Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vito Tomažin

Implementacija storitve časovni žig

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Osrednji cilj naj bo razvoj spletne storitve časovni žig. V delu predstavite kriptografske osnove, kot so npr. zgoščevalne funkcije in asimetrična kriptografija (dogovor o ključu, digitalni podpis in digitalna potrdila), ki so potrebne za varno implementacijo. Preučite poznane sheme časovnega žigosanja ter za rešitev izberite ustrezno. Naredite tudi pregled uporabljenih orodij in tehnologije. Sledi naj podroben opis uporabe in delovanja same aplikacije.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Vito Tomažin, z vpisno številko **63100291**, sem avtor diplomskega dela z naslovom:

Implementacija storitve časovni žig

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Aleksandra Jurišića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

Kratica	Angleško	Slovensko
TSA	time stamping authority	izdajatelj časovnih žigov
TSS	time stamping service	storitev za časovno žigosanje
PKI	public key infrastructure	infrastruktura javnih ključev
CA	certification authority	certifikatna agencija
WoT	web of trust	mreža zaupanja
RNG	random number generator	generator naključnih števil
DoS	denial of service	ohromitev storitve

Povzetek

Diplomsko delo obravnava načine varnega časovnega žigosanja digitalnih vsebin. Potreba po časovnem žigosanju se pojavi, kadar bi želeli ustvariti dokaz, da je določena vsebina obstajala (ali pa oseba bila nekje prisotna) ob natanko določenem času in na katerega se bomo v prihodnosti lahko sklicevali. V digitalnem svetu je umeščanje podatkov v časovno obdobje precej zahtevno, saj noben podatek ni zapisan trajno in tudi čas je predstavljen zgolj kot podatek. Sistemsko uro digitalnih naprav, ki čas meri s pomočjo oscilatorja, se da brez težav manipulirati. Komu v takem primeru sploh zaupati? Na spletu obstajajo v ta namen specializirane storitve, ki podatke žigosajo z verodostojnim časom. Naša študija zajema analizo znanih shem zaupanja kot tudi implementacijo storitve časovnega žigosanja, ki uporablja različne tehnike za dokazovanje časovne umeščenosti.

Ključne besede: časovno žigosanje, kriptografija, infrastruktura javnih ključev, zgoščevalna funkcija, zgoščevalna veriga, spletni vmesnik, računalniška varnost.

Abstract

This thesis deals with different ways of how to safely time-stamp digital content. The need for time-stamping arises when we would like to create evidence that certain content existed (or that a certain person was present somewhere) in a precisely specified period of time, so that later we can refer to this evidence. Creating temporal order in the digital world can prove to be difficult, since no data, including time, is recorded permanently. The system clock, which measures time with the help of an oscillator, can easily be manipulated. Who to trust in such case? Specialized services, which time-stamp data with a reliable temporal accuracy, can be found online. Our study includes an analysis of known trust based schemes as well as the implementation of time-stamping service which uses various techniques to prove the placement of data within a specified time period.

Keywords: time stamping, cryptography, public key infrastructure, hash function, hash chain, web interface, computer security.

Poglavje 1

Uvod

Potrebe po varnem beleženju časa nastanka in spremembe dokumentov oziroma podatkov nasplošno so se pojavile že davno pred množično uporabo računalnikov. Najpogosteje seveda na področju intelektualne lastnine, za zaščito idej ter inovacij. Raziskovalci in izumitelji so potrebovali metode, s katerimi so lahko zaščitili avtorstvo iznajdb, preden so jih predstavili širši javnosti. S časoma se je namen uporabe razširil na dokazovanje dokumentov, ki niso ravno namenjeni patentiranju, še zmeraj pa je pomemben čas njihovega nastanka. Na primer, na misel nam pride poslovna ideja, za katero smo prepričani, da ima ogromen potencial. Vendar je za samo realizacijo potrebna večja investicija, žal pa sami nimamo zadostnega kapitala. Imamo problem, saj ideje ne želimo razkriti, kjub temu pa moramo poiskati investitorja in mu idejo seveda predstaviti. Kako se zaščititi pred krajo ideje? S podobnim problemom se sooči tudi študent pri izdelavi domače naloge. Študent pride do rešitve naloge, sedaj bi rad pomagal kolegom in jim pokazal svoj izdelek. Vendar ga je strah, da bo za inovativno rešitev nagrajen nekdo drug ali sam celo obtožen plagiatorstva. Kako se študent lahko zaščiti kot avtor? Časovno dokazovanje je uporabno tudi pri zavarovalnicah ob nezgodah. V današnjem času, ko ima skoraj vsakdo izmed nas v žepu pametni telefon, lahko ob nezgodi nastalo škodo fotografira, ali še bolje, posname. Gradivo časovno žigosa in se tako izogne kasnejšim nevšečnostim. Primerov, v katerih

bi se oseba rada zaščitila pred krajo ideje oziroma bi kasneje rada dokazala, da je v določenem času že imela specifično informacijo, je ogromno. Splošen namen časovnega žigosanja je torej ustvariti dokaz, da je določen podatek obstajal ob nekem času.

Osnovna metoda za časovno žigosanje je uporaba knjige/dnevnika, v katerega se vpisuje vnose. Le-ti so kronološko urejeni oziroma se zapisujejo zaporedoma brez praznih listov in vrstic. Če so strani oštevilčene in vezane ali lepljene, je vnose v takšnem dnevniku izredno težko vrivati in spreminjati. K dodatni kredibilnosti pripomore redno žigosanje ali podpisovanje samih strani, kar opravlja nepristranska oseba v funkciji notarja. Obstajajo še druge metode časovnega žigosanja. Lahko si pošljemo pismo in ga pustimo zaprtega. Žig datuma, kdaj je bilo pismo odpremljeno na pisemski ovojnici tako dokazuje, da je bila vsebina pisma napisana pred tem. Podjetja za kredibilnost svojih internih dokumentov ponavadi uporabljajo različne postopke, kjer več oseb pregleda dokument oziroma nadzoruje vnose. Kakršnokoli spreminjanje s strani ene osebe bi ostali razkrili [9].

Žal omenjena metoda danes ni več učinkovita, saj je prepočasna in ne zadosti našim potrebam. Z razvojem tehnologije je zmeraj več dokumentov, slik, zvočnih in video posnetkov v digitalni obliki. Z besedo dokument v nadaljevanju besedila označujemo kakršno koli digitalno vsebino, ne nujno le v obliki besedila. Tako so se pojavili novi izzivi, kako overiti, kdaj je bila vsebina zabeležena in nazadnje spremenjena. Problemi, s katerimi se soočamo, so sledeči:

- Za razliko od rokopisa digitalno zakodirana vsebina ni unikatna.
- Digitalna vsebina se nahaja na medijih, ki jih je enostavno spreminjati (kopiranje in spreminjanje vsebine za seboj ne pušča nobenih sledi).
- Vsebina ni fizično vezana na medij. Navadno nimamo dostopa do originala oziroma niti ne vemo, kje se ta nahaja, saj so kopije identične.
- Čas v digitalni obliki je predstavljen kot vsak drug podatek. Torej kot zaporedje bitov, in če se nahaja na mediju, katerega vsebino se lahko

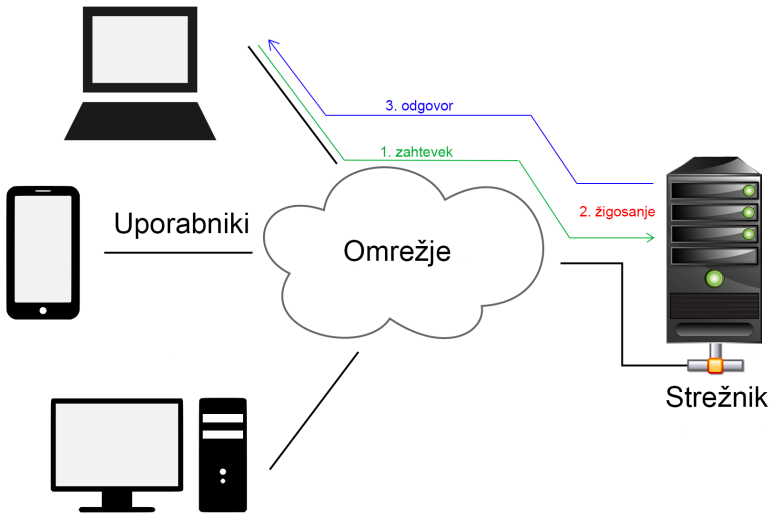
spreminja, mu ne moremo zaupati.

Iz vseh zgoraj omenjenih primerov in preprek lahko povzamemo, da za kredibilno časovno žigosanje in kasnejše dokazovanje potrebujemo mehanizem, ki vsebuje komponento v vlogi tretje osebe (naj gre za eno osebo, skupino ljudi ali širšo javnost). Sama izbira modela je povsem odvisna od stopnje varnosti, ki je za nas sprejemljiva, in tega, za kakšne podatke gre. Potreba je torej po zaupanja vredni digitalni storitvi za časovno žigosanje (angl. trusted Time Stamping Service - TSS). V naslednjem podpoglavju bomo opisali logično shemo storitve in kakšnim kriterijem mora ustrezati.

1.1 Shema in kriteriji

Zasnova storitve je odvisna od tega, komu je namenjena. Časovno žigosanje je nenazadnje lahko namenjeno posameznikom (končnim uporabnikom) ali drugim računalniškim storitvam ter programom, pri katerih se pojavi potreba po beleženju časa dogodkov ali podatkov. Seveda lahko ustreza kombinaciji obeh. Bistvena razlika je v sami količini prometa oziroma številu zahtevkov na časovno enoto. V našem primeru bo storitev primarno namenjena Fakulteti za računalništvo in informatiko. Tako študentom fakultete za žigosanje domačih nalog in raznih dokumentov kot tudi profesorjem in ostalim zaposlenim.

Potrebujemo storitev, ki bo delovala v omrežju, naj bo to internet ali lokalno omrežje znotraj fakultete, kjer bo zmeraj na voljo in njena uporaba hitra. Logično je torej delovanje storitve kot strežnika (angl. server), ki čaka na zahteve žigosanja poslane s strani uporabnikov (angl. client). V grobem uporabnik pošlje dokument, storitev žigosa vsebino ter vrne informacijo, ali je proces uspel. Ponazoritev je na sliki 1.1. Poleg časovnega žigosanja mora nuditi tudi možnost kasnejšega preverjanja časovnih žigov. Celoten proces se mora opraviti na nivoju podatkov in ne sme dopuščati spreminjanja le-teh, ne glede na to, na kakšnem mediju so shranjeni. Želeli bi, da manipulacija časa ni možna niti s strani vzdrževalcev storitve. Zagotovljena mora biti



Slika 1.1: Postopek žigosanja

tudi zaupnost dokumentov, saj ne želimo, da bi kdorkoli lahko prebral vsebino med samim prenosom ali v času, ko je dokument na strežniku. Prav tako ne smemo pozabiti na uporabniški vmesnik, saj pri vsaki storitvi, ki je namenjena končnim uporabnikom, želimo, da je njena uporaba kar se da enostavna.

1.2 Struktura diplomske naloge

V 2. poglavju bomo predstavili kriptografijo, ki je temelj varne računalniške komunikacije. Podrobneje bomo opisali delovanje zgoščevalnih funkcij in infrastrukturo javnih ključev, s katerima smo si pomagali pri rešitvi. Znani pristopi in modeli storitve za časovno žigosanje so opisani v poglavju 3. Sledi predstavitev implementacije in uporabe naše rešitve, ki se nahaja v 4. poglavju. Nalogo zaključimo v poglavju 5, kjer povzamemo naše ugotovitve.

Poglavje 2

Kriptografija

Kriptografija je znanost o šifriranju podatkov. Sama beseda je sestavljena iz besed grškega izvora. *Kryptós*, ki pomeni skrito, in *graphein*, ki pomeni pisanje. V zgodovini je bila kriptografija omejena predvsem na zaščito podatkov pred neželenimi bralci. Šifriranje je namreč proces, ki pretvori berljivo sporočilo (čistopis) v neberljivo (tajnopis), ki ga je zelo težko ali celo nemogoče razbrati brez ustrezne dodatne informacije. "Eden izmed najstarejših znanih primerov uporabe kriptografije izhaja od leta 1500 pred našim štetjem. V tistem času je mezopotamski lončar uporabil skrivne znake za zapis postopka glazure na glinaste tablice. Že pred 3500 leti je bila potreba po skrivanju industrijskih skrivnosti pred konkurenco [19]."Prikrivanje oziroma na drugi strani prestrezanje informacij je ključnega pomena tudi v vojnah. Tak primer je Cezarjeva šifra - postopek, ki ga je uporabljal Julij Cezar za varno komunikacijo. Gre za enostavno zamenjalno šifro, kjer se izvede transformacija med črkami. Cezar je imel navado zamakniti celotno abecedo za tri črke, kot je prikazano na sliki 2.1.

