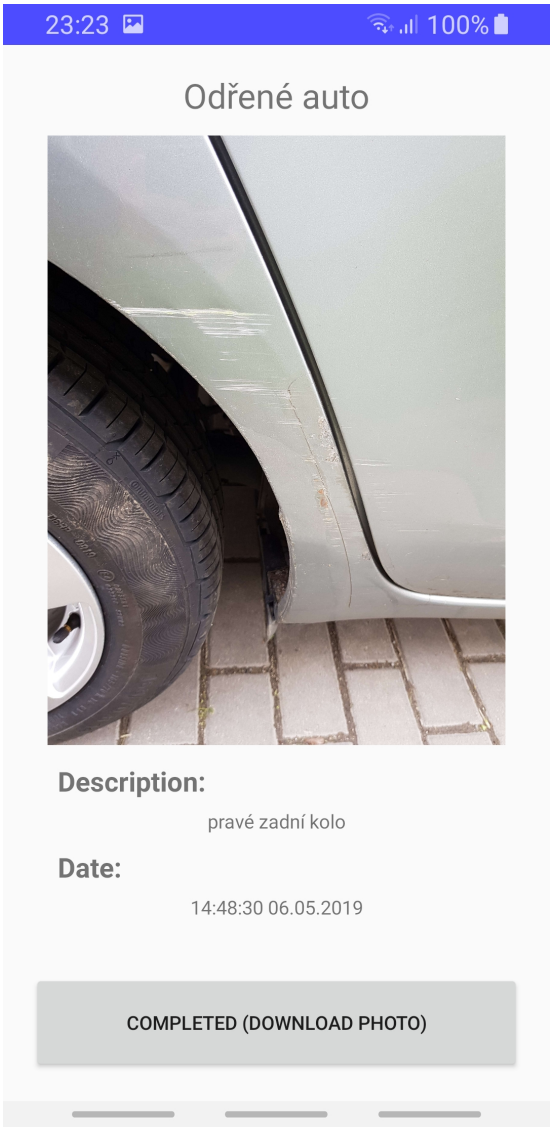


(b) Dotaz zda je uživatel rozhodnut záznam odeslat k časovému orazítkování.

Obrázek B.3: Obrazovky mobilní aplikace č.3



(a) Obrazovka s připraveným záznamem k odeslaní na časové orazítkovaní



(b) Bylo zjištěno, že se fotografie časově orazítkovaného záznamu nenachází v uložišti telefonu na původním místě.

Obrázek B.4: Obrazovky mobilní aplikace č.4

Webová aplikace

(a) Úvodní obrazovka webové stránky

(b) Úspěšné ověření

Obrázek B.5: Obrazovky webové aplikace č.1

PARTIAL-SUCCESS: Timestamp is CORRECT
ERROR: Selected public key wasn't used to create this timestamp.

(a) Fotografie je časově orazítkována, ale nebyl použit původní veřejný klíč autora.

ERROR: Timestamp doesn't exist.

(b) Při výskytu chyby ověřování.

Obrázek B.6: Obrazovky webové aplikace č.2

Příloha C

Manuály

Zde jsou vypsány uživateské manuály pro využití jednotlivých uživatelských aplikací.

Mobilní aplikace

Mobilní aplikace InsuranceAgent slouží k pořizování fotografií a k těmto fotografiím vytvářet časové řazítka.

Nastavení klíču uživatele

Pro správnou funkčnost této aplikace je zapotřebí mít vygenerovaný a nainstalovaný vlastní pár klíčů pro elektronický podpis (RSA). Veřejný klíč ve formátu **X.509** a klíč soukromý formátu **PKCS 8** klíč pro RSA podpis. Pokud tyto klíče dosud nevlastníte aplikace vám je sama nabídne při vytváření podpisu vygenerovat.

Ovšem pokud jste si klíče již vygeneroval a nebo nechal vygenerovat pro používání jiného zařízení, můžete si je do telefonu přenést, tím že tyto soubory je neprve třeba pojmenovat jako `publickey` a `privatekey` a umístit je do paměti mobilního zařízení, do adresáře `/InsuranceAgent/`. Pokud nejste technicky zdatní, tak je doporučeno pro tuto operaci kontaktaktovat technického správce, který vám s tímto nastavením pomůže.

