Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Blaž Kostanjšek

Uporaba verige blokov za zagotavljanje zaupnosti in integritete podatkov v obstoječih sistemih

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Mojca Ciglarič

Ljubljana, 2017

Zahvala

Na tem mestu bi se rad zahvalil mentorici dr. Mojci Ciglarič za vodenje pri izdelavi dela. Hvala tudi mojim bližnjim, posebej Živi, za vztrajno podporo na celotni študijski poti.

Blaž Kostanjšek, 2017

Contents

Povzetek

Naslov: Uporaba verige blokov za zagotavljanje zaupnosti in integritete podatkov v obstoječih sistemih

Ko govorimo o zaupnosti in integriteti podatkov, smo uporabniki v tradicionalnih sistemih vedno primorani zaupati centralni avtoriteti. S prihodom prvih kripto-valut in hitrim porastom uporabe verig blokov tudi na ostalih področjih se pojavi možnost novega pristopa k reševanju te problematike. V delu tako predlagamo nov način reševanja zaupanja in integritete podatkov, ki temelji na uporabi verige blokov in za svoje delovanje ne potrebuje zaupanja vredne centralne avtoritete. Tekom naloge postopoma razvijemo načrt rešitve, z implementacijo predlagane rešitve izvedemo študijo izvedljivosti takšnega sistema ter glede na omejitve implementirane rešitve ocenimo smiselnost uporabe v obstoječih sistemih.

Ključne besede

veriga blokov, pametne pogodbe, zaupnost, integriteta podatkov, omrežja vsak z vsakim

Abstract

Title: Using Blockchain to provide confidentiality and data integrity in existing systems

When it comes to confidentiality and data integrity, in traditional systems users always need to trust central authorities. The arrival of first cryptocurrencies and the rapid increase in use of blockchains on non-financial areas opens the possibility of a new approach to solving this problem. In this work, we propose a new way of providing confidentiality and data integrity based on blockchain, which does not require a trusted central authority. We gradually develop an idea of solution and then make a proof of concept by implementing the proposed solution. Considering the limitations of the implemented solution we also evaluate where the integration of implemented solution might be reasonable.

Keywords

blockchain, smart contracts, confidentiality, data integrity, peer to peer networks

Poglavje 1

Uvod

1.1 Motivacija

S prihodom prve elektronske kripto valute Bitcoin se je pojavila nova porazdeljena podatkovna struktura imenovana veriga blokov (blockchain) [1], ki omogoča hranjenje podatkov v omrežju tipa vsak z vsakim, ter pri tem zagotavlja nespremenljivost podatkov, ne da bi za delovanje potrebovali centralno avtoriteto, ki bi ji bilo potrebno zaupati. Zaradi teh lastnosti se je uporaba verige blokov hitro razširila tudi na področja, ki niso povezana z elektronskimi valutami [2][3][4], hkrati pa uporaba verige blokov na ostalih področjih predstavlja pomemben korak pri njenem nadaljnjem razvoju [5].

V magistrski nalogi tako želimo uporabo verige blokov razširiti na področje zaupnosti in integritete podatkov. V veliki večini smo namreč uporabniki raznih spletnih storitev, ter samega interneta, primorani zaupati ponudnikom storitev, da z našimi podatki upravljajo skladno s predpisi - da objavljene vsebine ne spreminjajo in ne dopuščajo nepooblaščenih vpogledov. V nalogi želimo tako pokazati, da se reševanja takšne problematike lahko lotimo tudi z uporabo te nove tehnologije.

1.2 Predlog rešitve

Da bi rešili ta problem, bomo pripravili dokaz izvedljivosti uporabe verige blokov v namen zagotavljanja zaupnosti ter integritete podatkov v okoljih, kjer z našimi podatki upravlja ponudnik storitve, pri tem pa bomo ponudnikovo storitev zgolj nadgradili z verigo blokov ter s tem samo storitev pustili nespremenjeno, kar bi omogočalo enostavno vključitev razvite funkcionalnosti v že obstoječe sisteme.

Ker gre pri zaupnosti in integriteti podatkov za precej široko področje, se pri izdelavi rešitve odločimo za bolj specifični rešitvi: pri zagotavljanju integritete podatkov bomo z uporabo verige blokov izdelali rešitev za shranjevanje izvlečkov vsebine, pri zagotavljanju zaupnosti pa nameravamo razviti sistem za dodeljevanje pravic vpogleda.