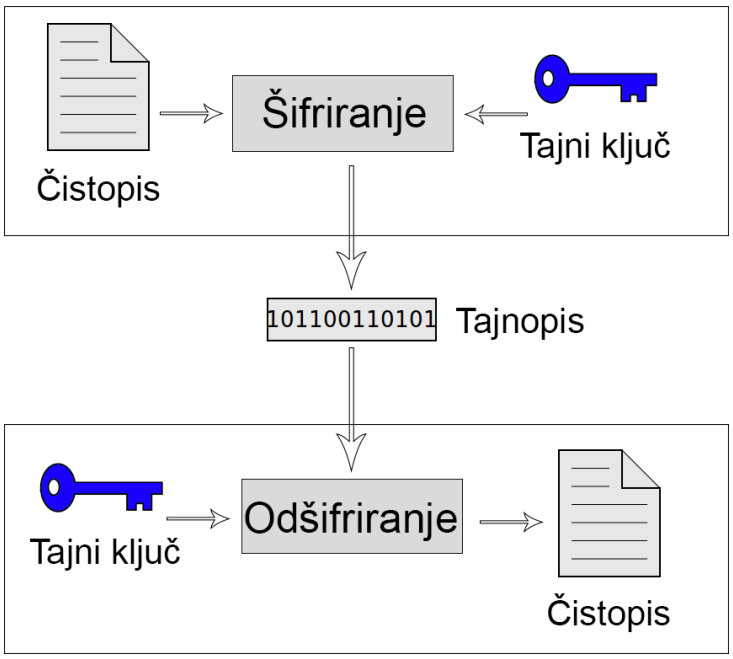
Standardna abeceda: ABCČDEFGHIJKLMNOPQRSŠTUVWXYZŽ
Tajna abeceda: ČDEFGHIJKLMNOPQRSŠTUVWXYZŽABC
Čistopis: Cezar
Tajnopis: Ehbčt

Slika 2.1: Primer Cezarjeve šifre

Sodobna kriptografija je danes tesno povezana z matematiko in računalništvom, saj sloni na reševanju matematičnih problemov, računalništvo pa ponuja računsko zmogljivost. Je eden najboljših mehanizmov za zagotavljanje varnosti v digitalnem svetu, kjer poskuša reševati naslednje štiri probleme:

1. Zaupnost. Podatki niso razumljivi osebam katerim niso namenjeni.
2. Celovitost. Možnost zaznave, če so bili podatki spremenjeni.
3. Nezatajljivost. Avtor kasneje ne more zanikati svojega dejanja.
4. Avtentikacija. Overitev in nespornost identitete.

Za potrebe našega dela bomo kriptografijo razdelili na tri podpodročja: kriptografija tajnih ključev, kriptografija javnih ključev in zgoščevalne funkcije. Čeprav kriptografija tajnih ključev za časovno žigosanje ni pomembna, se nam vseeno zdi vredno, da predstavimo njen koncept. Najprej je potrebno razčistiti pomen dveh osnovnih pojmov. **Algoritem** je postopek ali recept za rešitev nekega problema. V našem primeru je kriptografski algoritem postopek za šifriranje podatkov oziroma opisuje, na kakšen način se bodo podatki transformirali. Poleg postopka uporabljamo **ključe**, ki niso nič drugega kot skrivnosti oziroma gesla in enolično določajo transformacijo. Pri simetričnih algoritmih uporabljamo en ključ, ki služi za šifriranje in odšifriranje, kot je prikazano na sliki 2.2. Najbolj razširjena simetrična algoritma sta DES/3DES in AES. Takšni postopki nudijo srednjo stopnjo varnosti, so hitri, največji problem pa je izmenjava ključa, če jih uporabljamo za šifriranje komunikacije



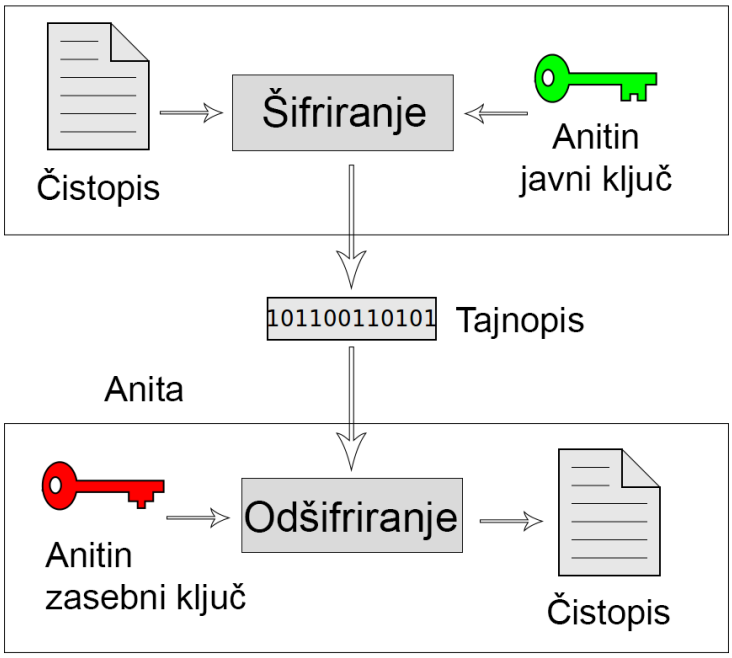
Slika 2.2: Simetrična kriptografija

(t. i. sejni ključi). Osebi, ki bi želeli varno komunicirati, se morata najprej sestati in dogovoriti kakšen bo ključ, ali uporabiti kateri drug varen komunikacijski kanal. Problem nastane tudi pri hranjenju ključev. V množici n uporabnikov, ki bi želeli komunicirati drug z drugim, bi vsak moral hraniti $n-1$ ključev. Skupno število tajnih ključev bi tako znašalo $n(n-1)/2$.

2.1 Kriptografija javnih ključev

"Kriptografijo javnih ključev sta javnosti prvič predstavila Diffie in Hellman leta 1976. Osnovna ideja je, da ločimo funkcijo zaklepanja in odklepanja, tj. šifriranje in odšifriranje. Namesto da bi uporabili isti ključ za obe operaciji, uporabimo par ključev, ki se med seboj razlikujeta. Preprost opis je sledeč. Anita generira svoj par ključev. Zasebni ključ (angl. private key) *sk* obdrži

zase, javni ključ (angl. public key) *pk* pa objavi. Kdorkoli, ki želi poslati Aniti sporočilo, lahko zašifrira čistopis *m* v tajnopis *c* z uporabo Anitinega javnega ključa *pk*. Tako zašifriran tajnopis *c* lahko odšifrira samo Anita z uporabo svojega zasebnega ključa *sk* [22]."Postopek je prikazan na sliki 2.3.



Slika 2.3: Asimetrična kriptografija

Glavna prednost takšnega pristopa pred uporabo simetričnih algoritmov je, da omogoča varno komunikacijo, ne da bi se akterji morali predhodno srečati zaradi dogovora o tajnem ključu. Vsak lahko z javnim ključem zašifrira sporočilo, prebral pa ga bo lahko samo lastnik zasebnega komplementarnega ključa. Koncept lahko primerjamo z uporabo poštnega nabiralnika, v katerega ima možnost odvreči pismo kdorkoli, odpre pa ga le lastnik.

Zaklepanje in odklepanje seveda deluje tudi v nasprotni smeri. Če sporočilo zašifriramo z zasebnim ključem, se ga da odšifrirati le z javnim ključem.

Ker ima zaseben ključ le njegov lastnik, lahko na tak način dokazuje identiteto. Omenjeni način uporabe asimetričnih algoritmov se uporablja za digitalne podpise. V Sloveniji njihovo pravno uporabo in veljavnost ureja ZEPEP (Zakon o elektronskem poslovanju in elektronskem podpisu) [23].

2.1.1 Digitalni podpis

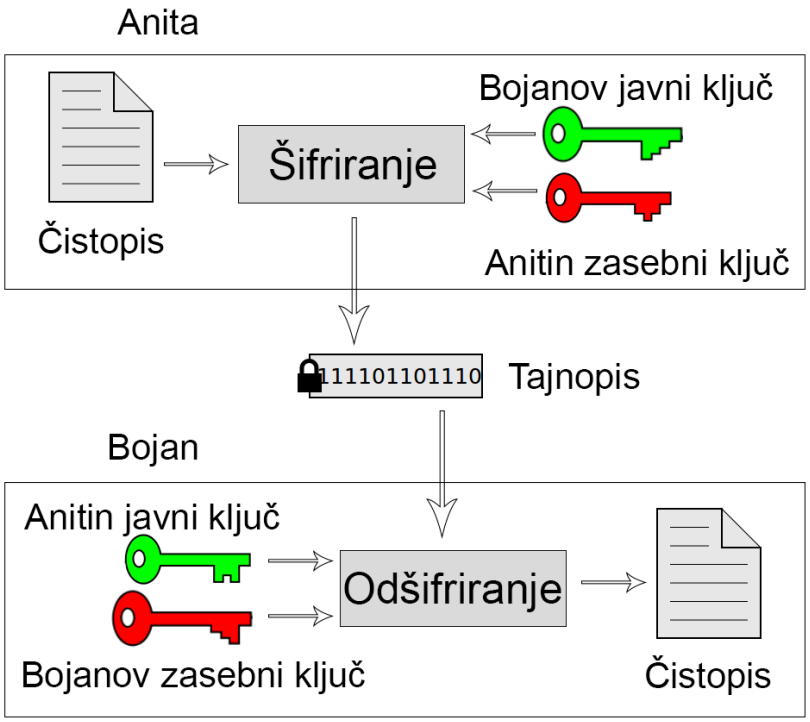
Digitalni podpis (angl. digital signature) ima enako funkcijo kot lastnoročni podpis, vendar za dokumente v digitalni obliki. Zagotoviti mora torej naslednje lastnosti:

- Avtentičnost (podpisnik je res tisti, za kogar se predstavlja).
- Podpisa se ne da ponarediti.
- Podpisa se ne da kopirati (oziroma prenesti na drugo sporočilo).
- Podpisanega dokumenta se ne da spremeniti.
- Podpisa se ne da zanikati (podpisnik ne more reči, da ni on podpisal dokumenta).

Kot že omenjeno, lahko za digitalno podpisovanje uporabimo kriptosistem javnih ključev, saj ustreza vsem zgoraj naštetim kriterijem. Ker pa je šifriranje dolgih datotek z asimetričnimi algoritmi prepočasno, se v praksi uporablja zgoščevalna funkcija, ki je bolj podrobno opisana v podpoglavju 2.2. Postopek podpisovanja je sledeč. S pomočjo zgoščevalne funkcije iz dokumenta izračunamo zgostitev ter jo zašifriramo z zasebim ključem. Rezultat šifriranja je podpis, ki se priloži originalnemu dokumentu. Prav tako je priloženo še digitalno potrdilo oziroma **certifikat**, ki je sestavljen iz javnega ključa in podatkov o lastniku. Potrjuje povezavo med osebo in njenim javnim ključem.

Preverjanje digitalnega podpisa se izvede v obratnem vrstnem redu. Najprej se digitalni podpis razčleni na podpisano zgostitev in podpisnikov certifikat. Nato se s pomočjo javnega ključa v certifikatu odšifrira zgostitev,

s čimer se pridobi originalno zgostitev. Iz sporočila/dokumenta se ponovno izračuna zgostitev, ki se jo primerja z originalno zgostitvijo. Če se ujemata, pomeni, da sporočilo ni bilo spremenjeno, odkar je bilo podpisano, in podpis lahko štejemo za veljaven.



Slika 2.4: Podpisovanje in šifriranje

Vredno je omeniti, da se podpisovanje in šifriranje sporočila ne izključujeta. Gre namreč za ločeni operaciji, ki ne uporabljata istega para ključev. Poglejmo si primer. Anita želi Bojanu poslati sporočilo z ljubezensko vsebino. Ker želi, da bo sporočilo lahko prebral samo Bojan in nihče drug, ga zašifrira z njegovim javnim ključem. Da pa se bo Bojan lahko prepričal, ali je sporočilo res Anitino, ga mora Anita tudi podpisati. Za to bo uporabila svoj zasebni ključ. Postopek je prikazan tudi na sliki 2.4.