Přihlášení

Při otevření aplikace je spuštěna úvodní obrazovka pro přihlášení. Po kliknutí na tlačítko přihlásit vám bude zobrazena nabídka Google účtů. Jakmile jednou účet zvolíte bude volba zapamatována, až do případného odhlášení.

Snímání fotografií

Pokud nastalo přihlašování úspěšně je vám zobrazena obrazovka se seznamem jednotlivých pracovních úkolů (záznamů) s fotografiemi. Pokud chcete pořídit fotografii novou, tak pro tento účel se v pravém dolním rohu nachází tlačítko „+“, na jehož kliknutí je oteřeno vyskakovací okno se žádostí o vyplnění názvu pracovního úkolu (záznamu), pod který bude daná fotografie patrit.

Jakmile je záznam vytvořen je možné k němu pořídit fotografii tak, že kliknutím na jeho název z hlavní obrazovky otevřete jeho detail.

V detailu nově vytvořeného záznamu, bez přiřazené fotografie je defaultní ikona fotoaparátu, na jehož kliknutí je spuštěn fotoaparát, kterým fotografii pořídíte. V rámci detailu je také možné připsat podrobnější popis do kolonky `Description`.

Proces vytváření podpisu

Jakmile je k záznamu vyfotografována fotografie, je možné z této fotografie vytvořit podpis pře kliknutí na tlačítko `SIGN` situované ve spodní části obrazovky detailu zánamu. Jakmile je záznam podepsán je stále možné fotografii přefotit, ale bude nutné podpis znovu vytvořit.

Proces žádosti o časové razítko

V moment kdy je fotografie podepsána je v detailu záznamu ve spodní části zobrazeno tlačítko `CREATE TIMESTAMP`, je zobrazena žádost o potvrzení, jelikož po odeslání fotografie k časovému orazítkování, již nebudete schopni z mobilní aplikace jakoliv data v databázi změnit. Jakmile proběhne časové orazítkování fotografie, je v detailu záznamu zobrazen popisek `COMPLETED` spolu s kolonkou `Date`, ve které je čas vystavení časového razítka. V rámci seznamu záznamů je jasně označen symbolem zaškrtnutí.

Zpětné stažení fotografie

Pokud fotografii z lokálního uložistě smažete, je vám u podepsaného záznamu místo tlačítka `COMPLETED` zobrazeno `COMPLETED (DOWNLOAD PHOTO)`. Při kliknutí na toto tlačítko se fotografie do vašeho zařízení uloží.

Odhlášení

V hlavním obrazovce se seznamem pracovních úkolů (záznamů) se v horním pravém rohu nachází tlačítko s možnostmi, ve kterém zvolte položku `odhlásit`.

Webová aplikace

Webová aplikace slouží k ověření úspěšného vytvoření časového razítka ke konkrétní fotografii. Případně ověření totožnosti autora fotografie.

Na úvodní stránce je možné nahrát soubor fotografie a volitelně veřejného klíče. Po zadání těchto souborů se proces ověření spouští klepnutím na tlačítko `Verify`. Na základě výsledku procesu vám bude zobrazena odpovídající odpověd.

Verifikátor

InsuranceAgent verificator je nástroj v jazyce Python ve verzi 3.6. Pro správnou funkčnost tohoto programu je potřeba mít na počítači nainstalovaný nástroj **OpenSSL ts** dostupný na adrese https://www.openssl.org/docs/manmaster/man1/ts.html.

Tento program slouží k ověření, že hromadné (skupinové) časové razítko pro danou fotografii bylo úspěšně vytvořeno. Volitelně je zde možné vložit certifikát autora a tím ověřit jeho totožnost. Zahrnuje také možnost exportu dat k možnosti ověření vystaveného hromadného časového razítka i ve vnějším systému.

Parametry

```
verificator.py --inputfile <inputfile> [--authorcert <authorcert>]
[--details] [--outputfile <outputfile>]
```

- **inputfile (-i)** - Cesta k fotografii určené k ověření.
- **authorcert (-a)** - Autorův veřejný certifikát.
- **details (--d, -d)** - Zapnutí detailního výpisu na výstup.
- **outputfile (-o)** - Nastavení prefixu s cestou na základě kterých se vytvoří soubory s příponami `*.txt`, `*.tsq`, `*.tsr` (v případě použití parametru `authorcert` se také vytvoří soubor `*.signature`).