2.1.2 RSA

Obstaja več asimetričnih algoritmov. V naši implementaciji smo uporabili algoritem RSA, zato sledi podrobnejši opis le-tega. Ime je dobil po svojih avtorjih: Ronu Rivestu, Adiju Shamirju in Leonardu Adlemanu, ki so ga razvili leta 1977. Algoritem temelji na faktorizaciji celih števil in je danes še vedno najpogosteje uporabljen, čeprav ga počasi zamenjujejo algoritmi na podlagi eliptičnih krivulj. Za lažje razumevanje je najprej potrebno definirati nekaj osnovnih matematičnih pojmov. V nadaljevanju bomo množico celih števil označili z $\mathbb{Z}$ in množico naravnih števil z $\mathbb{N}$.

Definicija 1: Število $a \in \mathbb{Z}$ deli število $b \in \mathbb{Z}$, če obstaja tak $k \in \mathbb{Z}$, da velja $b = ka$. Pišemo $a \mid b$. Število a, ki deli b, je faktor števila b, če je $a > 0$ in $a \notin \{1, b\}$.

Definicija 2: Če po zgornji definiciji celoštevilski k ne obstaja, imamo opravka z ostankom pri deljenju, kar opiše enačba $b = ka + r$, kjer velja $r \in \mathbb{Z}$. Ostanek lahko predstavimo tudi kot $b = r \mod a$.

Definicija 3: Število a je praštevilo, če se ga ne da zapisati kot produkt faktorjev, tj. če se ga ne da faktorizirati. Sicer je sestavljeno število.

Definicija 4: Evklidov algoritem je postopek za iskanje največjega skupnega delitelja dveh celih števil, $D(a, b)$. Števili a in b sta si tuji, če je njun največji skupni deljitelj 1 [17].

Sledi opis izvedbe osnovnih operacij za algoritem RSA:

- Generiranje para ključev. Izberemo različno veliki praštevili p in q. V praksi gre za večstomestni števili podobne dolžine. Izračunamo modul $n = pq$. Nato izberemo šifrirni eksponent e tako, da velja $D(e, (p-1)(q-1)) = 1$. Slednje pomeni, da sta si števili e in $(p-1)(q-1)$ tuji. Sledi še izračun odšifrirnega eksponenta d s pomočjo enačbe $ed = 1 \mod (p-1)(q-1)$. Javni ključ je potem (e, n), zasebni pa (d, n).

- Šifriranje. Čistopis m zašifriramo v tajnopis c s pomočjo javnega ključa po naslednji formuli (v praksi se sporočilo najprej razdeli na krajše bloke, ki se jih šifrira posamično ter na koncu združi):

$$c = m^e \mod n.$$

- Odšifriranje. Tajnopis c odšifriramo v čistopis m s tajnim ključem po sledeči formuli:

$$m = c^d \mod n \text{ [22] [12].}$$

Varnost algoritma RSA temelji na faktorizaciji števila n. Torej iskanje velikih praštevil p in q, če poznamo samo n. Za razliko od množenja ($p * q$) zaenkrat velja, da je razcep veliko zahtevnejša matematična operacija. Verjetno postane jasno, zakaj je dolžina ključev tako izredno pomembna.

2.1.3 Upravljanje ključev

Pri uporabi digitalnih podpisov in certifikatov se pojavi potreba po upravljanju le-teh. Vsak posameznik si lahko generira neomejeno število parov ključev. Vendar kako bomo vedeli, da je lastnik nekega ključa res tista oseba, za katero se izdaja? Potrebujemo dokaz oziroma overitev javnega ključa. Ta problem rešuje infrastruktura javnih ključev (angl. public key infrastructure - PKI), ki je nabor tehnologij, postopkov, naprav in predpisov, ki so potrebni za varno upravljanje zasebnih in javnih ključev. Poleg samega overjanja imetnikov ključev zajema tudi generiranje in hranjenje ključev, razdelitev, objavljanje ter preklic certifikatov. Poznana sta dva modela infrastrukture oziroma modela zaupanja.

Prvi je **hierarhičen model**, v katerem glavno vlogo opravljajo certifikatne agencije (angl. Certification Authority - CA), pravimo jim tudi overitelji javnih ključev, ki nastopajo kot centri zaupanja. Gre za centralizirano storitev/ustanovo, ki upravlja s ključi in sama določi politike delovanja ter izbiro tehnologij. Pred izdajo vsakega certifikata ustrezno poskrbi za preverjanje

identitete osebe, ki naroča izdajo certifikata. Certifikat nato podpiše s svojim zasebim ključem, kar služi kot overitev. Certifikatna agencija javno objavi izdane certifikate, tako aktivne kot preklicane. Na ta način lahko poiščemo certifikat določene osebe ali preverimo njegovo veljavnost. Takšen PKI model je definiran v standardu X.509 s strani IETF (Internet Engineering Task Force) in je danes najbolj pogosto uporabljen. Omenimo dve slabosti omenjene arhitekture. Zaupati moramo CA, da ne bo izdajala lažnih certifikatov. Zasebni ključ agencije mora biti pravilno varovan, saj zaupanje sloni na njem.

Drug model temelji na **mreži zaupanja** (angl. web of trust - WoT) in ga uporablja standard OpenPGP. Izpeljan je iz programa PGP (Pretty Good Privacy), ki ga je izdelal Phil Zimmermann leta 1991 in služi za varno pošiljanje elektronske pošte. V tem modelu zaupanja ni centralne agencije. Uporabniki drug drugemu overjajo/podpisujejo ključe. Na primer, ker Anita pozna Bojana in mu zaupa, podpiše njegov javni ključ. Ciril bi rad komuniciral z Bojanom, vendar ga osebno ne pozna. Je pa prijatelj z Anito in ker je ona podpisala Bojanov javni ključ, mu zaupa tudi Ciril. Seveda lahko Bojanov javni ključ podpiše več oseb, kar dodatno poveča zaupanje v njegovo identiteto. Bojan nato podpiše javni ključ Dejana in tako naprej. Na ta način se ustvarjajo povezave, preko katerih lahko zaupamo nekomu, čeprav ga ne poznamo. V mreži lahko obstajajo osebe, ki overijo mnogo certifikatov ter jim zaupa veliko število ljudi in tako delujejo v vlogi CA. Žal tudi takšen model zaupanja ni brez problemov. Učinkovitost je namreč odvisna od popularnosti uporabe standarda. V praksi se lahko zgodi, da do nekoga ne najdemo povezave zaupanja. Težave nastanejo tudi ob preklicu certifikata, saj moramo obvestiti vse, ki imajo shranjen naš certifikat.

Časovno žigosanje ima pomembno vlogo tudi v povezavi s certifikati. Recimo, da nam nepridipravi izmaknejo zasebni ključ. Če smo tatvino opazili, svoj certifikat kar se da hitro prekličemo. Vendar, kako bo nekdo vedel, ali je bil nek dokument podpisan z našim ključem pred ali po kraji? Če smo naše dokumente od podpisu tudi žigosali, se datum žigosanja in datum preklica enostavno primerja. Če ob podpisu dokumenta časovno ne žigosamo, obstaja

možnost lažne prijave kraje, če bi želeli svoj podpis kasneje zanikati.

2.2 Zgoščevalne funkcije

Za nadaljnje razumevanje diplomskega dela je potrebno razložiti še, kaj je enosmerna zgoščevalna funkcija. Torej gre za funkcijo, ki kot vhod sprejme poljubno dolg niz in ga preslika v rezultat fiksne dolžine. Rezultat imenujemo **zgostitev**, **povzetek** ali **prstni odtis** (angl. message digest, hash), ki je enolično definiran s strani vhodnega niza. To pomeni, da tudi najmanjša sprememba vhodnega niza drastično vpliva na zgostitev. Prav tako velja, da se isto sporočilo vedno preslika v enako zgostitev. Za funkcijo bomo rekli, da je **enosmerna**, kadar v doglednem času iz zgostitve ni mogoče povrniti vhodnega niza. Učinkovite enosmerne zgoščevalne funkcije imajo lastnost, da je izredno težko najti dve različni sporočili, ki bi ju preslikala v enako zgostitev.

Seveda v praksi stvari niso tako preproste. Hitro lahko ugotovimo, da za vhod obstaja ogromno različnih nizov, za katere imamo na voljo le omejeno število (končno množico) zgostitev. Omenjen pojav dobro opisuje Dirichletov princip. Slednji pravi, da če n predmetov razporedimo v r škatel, pri čemer je $n > r$, potem vsaj ena od škatel vsebuje več kot 1 predmet. Oglejmo si praktičen primer v kontekstu zgoščevalnih funkcij. Niz x definirajmo kot bitni zapis, torej zaporedje ničel (0) in enic (1), poljubne dolžine. Če bi zgostitve omejili na dolžino 10, bi imeli na voljo 2^{10} (1024) različnih možnosti. Zelo hitro bi našli drug vhodni niz x' z enako zgostitvijo. Temu pojavu pravimo trčenje (angl. collision) in ga matematično zapišemo kot $h(x) = h(x')$. Jasno je, da za vsako zgoščevalno funkcijo obstajajo trčenja. Vendar se z zadostno dolžino zgostitve da onemogočiti učinkovito iskanje takšnih trčenj s sedanjo tehnologijo. Pojavi se vprašanje, kolikšna je zadostna dolžina zgostitve. Danes ena najbolj razširjenih zgoščevalnih funkcij SHA-1, uporablja 160-bitno zgostitev. Žal pa se ji življenjska doba uporabe zaključuje, saj naj bi že obstajali (za enkrat še teoretični) napadi nanjo. Priporočena je uporaba

novejše različice SHA-2, ki ima možnost dolžin zgostitve 224, 256, 384 ali 512. Slednjo smo pri implementaciji uporabili tudi mi, njene zgostitve pa so 256-bitne. Omenimo, da je organizacija NIST (National Institute of Standards and Technology) po večletnem natečaju za iskanje algoritma SHA-3 izbrala zmagovalca [20]. Zmagal je algoritem z imenom *Keccak*, ki ima od predhodnikov družine SHA popolnoma drugačen pristop, t.i. gobasto konstrukcijo (angl. sponge construction). Slednja omogoča izredno prilagodljivost glede na želeno stopnjo varnosti in hitrost. Omenjeni algoritem je podrobno opisan v [15].

Poglavje 3

Sheme storitve časovni žig

3.1 Osnovni model

Osnovno rešitev si lahko predstavljamo kot digitalni sef. Kadar oseba želi časovno žigosati dokument, ga pošlje TSS. Storitev zabeleži datum in čas prejetja ter hrani kopijo dokumenta. Če želimo preveriti integriteto originala, ga enostavno primerjamo s kopijo, shranjeno v sefu. V primeru, da sta dokumenta identična, to dokazuje, da se vsebina od zabeleženega datuma ni spremenila. Tako enostavno rešitev je za resno uporabo potrebno še nekoliko dopolniti, saj ima veliko pomankljivosti:

1. Občutljive informacije, ki jih ne želimo razkriti, se nezaščiteno pošiljajo po omrežju. Prav tako lahko vsebino vidi vsak, ki ima ali si pridobi dostop do podatkovne shrambe. Želeli bi žigosanje brez razkritja samih informacij.

2. Storitev bi morala imeti ogromne kapacitete shranjevanja vseh kopij, ki bi se nabirale skozi celotno njeno delovanje. Ne smemo zanemariti tudi, da pošiljanje obsežnih vsebin lahko traja precej časa.

3. Če se kopija dokumenta, shranjena pri TSS, poškoduje ali izbriše, dokazovanje ni več mogoče.

4. Vprašljivo je zaupanje v storitev. Nič ji ne preprečuje, da manipulira s časom in tako lažno žigosa dokumente. Tudi podatke lahko enostavno spremeni in tako razveljavi žigosanje [9].

Kljub problemom opisana shema služi za lažje razumevanje koncepta in daje bralcu jasnejšo predstavo o zasnovi storitve in preprekah, ki se pojavijo v praksi. Prve tri probleme je možno dokaj enostavno rešiti z uporabo zgoščevalnih funkcij, opisanih v prejšnjem poglavju. Namesto pošiljanja celotnega dokumenta uporabnik najprej izračuna zgostitev ter jo nato pošlje storitvi. Na ta način podatki ostanejo nerazkriti, prav tako rešimo težave s prenosom in shranjevanjem. Bolj kompleksen je problem zaupanja, kateremu je namenjeno celotno 3. poglavje, saj zaenkrat ne poznamo enostavne rešitve.

3.2 Agencija za časovno žigosanje

V standardu X.509 je zraven CA definirana tudi agencija za časovno žigosanje (Time Stamping Authority - TSA). Protokol je natančneje opisan v dokumentu RFC3161 [10]. Gre za model storitve, ki temelji na infrastrukturi javnih ključev (PKI). Časovno žigosanje je v tem primeru zelo podobno klasičnemu digitalnemu podpisovanju. Agencija/storitev v vlogi zaupanja vredne tretje osebe poleg zgostitve, prejete od uporabnika, doda še trenutni čas ter oboje podpiše z zasebnim ključem. Gre torej za podpis zgostitve in časa s strani agencije, kar predstavlja **časovni žig**. Če želi agencija delovati po omenjenem protokolu, mora izpolnjevati naslednje zahteve:

- Uporablja zaupanja vreden izvor časa.
- Vsakemu časovnemu žigu pripne trenuten zaupanja vreden čas.
- Vsakemu časovnemu žigu pripne unikatno število.
- Ko prejme zahtevo s pravilnimi parametri, vedno generira časovni žig.
- Vsakemu časovnemu žigu priloži oznako politike za žigosanje, pod katero je časovni žig nastal.

- Časovno žigosa le zgostitev podatkov.
- Preveri, katera zgoščevalna funkcija je uporabljena v zahtevku ter ali je dolžina zgostitve pravilna.
- Ne preverja zgostitve na kakršen koli drug način razen dolžine.
- V žig ne vključuje nobene informacije o osebi, ki je časovni žig zahtevala.
- Podpisuje časovne žige le s ključem, ki je temu ekskluzivno namenjen in je to označeno tudi v certifikatu agencije.
- V žig doda vse dodatne informacije, ki so bile zahtevane s strani uporabnika.

```
Version: 1
Hash Algorithm: sha256
Message data:
    0000 - 64 74 c0 31 1e c8 de bd-16 5f 4c df 25 19 77 ac
    0010 - 17 3e 21 4a 72 61 35 08-f4 22 39 e7 9f f5 4a f5
Policy OID: unspecified
Nonce: 0xE7D21C76A1840AC3
Certificate required: yes
Extensions:
```

Slika 3.1: Zahtevek za časovno žigosanje

V protokolu je definirana tudi struktura zahtevka (angl. request) ter odgovora (angl. response). Na sliki 3.1 je prikazan primer veljavnega zahtevka. Vsebuje verzijo zahteve, *Hash algorithm* označuje, katera zgoščevalna funkcija je bila uporabljena, *Message data* pa predstavlja vsebino zgostitve. *Policy OID* definira, pod katero politiko naj se izvede časovno žigosanje. *Nonce* je naključno število, ki ga doda uporabnik, TSA pa prepiše v odgovor. S tem lahko uporabnik identificira, kateri odgovor pripada kateri zahtevi. Argument *Certificate required* določa, ali želimo, da TSA doda svoj certifikat. V polju *Extensions* pa se nahajajo vsi dodatni argumenti, ki bi jih želel vnesti uporabnik.

```
Status info:
Status: Granted.
Status description: unspecified
Failure info: unspecified

TST info:
Version: 1
Policy OID: 1.2.3.4.1
Hash Algorithm: sha256
Message data:
    0000 - 64 74 c0 31 1e c8 de bd-16 5f 4c df 25 19 77 ac
    0010 - 17 3e 21 4a 72 61 35 08-f4 22 39 e7 9f f5 4a f5
Serial number: 0x09
Time stamp: Mar  7 16:03:34 2015 GMT
Accuracy: 0x01 seconds, 0x01F4 millis, 0x64 micros
Ordering: yes
Nonce: 0x82AFDF73798741A2
TSA: DirName:/C=SI/ST=Slovenia/L=Ljubljana/O=Timestamper/
CN=Timestamper/emailAddress=vito.t@outlook.com
Extensions:
```

Slika 3.2: Odgovor časovnega žigosanja.

Odgovor je sestavljen iz informacije, ali je časovno žigosanje uspelo (*Status*), ter samega časovnega žiga (angl. Time Stamp Token - TST). V nasprotnem primeru (da žigosanje ni uspelo) je v polju *Failture info* opisano, kje je prišlo do napake. Veljaven odgovor, ki vsebuje časovni žig, je prikazan na sliki 3.2. Nekatera polja so enaka kot v zahtevku in vsebujejo enake vrednosti. *Serial number* je unikatno število časovnega žiga, ki ga je TSA kadarkoli izdala. *Timestamp* prikazuje čas, kdaj je bil žig ustvarjen in je predstavljen v formatu univerzalnega koordiniranega časa (Coordinated Universal Time - UTC). *Accuracy* se navezuje na prejšnje polje in opisuje odstopanje od dejanskega časa. Če je polje *Ordering* enako *yes*, to pomeni, da je časovne žige možno urediti po času njihovega nastanka. Polje *TSA* vsebuje informacije o agenciji, ki je izdala časovni žig. Agencija z odgovorom posreduje žig uporabniku in ga sama ni dolžna hraniti. Preverjanje časovnega žiga poteka na enak način kot preverjanje podpisa, torej s certifikatom. Glavna pomanjkljivost opisane sheme je, da ni mogoče zaznati zlorabe zasebnega ključa. V idealnih okoliščinah bi si želeli, da storitev ne more ustvariti lažnega žiga, tudi če bi to hotel narediti zlonamerni vzdrževalec.

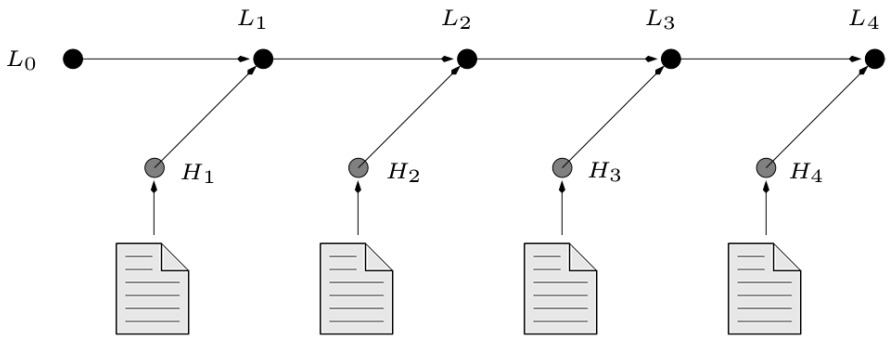
Potrebno je poudariti, da dokument RFC3161 ne določa načina transporta/izmenjave zahtevkov in odgovorov. To pomeni, da je komunikacija med uporabniki in agencijo odvisna od same implementacije. Omenimo nekaj možnih komunikacijskih protokolov, ki pridejo v poštev: SMTP (Simple Mail Transfer Protocol), HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protokol).

3.3 Porazdeljeno zaupanje

Ena izmed izboljšav sheme, opisane v prejšnjem podpoglavju, je uporaba **porazdeljenega zaupanja** (angl. distributed trust). Ideja je, da zahtevek za podpis pošljemo več strežnikom oziroma agencijam, ne samo eni. Sistem je sestavljen iz množice neodvisnih strežnikov, ki omogočajo podpisovanje. Uporabnik tako s pomočjo naključnega generatorja števil (angl. Random Number Generator - RNG) izmed n strežnikov izbere k, katerim pošlje zahtevek za žigosanje. Število možnosti izbire k-tih agencij med n-timi je enako binomskemu koeficientu $\binom{n}{k}$. Prednost takšnega decentraliziranega modela je predvsem v tem, da ni odvisen od ene same agencije. Večja izbira je dobra tudi za preprečevanje napadov, ki želijo ohromiti delovanje (angl. Denial of Service - DoS). Ko uporabnik prejme časovne žige, primerja čase njihovega nastanka, ki se morajo ujemati oziroma biti v določenem časovnem okviru. Zanimivi sta tudi vprašanji, koliko strežnikov je potrebnih v sistemu in kolikim bi se moral poslati zahtevek. Mi ju bomo pustili ob strani, saj pri naši implementaciji nismo uporabili omenjene sheme. Podrobnejša analiza je predstavljena v [4]. Omenimo samo, da bi napadalec, ki bi želel izdelati lažen časovni žig, moral pridobiti dostop do več strežnikov, kar je v praksi izredno težko. Poleg tega je podmnožica k strežnikov izbrana naključno in tako je nemogoče predvideti, katerim strežnikom bodo poslani zahtevki.

3.4 Veriženje žigov

Sledi predstavitev sheme za časovno žigosanje, ki ne sloni na infrastrukturi javnih ključev. Gre za tako imenovano **verižno žigosanje** (angl. linked timestamping), kjer so zgostitve dokumentov oziroma žigi povezani med seboj. Dodani so v podatkovno strukturo, iz katere je mogoče razbrati vrstni red dodajanja žigov in kakršno koli spreminjanje strukture je zaznano. Ena izmed takšnih podatkovnih struktur je linearna zgoščevalna veriga, prikazana na sliki 3.3.



Slika 3.3: Zgoščevalna veriga

Potrebno je opozoriti, da beseda **časovni žig** v tem kontekstu dobi drugačen pomen. Digitalni podpis v tem primeru nadomestimo z zgostitvijo, ki je rezultat zgoščevalne funkcije. Vsak posamezen žig je torej definiran kot $L_n = h(L_{n-1}|H_n|Meta - podatki)$, kjer h predstavlja zgoščevalno funkcijo, L_{n-1} prejšnji žig, H_n zgostitev n-tega dokumenta, $Meta - podatki$ pa vsebujejo čas, zaporedno številko žiga in druge informacije. Na tak način vsak naslednji žig potrjuje tako prejšnjega kot tudi celotno verigo.

Postopek žigosanja, pri zgoščevalni verigi, se za uporabnika bistveno ne spremeni. Tako kot pri ostalih shemah mora zgolj posredovati zgostitev dokumenta. Se pa razlikuje samo preverjanje žigov. Če želimo preveriti določen žig v verigi, moramo ponovno izračunati vse zgostitve oziroma žige, dokler

ne pridemo do žiga, kateremu zaupamo. V najslabšem primeru je to zadnji žig, katerega smo ustvarili sami. Potreba po ponovnem izračunu verige je namreč nujna, saj se na ta način prepričamo, da le-ta ni bila spremenjena. Zaradi kompleksnejšega preverjanja se pojavijo tri težave:

1. Ker za preverjanje žigov potrebujemo celotno verigo, je v praksi nesprejemljivo, da hranjenje žigov prepustimo uporabnikom. Podatki, ki so dodani z žigom L_n, bodo potrjeni šele z naslednjim žigom, L_{n+1}. Kateri žig naj torej hrani uporabnik, čigar podatki so vključeni v žig L_n? Prav tako lahko pričakujemo, da se bo kakšen žig tudi izgubil, kar bi pomenilo prekinjeno verigo in onemogočeno preverjanje.

2. Celotna veriga mora biti javno dostopna ter hkrati varno shranjena. Potrebno je poudariti, da hranjenje podatkovne strukture na enem samem mestu ni ravno priporočljivo.

3. Preverjanje verige, v katero se je nekaj let vsakodnevno dodajalo več sto ali celo tisoč žigov, je izredno počasen proces. Odvisno je seveda kje v verigi se žig nahaja, vendar v vsakem primeru velja asimptotična časovna zahtevnost $O(n)$, kjer je n število členov med preverjanim in zaupanja vrednim žigom.

Problem pod 1. točko smo dejansko že rešili, če predpostavimo, da hranjenje verige ne bomo prepustili uporabnikom. Največja ranljivost opisanega modela je torej sama podatkovna struktura (problem opisan pod 2. točko). Nanjo sta možni dve vrsti napadov, če izvzamemo napade na zgoščevalno funkcijo, saj bi to pomenilo ranljivost tudi ostalih shem. Napada sta sledeča:

Uničenje verige

Gre predvsem za napad na slabo implementacijo storitve. Z nepooblaščenim dostopom do podatkovne strukture bi jo lahko nekdo pokvaril ali pobrisal. Če je za hranjenje odgovorna storitev, je najenostavnejša zaščita že redno ustvarjanje varnostnih kopij. Ta ukrep je nenazadnje nujen že zaradi možnosti mehanske ali tehnične napake. Če pa

izguba kakšnega žiga ob omenjenem dogodku za nas ni sprejemljiva, je potrebno verigo hraniti na več strežnikih, ki se med sabo sinhronizirajo. Primer takšnega pristopa je veriga transakcij kriptovalute Bitcoin [14].

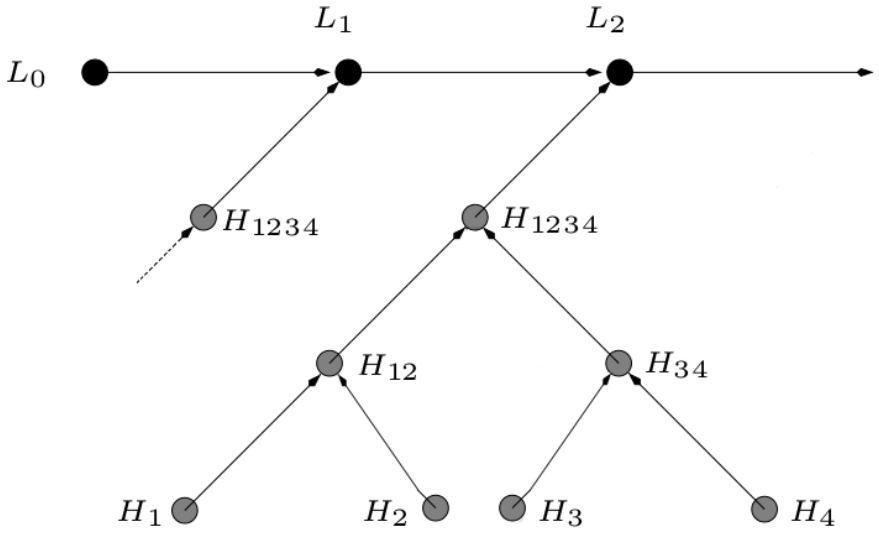
Zamenjava verige

Napadalec z dostopom do verige bi slednjo lahko spremenil tako, da bi zamenjal njen zadnji del. Torej, ne vmesnih členov, ampak zadnjih nekaj členov, saj tako veriga ostane veljavna. Takšen napad je zaznan šele, ko bi nekdo želel preveriti žig, ki je bil v odstranjenem/zamenjanem delu verige. Ena izmed rešitev je periodično objavljanje žigov v javnih medijih. Na primer, vsak teden v časopisu objavimo zadnji ustvarjen žig ali zgostitev celotne verige. Če bi kdo spremenil člene, ki se nahajajo pred objavljenim žigom, se ta ne bi več ujemal. Žal še zmeraj obstaja možnost, da nekdo zamenja žige zadnjega tedna, ki še niso bili potrjeni z objavo v mediju. Priporočeni so torej čim krajši intervali objavljanja. Druga rešitev problema je že prej omenjena decentralizacija podatkovne strukture. Ob sinhronizaciji med strežniki bi bila sprememba oziroma neujemanje verige takoj zaznano.

Problem časovne zahtevnosti, opisane pod točko 3, se delno lahko izboljša že z objavljanjem žigov v medijih, ki jih štejemo za zaupanja vredne, in tako skrajšamo dolžino verige, ki jo je treba ponovno izračunati. V praksi se žal izkaže, da starejši kot je žig, ki bi ga želeli preveriti, manjša je verjetnost, da imamo na voljo časopis iz tistega časa. Na primer, za preverjanje žiga izpred 14 dni nam včerajšnji časopis ne prihrani prav veliko časa. Za preverjanje žiga izpred 3 let pa najbrž nimamo tako starega časopisa. Bolj učinkovita rešitev je izboljšava podatkovne strukture.

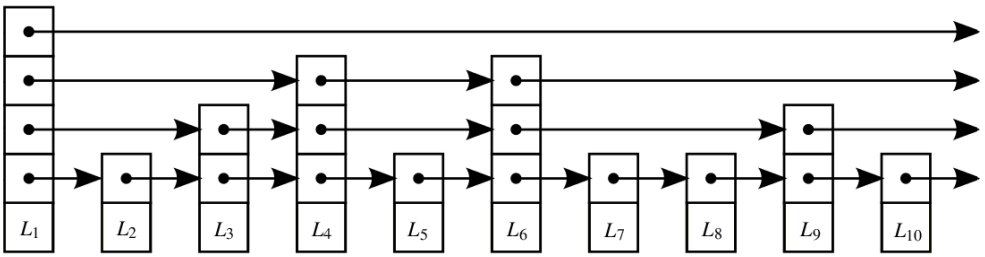
Klasičen linearen seznam lahko nadgradimo tako, da zgostitve dokumentov združujemo preden jih dodamo v verigo. Na primer, zgostitvi dveh različnih dokumetov združimo tako, da iz njiju ponovno izračunamo zgostitev ter le njo dodamo v verigo. Na ta način prepolovimo dolžino verige. Seveda nismo omejeni z združevanjem le dveh zgostitev. Z uporabo binarnih zgoščevalnih dreves (angl. binary hash tree), imenovanih tudi Merklova

drevesa, lahko združimo večje število zgostitev, v verigo pa dodamo le korensko vozlišče. Drevo je sestavljeno s pomočjo zgoščevalne funkcije. Listi predstavljajo zgostitve dokumentov, vmesna vozlišča pa so izračunana po principu $h(leva - zgostitev|desna - zgostitev)$. Dejansko gre za kombinacijo dveh struktur, ki ju uporabljamo na različnih nivojih, kot prikazuje slika 3.4. Združevanje zgostitev lahko poteka v časovno določenih intervalih ali je številsko omejeno. Na primer, združimo zgostitve, ki smo jih prejeli od uporabnikov v določeni uri, ali čakamo, dokler ne prejmemo 100 zahtevkov za žigosanje. Žal imata oba pristopa težavo. Če prejemamo premajhno količino zahtevkov na izbran časovni interval, pomeni, da struktura ni učinkovita, saj se lahko izrodi v klasičen linearen seznam. V drugem primeru, če čakamo na zadostno število zahtevkov, pa se lahko zgodi, da je časovna razlika med prvim in zadnjim prejetim zahtevkom v istem ciklu enostavno prevelika. Zelo pomembna je torej izbira, kako pogosto se zgostitve grupira in dodaja v verigo. Več o metodah združevanja zgostitev v binarna drevesa si lahko preberete v [5], [1] in [18].



Slika 3.4: Združevanje zgostitev v ciklih

Preskočni seznam (angl. skip list) je podatkovna struktura, ki za razliko od linearnega/verižnega seznama vsebuje dodatne povezave med členi. Strukturo prikazuje slika 3.5, kjer v našem primeru posamezen člen predstavlja žig. S povezovanjem več različno oddaljenih žigov izboljšamo hitrost preverjanja. Ni več potrebno računati vsakega žiga, ampak lahko izvajamo večje skoke med členi. Z optimalnim povezovanjem žigov, ki omogoča preverjanje po principu bisekcije, se časovna zahtevnost zmanjša na $O(\log_2 n)$ [6] [3]. Namesto preskočnega seznama lahko uporabimo tudi binarno povezovalno verigo, predstavljeno v [7] in [8]. Gre za strukturo, zelo podobno preskočnemu seznamu, ki med členi prav tako ustvarja več povezav in jo lahko definiramo/opišemo kot usmerjen acikličen graf.



Slika 3.5: Preskočni seznam

Poglavje 4

Implementacija rešitve

V tem poglavju je predstavljena zasnova naše rešitve, tehnični opis implementacije ter podrobnejše delovanje storitve. Različne sheme storitve za žigosanje, opisane v prejšnjem poglavju, se navadno izključujejo. Vendar to ni nujno. Nekaj komercialnih storitev na spletu že uporablja kombinacije različnih shem. Za takšen pristop smo se odločili tudi mi. Ena od shem, ki smo jo uporabili, je TSA, definirana v standardu X.509. Slednji smo dodali še podatkovno strukturo, in sicer linearno zgoščevalno verigo. Sistem tako omogoča dvojno vzporedno preverjanje časovne umeščenosti dokumentov. Največja pomanjkljivost modela, ki uporablja PKI je, da izdajanja lažnih žigov ni mogoče zaznati. To poskušamo odpraviti z dodajanjem zgostitev v verigo, ki je javno dostopna. Tako se lahko vsakdo prepriča, ali je morda kdaj prišlo do zlorabe, saj bi napadalec moral poleg lažnega podpisa spremeniti tudi podatkovno strukturo. Časovni žig, generiran z zasebnim ključem, mora hraniti uporabnik, žgoščevalno verigo pa hrani strežnik.

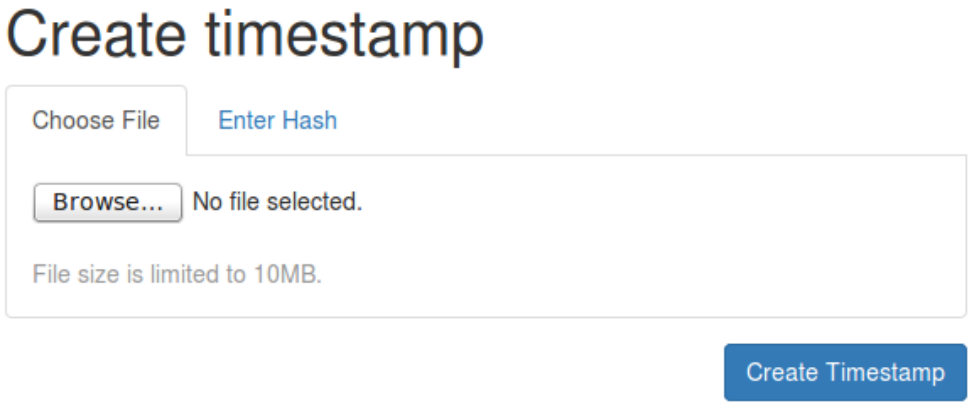
Za komunikacijski protokol smo se odločili uporabiti HTTP. Z drugimi besedami, uporabnik bo s storitvijo komuniciral preko spletnega uporabniškega vmesnika (spletne strani). Razlogi za to so praktične narave. Uporabniku ni treba nameščati nobene dodatne programske opreme. Potreben je le brskalnik, kar omogoča enostavno uporabo, saj so ga ljudje navajeni. Prav tako se nam ni treba ukvarjati s kompatibilnostjo na različnih operacijskih sistemih.

Brskalnike imajo nenazadnje tudi pametni telefoni. Odločili smo se, da storitev najprej implementiramo na testnem strežniku. Želimo namreč dodobra stestirati delovanje, preden se začne množično uporabljati. Testna različica je dostopna na spletnem naslovu `http://tsa.vito.tk`.

4.1 Uporabniški vmesnik

Spletna stran je zaradi prisotnosti tujih študentov na fakulteti v angleškem jeziku. Razdeljena je na tri podstrani oziroma zavihke: *Create timestamp*, *Verify* ter *Hashchain*. Prvi dve podstrani sta vertikalno razdeljeni na dve simetrični polovici. Na levi polovici se pri obeh nahaja enaka vsebina, in sicer opis storitve ter navodila za uporabo.

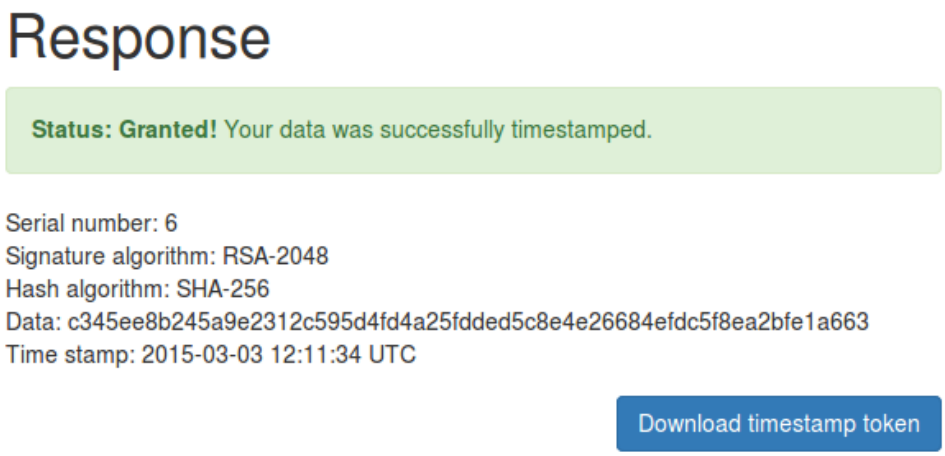
Na podstrani *Create timestamp*, ki je obenem privzeta domača stran, se na desni polovici nahaja obrazec, s pomočjo katerega uporabnik pošlje zahtevek za žigosanje (slika 4.1). Na voljo ima dve možnosti. S pomočjo izbirnega okna lahko izbere katero koli datoteko, ki jo želi žigosati, ali neposredno vnese datotečno zgostitev, če jo je izračunal sam.



Slika 4.1: Obrazec za izbiro datoteke.

Ob kliku na gumb *Create Timestamp* se podatki pošljejo strežniku, ki izvede žigosanje in odgovori, ali je le-to uspelo. Podrobnejši opis procesa

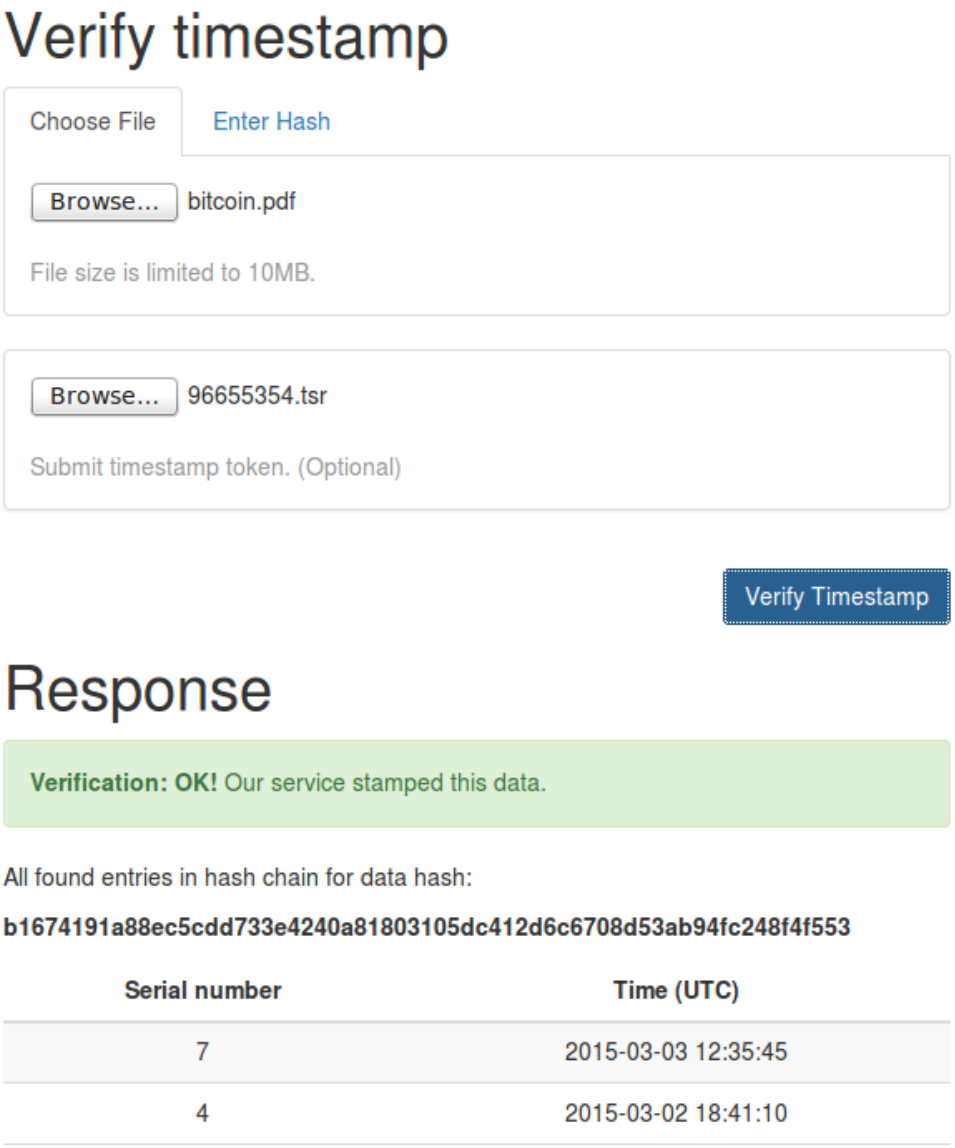
se nahaja v podpoglavju 4.3. Odgovor strežnika se prikaže v desnem spodnjem kotu (slika 4.2). V primeru, da je žigosanje uspelo, se izpišejo ključni podatki. *Serial number*, ki je zaporedna številka žiga. Algoritmi, ki so bili uporabljeni za izračun zgostitve in kreiranje žiga. *Data* predstavlja vrednost datotečne zgostitve, *Time stamp* pa dejanski čas podpisa. Prav tako se pojavi gumb *Download timestamp token*, s pomočjo katerega lahko uporabnik prenese/shrani časovni žig. Hramba žiga je obvezna, saj je potreben za kasnejše preverjanje (predstavlja del dokaza). V primeru, da žigosanje ni uspelo, se izpiše vzrok napake.



Slika 4.2: Odgovor časovnega žigosanja

Na podstrani *Verify*, ki je namenjena preverjanju datoteke oziroma zgostitve, se na desni polovici nahajata dva obrazca (slika 4.3). Prvi je identičen tistemu na podstrani *Create timestamp* in ima tudi enako funkcijo. Uporabnik z njegovo pomočjo izbere datoteko, ki jo želi preveriti. Drug obrazec je namenjen posredovanju časovnega žiga (TST). Ob kliku na gumb *Verify Timestamp* se strežniku pošlje ukaz, naj preveri, ali se posredovana datotečna zgostitev in časovni žig ujemata in ali je bil žig podpisan z zasebnim ključem storitve. Ta proces je možno izvesti tudi ročno, saj se na spletni strani nahaja povezava do certifikata storitve. Prav tako strežnik preveri tudi zgoščevalno

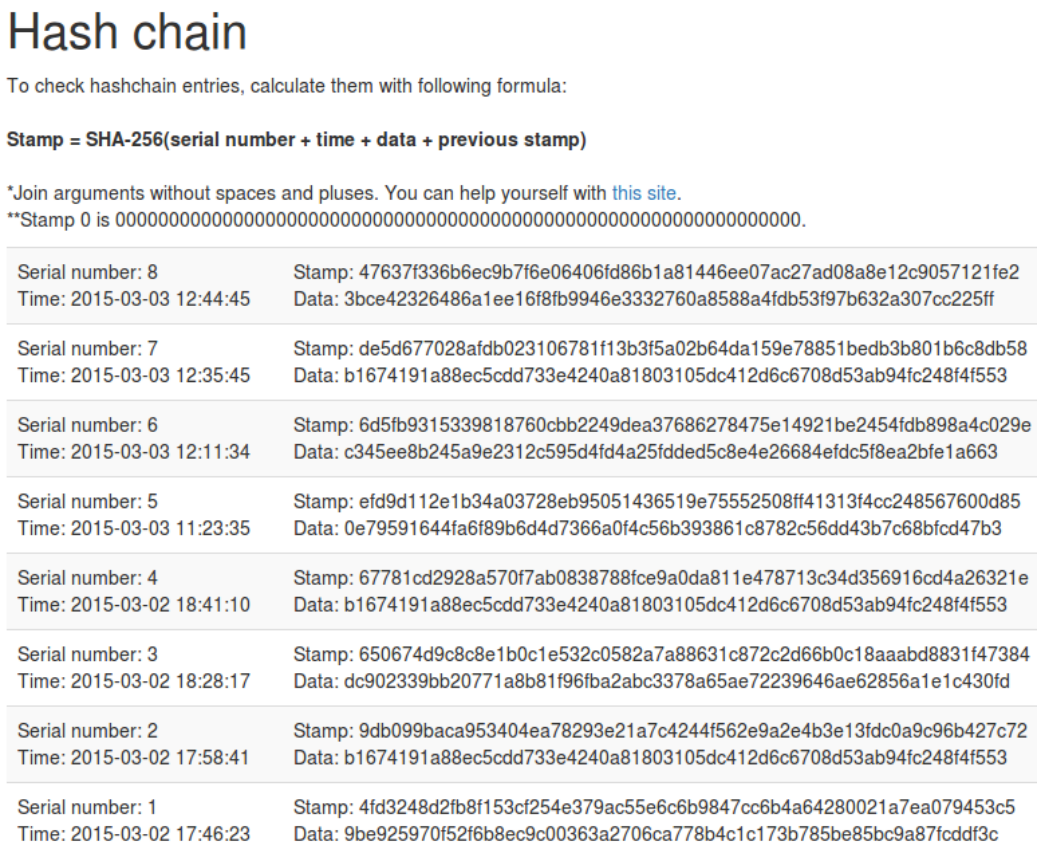
verigo in izpiše vse vnose, ki vsebujejo posredovano zgostitev.



Slika 4.3: Obrazec in odgovor preverjanja

Če se zgostitev nahaja v zgoščevalni verigi, kot je prikazano na sliki 4.3, to še ni zadosten dokaz, da je veriga dejansko veljavna. Spomnimo se podpoglavja 3.4, kjer smo napisali, da je potrebno ponovno izračunati del verige med preverjanim in zaupanja vrednim žigom. Temu je namenjena podstran

Hashchain, kjer so izpisani vsi členi verige. Preverjanje verige je žal potrebno izvesti ročno, saj ne moremo predvideti, kateremu žigu preverjajoča oseba zaupa. Za pomoč uporabniku je podana formula, po kateri lahko izračuna posamezne člene in rekonstruira verigo, ter povezava do spletne strani, ki omogoča računanje SHA-256 zgostitev.



Hash chain

To check hashchain entries, calculate them with following formula:

Stamp = SHA-256(serial number + time + data + previous stamp)

*Join arguments without spaces and pluses. You can help yourself with this site.
**Stamp 0 is 0000000000000000000000000000000000000000000000000000000000000000.

Serial number: 8 Time: 2015-03-03 12:44:45	Stamp: 47637f336b6ec9b7f6e06406fd86b1a81446ee07ac27ad08a8e12c9057121fe2 Data: 3bce42326486a1ee16f8fb9946e3332760a8588a4fdb53f97b632a307cc225ff
Serial number: 7 Time: 2015-03-03 12:35:45	Stamp: de5d677028afdb023106781f13b3f5a02b64da159e78851bedb3b801b6c8db58 Data: b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553
Serial number: 6 Time: 2015-03-03 12:11:34	Stamp: 6d5fb9315339818760cbb2249dea37686278475e14921be2454fdb898a4c029e Data: c345ee8b245a9e2312c595d4fd4a25fdded5c8e4e26684efdc5f8ea2bfe1a663
Serial number: 5 Time: 2015-03-03 11:23:35	Stamp: efd9d112e1b34a03728eb95051436519e75552508ff41313f4cc248567600d85 Data: 0e79591644fa6f89b6d4d7366a0f4c56b393861c8782c56dd43b7c68bfcd47b3
Serial number: 4 Time: 2015-03-02 18:41:10	Stamp: 67781cd2928a570f7ab0838788fce9a0da811e478713c34d356916cd4a26321e Data: b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553
Serial number: 3 Time: 2015-03-02 18:28:17	Stamp: 650674d9c8c8e1b0c1e532c0582a7a88631c872c2d66b0c18aaabd8831f47384 Data: dc902339bb20771a8b81f96fba2abc3378a65ae72239646ae62856a1e1c430fd
Serial number: 2 Time: 2015-03-02 17:58:41	Stamp: 9db099baca953404ea78293e21a7c4244f562e9a2e4b3e13fdc0a9c96b427c72 Data: b1674191a88ec5cdd733e4240a81803105dc412d6c6708d53ab94fc248f4f553
Serial number: 1 Time: 2015-03-02 17:46:23	Stamp: 4fd3248d2fb8f153cf254e379ac55e6c6b9847cc6b4a64280021a7ea079453c5 Data: 9be925970f52f6b8ec9c00363a2706ca778b4c1c173b785be85bc9a87fcddf3c

Slika 4.4: Podstran Hashchain

4.2 Nastavitve strežnika

Na strežnik smo namestili operacijski sistem Ubuntu. Gre za Linux distribucijo, ki je bila izbrana zaradi razširjenosti ter enostavne konfiguracije. Za programsko opremo, ki skrbi za streženje podatkov, smo izbrali Apache,

saj podpira skriptni programski jezik PHP, v katerem smo napisali del naše rešitve. Za ustvarjanje žigov z zasebnim ključem smo uporabili knjižnico OpenSSL, v podatkovni bazi MySQL pa hranimo žgoščevalno verigo. Vse potrebne programske pakete smo namestili s sledečim ukazom:

```
sudo apt-get install apache2 libapache2-mod-php5 mysql-server
libapache2-mod-auth-mysql php5-mysql ntp openssl tmpreaper.
```

4.2.1 Nastavitve Apache

Apache velja za najbolj razširjeno strežniško programsko opremo. Nastavitvene datoteke zanj se nahajajo v direktoriju `/etc/apache2` in njegovih poddirektorijih (`conf-available`, `sites-available`, `mods-available`). Glavna nastavitvena datoteka je `apache2.conf`. Prikazovanje naše strani smo omogočili z naslednjimi koraki. V poddirektoriju `sites-available` smo naredili kopijo datoteke `000-default.conf`, jo preimenovali v `tsa.conf` ter ustrezno popravili njeno vsebino (zapis 4.1). Na koncu smo v ukazni lupini izvedli ukaz `sudo a2ensite tsa`.

```
<VirtualHost *:80>
    ServerName tsa.vito.tk
    ServerAdmin vito.t@outlook.com
    DocumentRoot /var/www/tsa
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log
    ErrorDocument 404 /index.php
    ErrorDocument 403 /index.php
    <Directory /var/www/tsa>
        Options FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Zapis 4.1: Vsebina datoteke tsa.conf

Omenimo nalogi dveh pomembnih parametrov. `ServerName` določa, pod katero domeno bo stran dostopna. Saj v primeru, da imamo na istem strežniku več spletnih strani, Apache ponudi ustrezno. `DocumentRoot` definira, kje na strežniku se vsebina nahaja. Kot se da razbrati iz zgornjega izpisa, smo se odločili vsebino shraniti v direktoriju `/var/www/tsa`. Struktura slednjega je prikazana v nadaljevanju (zapis 4.2).

```
tsa
├── js
│   ├── bootstrap.js
│   └── main.js
├── css
│   ├── bootstrap.css
│   └── style.css
├── includes
│   ├── functions.php
│   └── .htaccess
├── response
├── index.php
├── verify.php
├── hashchain.php
├── ajax.php
└── timestamper.cer
```

Zapis 4.2: Struktura direktorija tsa

Za varno delovanje strežnika je bilo potrebno prilagoditi še nekaj nastavitev. V datoteki `security.conf` smo spremenili vrednost parametra `ServerSignature` na `Off` ter `ServerTokens` na vrednost `Prod`. S tem smo izključili pošiljanje/vključevanje informacij o operacijskem sistemu in programski opremi v odgovore strežnika. Gre za preventivni ukrep, da potencialnemu napadalcu posredujemo čim manj informacij o našem sistemu in mu tako otežimo delo ali ga celo odvrnemo od napada. Pomemben varnostni ukrep je tudi pravilna nastavitev pravic nad datotekami. Želimo namreč, da lahko datoteke ter direktorije bere, piše ali poganja le uporabnik, za katerega je to nujno. Tako smo za celoten direktorij `tsa` kot lastnika nastavili

administratorja, kateremu smo dodelili vse pravice. Za skupino smo določili `www-data`, v kateri je istoimenski uporabnik in je namenjen programu Apache. Slednji ima le pravice za branje. Ostali uporabniki do direktorija `tsa` nimajo dostopa (zapis 4.3).

```
owner: administrator
group: www-data
directories: drwx r-x ---
files:       -rw- r-- ---
```

Zapis 4.3: Politika pravic v direkotiju tsa

Izjema je poddirektorij `response`, v katerem uporabnik `www-data` ustvarja časovne žige, zato potrebuje pravico pisanja. Za redno čiščenje omenjenega direktorija smo uporabili orodje *tmpreaper*, ki počisti žige, starejše od ene ure. Skripto, ki izvede čiščenje, smo enostavno dodali v direktorij `/etc/cron.hourly/tmpreaper`.

4.2.2 Nastavitve MySQL in PHP

Nastavitev tolmača za jezik PHP (v datoteki `php.ini`) nismo spreminjali, saj že privzeto deluje v načinu `production`. Prav tako nismo spreminjali nastavitev programske opreme MySQL, ki se nahajajo v datoteki `my.cnf`. Potrebno je bilo ustvariti podatkovno bazo ter tabelo, v katero se shranjuje zgoščevalna veriga. To smo izvedli s pomočjo ukazne lupine in orodjem *mysql*, kamor smo z uporabnikom `root` vnesli naslednje ukaze:

```
create database tsa;
use tsa;
create table hashchain (
    serial INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    time VARCHAR NOT NULL
    hash VARCHAR(64) NOT NULL,
    previous_stamp VARCHAR(64) NOT NULL,
    stamp VARCHAR(64) NOT NULL,
);.
```

Nato sledijo še ukazi za kreiranje uporabnika, preko katerega bo z bazo upravljal Apache. Dodelili smo mu le tri osnovne pravice. V primeru zlorabe s tem uporabnikom ni mogoče spremeniti ali izbrisati zgoščevalne verige.

```
GRANT SELECT, INSERT, LOCK TABLES ON tsa.* TO
'www-data'@'localhost' IDENTIFIED BY 'password';
flush privileges;.
```

Ker se v direktoriju `includes` nahajajo občutljive informacije, med drugim tudi geslo za zgoraj kreiranega uporabnika, smo vanj dodali datoteko `.htaccess`. Z vsebino `deny from all` smo tako preprečili dostop do omenjenega direktorija. Datoteke `.htaccess` nam namreč omogočajo natančnejše nastavitve znotraj posameznega direktorija in prevladajo nad globalnimi nastavitvami.

4.2.3 Nastavitve OpenSSL

OpenSSL je odprtokodna knjižnica, ki implementira protokola SSL in TLS kot tudi glavne kriptosisteme. Za nas najpomembnejši ukaz orodja *openssl* je *ts*, s pomočjo katerega lahko ustvarjamo zahtevke za žigosanje ter žige po standardu X.509. V nastavitveni datoteki `openssl.cnf` je možno definirati različne parametre, ki bodo uporabljeni pri generiranju žigov (zapis 4.4).

```
[ tsa ]
default_tsa = tsa_config
[ tsa_config ]
dir = /etc/ssl/tsa
serial = $dir/serial
crypto_device = builtin
signer_cert = $dir/tsa.crt
certs = $dir/caroot.crt
signer_key = $dir/tsa.key
default_policy = tsa_policy1
digests = sha256
accuracy = secs:1, millisecs:500, microsecs:100
```

```
clock_precision_digits = 0
ordering = yes
tsa_name = yes
ess_cert_id_chain = no
```

Zapis 4.4: Vsebina datoteke openssl.cnf

Za ustvarjanje žigov smo uporabili 2048 bitov dolge ključe RSA, ki bodo, glede na poročilo organizacije NIST [16], varni še vsaj nekaj let. Certifikat storitve smo overili za nadaljnjih 5 let.

4.2.4 Pametne kartice

Pri vsaki uporabi zasebnih ključev je najbolj pomembno njihovo varovanje. Namesto hranjenja ključev na strežniku se pri sistemih, ki slonijo na PKI, uporabljajo izključno temu namenjene naprave HSM (Hardware Security Modul). Takšne naprave ponujajo varno generiranje in hranjenje ključev, hitro izvajajo kriptografske operacije ter podpirajo dodatne napredne varnostne mehanizme. V situacijah, kot je osebna uporaba in manjši sistemi, kjer ne potrebujemo visoke zmogljivosti, zadostujejo pametne kartice [2], ki služijo istemu namenu.

Od pasivnih (spominskih) kartic, ki se uporabljajo predvsem za hranjenje podatkov in vsebujejo le pomnilniške čipe, se pametne kartice razlikujejo v tem, da imajo lasten procesor. Gre za popolnoma samostojno napravo, lahko bi ji rekli tudi *mikroračunalnik*. Glavna prednost lastnega procesorja je, da omejuje dostop do podatkov na pomnilniku in zmore izvajati kriptografske operacije. Na kartici je tako mogoče generirati pare ključev, zasebni ključ pa je shranjen v delu pomnilnika, do katerega ima dostop le procesor. Komunikacija s kartico poteka preko čitalca, uporabnik pa mora za avtentikacijo navadno vnesti osebno identifikacijsko številko (Personal Identification Number - PIN). Več o razvoju kartic si lahko preberete v [13].

Pametne kartice je možno uporabiti tudi v povezavi s knjižnico OpenSSL. Potrebno je namestiti modul `engine_pkcs11`, ki je del projekta OpenSC in omogoča komunikacijo s kartico. Prav tako moramo ustrezno dopolniti

nastavitveno datoteko `openssl.cnf` (zapis 4.5).

```
engines = engine_section
[engine_section]
pkcs11 = pkcs11_section
[pkcs11_section]
engine_id = pkcs11
dynamic_path = /usr/lib/engines/engine_pkcs11.so
MODULE_PATH = /usr/lib/opensc-pkcs11.so
PIN = ****
init = 0
```

Zapis 4.5: Dodatna vsebina datoteke openssl.cnf

4.2.5 Vir časa

V diplomskem delu do zdaj še ni bilo prav dosti napisano o zagotavljanju točnega časa. Ne smemo pozabiti nanj, saj je brez preciznega časa žigosanje nesmiselno. V definiciji protokola TSA je že bilo omenjeno, da je potreben zaupanja vreden vir časa. Najnatančneje čas merijo atomske ure, ki slonijo na merjenju frekvence resonančnega prehoda elektronov določenih atomov. Takšne ure so predrage, da bi si jih lahko privoščili za posamezno storitev. Zato obratujejo kot neke vrste časovne postaje in ponujajo točen čas preko različnih komunikacijskih kanalov. Sinhronizacijo ure preko interneta omogoča protokol NTP (Network Time Protocol)[11], katerega smo uporabili tudi mi, saj je konfiguracija najenostavnejša.

Protokol deluje tako, da naš strežnik pošlje zahtevek enemu ali več časovnim strežnikom. Ko prejme informacijo o času, izračuna povprečno vrednost ter upošteva časovni zamik zaradi komunikacije med strežniki. Lokalni/sistemski čas nato ustrezno popravi. Ker komunikacija poteka preko protokola UDP, zaradi čim manjše zakasnitve, to omogoča napadalcu modifikacijo paketov. Da ne bi prišlo do zlorabe, protokol dopušča le zelo majhne spremembe sistemske ure. V primeru večjega odstopanja bi se napačen čas nenazadnje opazil tudi v zgoščevalni verigi. Prav tako je za pridobitev časa smotrno

uporabljati več strežnikov zaradi večje zanesljivosti. Naslove strežnikov, od katerih bi želeli prejemati informacijo o času, vnesemo v nastavitveno datoteko `/etc/ntp.conf`. Mi smo se odločili za uporabo štirih strežnikov izmed trenutno dvajsetih [21], kolikor se jih nahaja v Sloveniji. Za definiranje le-teh smo uporabili t. i. strežniški bazen, iz katerega so strežniki vsakič naključno izbrani iz celotne množice. V našem primeru gre za izbiro štirih iz bazena `si.pool.ntp.org`.

4.3 Delovanje in programska koda

4.3.1 Spletni vmesnik

Za oblikovanje spletne strani smo si pomagali z ogrodjem Bootstrap. Olajša namreč delo s HTML, CSS in JavaScript jeziki, kar pripomore k hitrejši izdelavi spletnega vmesnika. Podpira tudi t. i. odziven dizajn, ki poskrbi, da spletna stran ohranja strukturo na različnih elektronskih napravah. Za elegantnejšo komunikacijo s strežnikom smo uporabili tehnologijo AJAX (Asynchronous JavaScript and XML), ki omogoča osveževanje in prikazovanje nove vsebine brez potrebe nalaganja celotne spletne strani.

```
function form_submit(action, hash){
    var formData = new FormData();
    formData.append("action", action);
    formData.append("hash", hash);
    if (action == "verify"){
        var file = $jq("#token").files[0];
        formData.append("token", file);
    }
    $jq.ajax({
        url: "ajax.php",
        type: "POST",
        data : formData,
        processData: false,
        contentType: false,
        success : function(data){
```

```
            $jq(".response").html(data);
        }
    });
}
function calculate_hash(){
    var file = $jq("#files").files[0];
    var reader = new FileReader();
    reader.onloadend = function(evt){
        if (evt.target.readyState == FileReader.DONE){
            var sha256 = CryptoJS.algo.SHA256.create();
            sha256.update(CryptoJS.parse(evt.target.result));
            form_submit(action, sha256.finalize());
        }
    };
    reader.readAsBinaryString(file);
}
```

Zapis 4.6: Del kode JavaScript

Ker na spletni strani za žigosanje in preverjanje uporabljamo isti obrazec, se tudi ob kliku na gumba *Create Timestamp* ali *Verify Timestamp* v ozadju sproži enak proces. V primeru, da je uporabnik v obrazcu izbral datoteko, se s pomočjo knjižnice CryptoJS lokalno izračuna njena zgostitev. Strežniku se nato pošlje zahtevek POST. Vsebuje parameter *hash* z vrednostjo zgostitve ter parameter *action*, ki ima lahko vrednost *stamp* ali *verify*. Če gre za proces preverjanja, se zahtevku POST pripne še datoteka s časovnim žigom. Osrednji funkciji opisanega procesa sta prikazani v zapisu 4.6.

4.3.2 Postopek žigosanja

Kot že omenjeno, smo za procesiranje zahtevkov na strežniku uporabili jezik PHP. Ob prejetem zahtevku se najprej preveri vrednosti parametrov. Ne želimo namreč, da nekdo pošlje vrinjeno programsko kodo, ki bi se lahko izvedla. Prav tako se dodatno preveri še veljavnost zgostitve. S pomočjo knjižnice OpenSSL se nato generira zahtevek za žigosanje in ustvari žig (TST). Da ne bi prišlo do zmešnjave, imenujmo žig, ustvarjen z zasebnim ključem **pri-**

marni žig. Funkcije, ki poskrbijo za kreiranje primarnega žiga, so prikazane v nadaljevanju (zapis 4.7).

```
function filterInput($data){
    return htmlspecialchars(stripslashes(trim($data)));
}
function validateHash($hash){
    if (strlen($hash) !== 64 || !ctype_xdigit($hash))
        throw new Exception("Invalid hash.");
    return $hash;
}
function createRequest($hash){
    $requestfile = createTempFile();
    exec("openssl ts -query -digest ".$hash." -cert -sha256 -out ".
    $requestfile);
    return $requestfile;
}
function createResponse($requestfile){
    if (!file_exists($requestfile))
        throw new Exception("Requestfile not found");
    $responsefile = createTempFile();
    exec("openssl ts -reply -config /etc/ssl/openssl.cnf -queryfile "
    $requestfile." -out "$responsefile);
    return $responsefile;
}
```

Zapis 4.7: Prvi del kode PHP namenjene žigosanju

Sledi drugi del žigosanja, dodajanje novega člena v zgoščevalno verigo. Žig, ustvarjen na podlagi zgostitve, poimenujmo **sekundarni žig**. Iz podatkovne tabele *hashchain* se najprej prebere zadnji dodani žig. Slednjega skupaj s podatki iz primarnega žiga uporabimo za izračun novega sekundarnega žiga. Tudi tukaj se uporabi zgoščevana funkcija SHA-256. V podatkovno tabelo se doda vnos z naslednjimi atributi: `serijska številka, čas, zgostitev dokumenta, prejšnji žig, žig`. Ker je novonastali žig odvisen od vrednosti zadnjega v podatkovni tabeli, je treba pred poizvedbo tabelo zakleniti. V nasprotnem primeru bi se zaradi večnitnega delovanja programa Apache

lahko pojavilo tvegano stanje (angl. race condition). Omenjeni pojav bi nastopil, če bi več procesov žigosanja istočasno dostopalo do podatkovne baze in uporabilo isto vrednost prejšnjega žiga, kar je seveda narobe. Osrednji del programske kode je prikazan v zapisu 4.8.

```
function insertTS($db, $time, $hash, $previous_stamp, $stamp){
    $statement = $db->prepare("INSERT INTO hashchain (time, hash,
    previousstamp, stamp) VALUES (?, ?, ?, ?)");
    $statement->bind_param('ssss', $time, $hash, $previous_stamp, $stamp);
    $statement->execute();
}
function getLastTS($db){
    $stamp ="";
    $statement = $db->prepare("SELECT stamp FROM hashchain ORDER BY serial
     DESC LIMIT 1");
    $statement->execute();
    $statement->bind_result($stamp);
    $statement->fetch();
    return $stamp;
}
$db = new mysqli("localhost", "www-data", $password, "tsa");
$db->query("LOCK TABLES hashchain WRITE");
$previousstamp = getLastTS($db);
$stamp = hash("sha256", $serial.$time.$hash.$previousstamp);
insertTS($db, $time, $hash, $previousstamp, $stamp);
$db->query("UNLOCK TABLES");
$db->close();
```

Zapis 4.8: Drugi del kode PHP namenjene žigosanju

4.3.3 Postopek preverjanja

Tudi pri postopku preverjanja se najprej preveri vrednosti parametrov zahtevka POST. Parameter *action* mora imeti vrednost *verify*, zgostitev pa pravilno dolžino in vsebovati le heksadecimalne znake. Prav tako se preveri posredovano datoteko, ki vsebuje primarni žig. Biti mora ustrezne velikosti ter imeti končnico *.tsr*. Sledi postopek preverjanja primarnega žiga s

pomočjo knjižnice OpenSSL. Ujemati se mora zgostitev dokumenta in digitalni podpis, s katerim je bil žig ustvarjen. Nato se v podatkovni tabeli poišče vse vnose (sekundarne žige), ki vsebujejo posredovano zgostitev. Slednji se izpišejo uporabniku, da lahko ročno izvede drug del preverjanja (rekonstrukcijo verige). Sledi prikaz programske kode za proces preverjanja (zapis 4.9).

```
function uploadToken($file){
    $tokenfile = "/var/www/tsa/response/" . basename($file["name"]);
    if(validateToken($file)){
        move_uploaded_file($file["tmp_name"], $tokenfile)
    }
    return $tokenfile;
}
function verifyToken($hash, $tokenfile){
    $tsa_cert = "/var/www/tsa/timestamper.cer";
    $cmd = "openssl ts -verify -digest ".$hash." -in ".$tokenfile." -
    CAfile ".$tsa_cert;
    $retarray = array();
    exec($cmd, $retarray);
    if(strtolower(trim($retarray[0])) != "verification: ok"){
        throw new Exception("Verification failed.");
    }
    return true;
}
$db = new mysqli("localhost", "www-data", $password, "tsa");
$result = array();
$statement = $db->prepare("SELECT * FROM hashchain WHERE hash = ? ORDER BY
     serial DESC");
$statement->bind_param('s', $hash);
$statement->execute();
$statement->bind_result($result[0], $result[1], $result[2], $result[3],
    $result[4]);
while($statement->fetch()){
    printEntry($result[0], $result[1]);
}
$db->close();
```

Zapis 4.9: Del kode PHP namenjene preverjanju žiga

Poglavje 5

Zaključek

V okviru diplomskega dela smo razvili delujočo storitev časovni žig. Temelji na dveh različnih shemah zaupanja, ki dopolnjujeta slabosti drug druge. Podpisovanje zgostitev dokumentov z zasebnim ključem smo podkrepili z dodajanjem zgostitev v zgoščevalno verigo, ki je javno dostopna. Velik del pozornosti smo namenili strežniškim nastavitvam za varno delovanje. Za komunikacijo s storitvijo smo implementirali spletni uporabniški vmesnik, kar omogoča enostavno uporabo. Ciljna skupina uporabnikov so namreč študentje fakultete.

Med razvojem storitve smo prišli do ugotovitve, da je postavitev varnega računalniškega sistema izredno kompleksen problem. Ogromno je področij, na katera je treba biti pozoren, saj v primeru odpovedi enega izmed njih postane ranljiv celoten sistem. Varnosti se je torej treba lotiti sistematično. Potrebno je zagotoviti, da kriptosistemi slonijo na preverjeni matematični podlagi. Algoritme in programe je nato treba pravilno implemetirati. Ker to počnemo ljudje, se razumljivo pojavljajo napake. V veliki večini je ravno programska koda najšibkejši člen varnosti, kjer se odkrije največ ranljivosti. Preverjanje, pregledovanje in posodobitev programske opreme je tako ključnega pomena. Ne smemo pozabiti niti na fizično varnost in nadzor infrastrukture, na kateri tečejo aplikacije.

Pred širšo uporabo naše storitve bo potrebno rešiti še nekaj odprtih pro-

blemov. Strežnik, na katerem bo tekla naša aplikacija mora biti v temu namenjenem prostoru z omejenim dostopom. Poskrbeti bo treba za redno izdelovanje varnostnih kopij ter vpeljati mehanizme za preprečitev napadov DoS. Prav tako ne smemo pozabiti na redno objavljanje členov zgoščevalne verige v javnem mediju. Za slednjega bi lahko uporabili revijo FRIK, ki jo izdajajo študentje fakultete.

Čeprav je časovno žigosanje že poznan proces, ima kljub temu še ogromen potencial za vrsto različnih namenov uporabe, predvsem v večjih sistemih in organizacijah. Lahko bi ga vpeljali v različne državne ustanove, kot je na primer zdravstvo (za beleženje čakalnih vrst) ali sodstvo (za žigosanje vseh vrst dokumentov). Če domišljiji pustimo prosto pot, bi podatkovno strukturo lahko uporabljali kot javni globalni dnevnik spleta. Vanjo bi se namreč beležilo vse dogodke/aktivnosti na spletu.

Literatura

[1] D. Bayer, S. Haber, W. S. Stornetta, "Improving the Efficiency and Reliability of Digital Time-Stamping", *Sequences II*, str. 329-334, Springer-Verlag, 1993.

[2] G. Bernabé in N. Clarke, "Study of RSA Performance in Java Cards", *Computer and Information Security.*

[3] K. Blibech, A. Gabillon, "A New Timestamping Scheme Based on Skip Lists", *Computational Science and Its Applications - ICCSA*, str. 395-405, Springer-Verlag, 2006.

[4] A. Bonnecaze, P. Liardet, A. Gabillon in K. Blibech, "A Distributed Time Stamping Scheme", v zborniku *IEEE conference on Signal Image Technology and Internet based Systems*, 2005.

[5] A. Buldas, A. Kroonmaa in R. Laanoja, "Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees", *Secure IT Systems*, str. 313-320, Springer-Verlag, 2013.

[6] A. Buldas in H. Lipmaa, "Digital Signatures, Timestamping and the Corresponding Infrastructure", jan. 1998.

[7] A. Buldas, P. Laud, H. Lipmaa in J. Villemson, "Time-Stamping with Binary Linking Schemes", *Advances in Cryptology — CRYPTO '98*, str. 486-501, Springer-Verlag, 1998.

[8] A. Buldas in P. Laud, "New linking schemes for digital time-stamping", v zbirniku *First International Conference on Information Security and Cryptology*, 1998.

[9] S. Haber in W. S. Stornetta, "How to Time-Stamp a Digital Document", *Journal of Cryptology*, 1991.

[10] IETF, "Internet X.509 Public Key Infrastructure Timestamp Protocol", *RFC3161*, avg. 2001, dostopno na http://www.ietf.org/rfc/rfc3161.txt, obiskano 12. dec. 2014.

[11] IETF, "Network Time Protocol", *RFC5905*, jun. 2010, dostopno na https://www.ietf.org/rfc/rfc5905.txt, obiskano 15. jan. 2015.

[12] A. Jurišić, "Kriptografija in računalniška varnost", učno gradivo isto imenskega predmeta na FRI, 2013.

[13] A. Jurišić in A. Trojar, "Pametna kartica", jan. 1997.

[14] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System".

[15] NIST, "SHA-3 Standard: Permutation-Based Hash andExtendable-Output Functions", maj 2014.

[16] NIST, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths", jan. 2011.

[17] P. Nose, "Varnostna analiza protokolov za overjen dogovor o ključu in shem za digitalni podpis", doktorska disertacija na FRI, 2014.

[18] B. Preneel, B. V. Rompay, J. J. Quisquater, H. Massias, J. S. Avila, "Design of a timestamping system", tehnično poročilo projekta TIME-SEC.

[19] K. Schmeh, *Cryptography and Public Key Infrastructure on the Internet*, Wiley, jun. 2003.

[20] Spletna podstran ogranizacije NIST o razvoju SHA-3, dostopno na http://csrc.nist.gov/groups/ST/hash/index.html, obiskano 2. feb. 2015.

[21] Spletna stran s seznamom strežnikov NTP v Sloveniji, dostopno na http://www.pool.ntp.org/zone/si, obiskano 23. feb. 2015

[22] A. Tolič, "Kriptografija e-volitev", diplomsko delo na FRI, 2011.

[23] Zakon o elektronskem poslovanju in elektronskem podpisu, Uradni list RS št. 98/2004, http://www.uradni-list.si/1/content?id=51150, obiskano 20. feb. 2015.