1.3 Prispevki

Predvideni prispevki magistrske naloge so:

- Predlog načina uporabe verige blokov v namen povečanja zaupnosti in integritete podatkov v že obstoječih sistemih.
- Študija izvedljivosti predlaganega sistema ter implementacija takšne rešitve na primeru obstoječe spletne aplikacije družabnega omrežja.
- Pregled prednosti, slabosti in omejitev predlagane rešitve predvsem glede na omejitve same verige blokov, ter posledično smiselnost vpeljave takšne rešitve v obstoječ sistem.

1.4 Struktura naloge

V poglavju 2 na kratko predstavimo koncepte in tehnologije, uporabljene v magistrskem delu, ne le verigo blokov, temveč tudi tehnologije ki so ključne za razumevanje celotnega dela. V poglavju 3 predlagamo načrt rešitev za

zagotavljanje zaupnosti ter integritete podatkov z uporabo verige blokov. V poglavju 4 pregledamo ponudnike verige blokov ter izberemo verigo blokov, ki jo bomo uporabili pri implementaciji. V poglavju 5 nato sledi implementacija predlaganega načrta, v poglavju 6 pa izdelamo pregled omejitev implementirane rešitve.

Poglavje 2

Teoretično ozadje in sorodne raziskave

V tem poglavju na kratko predstavimo in opišemo koncepte, tehnologije in orodja, katerih razumevanje je ključnega pomena za razumevanje magistrskega dela. Začnemo z verigo blokov, kjer predstavimo ključne lastnosti, ki so vodile v idejo o rešitvi, ki jo predlagamo v nalogi. Nato predstavimo tehnologije, ki so se razvile na sami verigi blokov, kjer poudarimo predvsem pametne pogodbe, ki so pomembno orodje pri realizaciji drugega dela magistrske naloge. Poleg tehnologij pa opišemo tudi principe zagotavljanja zaupnosti ter integritete podatkov, na kratko predstavimo kriptografijo javnih ključev, ter pomembnejše zgoščevalne funkcije in kriptirne algoritme, ki jih uporabljamo kasneje v nalogi.

Da bi zagotovili primerno stopnjo zaupnosti ter integritete podatkov, je pomembno, da veriga blokov, s katero želimo reševati problem, nudi zahtevano stopnjo varnosti. Ker pa ni vsaka veriga blokov tudi varna veriga blokov, prikažemo, katere so varnostne pomanjkljivosti verige blokov in na kakšen način jih rešujemo.

Ob koncu poglavja naredimo tudi strnjen pregled sorodnih raziskav, kjer prikažemo dosedanje delo v širšem pomenu uporabe verige blokov, poudarimo pa seveda tudi raziskave, ki se dotikajo našega problema - zagotavljanje

zaupnosti in integritete podatkov s pomočjo verige blokov.

2.1 Veriga blokov

Začetki verige blokov (ang. blockchain) segajo v leto 2008, ko je avtor, znan pod psevdonimom Satoshi Nakamoto, objavil članek [1], v katerem podrobno opiše inovativen sistem elektronskega denarja imenovanega Bitcoin, ki deluje na principu omrežja vsak z vsakim (ang. peer to peer) in omogoča spletna plačila brez potrebe po posredniku - centralni avtoriteti, ki bi ji bilo potrebno zaupati. Zanimivo je, da še danes ne vemo, kdo je oseba (ali morda skupina ljudi), ki stoji za psevdonimom Satoshi Nakamoto.

Predlagan plačilni sistem Bitcoin je bil revolucionaren in se je hitro prijel, vendar pa je kmalu postalo jasno, da je pravzaprav način delovanja, ki stoji za kripto valuto, tista novost, ki je zares revolucionarna. Razlog se skriva predvsem v dejstvu, da je do takrat celotna digitalna ekonomija temeljila na zaupanju določeni centralni avtoriteti. Še več, v to lahko vključimo tudi področja, ki niso povezana z ekonomijo. Vse spletne aktivnosti, ki jih kot uporabniki izvajamo, temeljijo na zaupanju osebi ali organizaciji, za katerega predpostavimo, da je vreden zaupanja - naj si bo to organ za potrjevanje (ang. certification authority - CA), ki nam zagotavlja, da je nek digitalni certifikat veljaven, ali pa ponudnik elektronske pošte, ki zagotavlja, da je bilo naše elektronsko sporočilo poslano, ali pa ponudniki spletnih družabnih omrežij, kot je npr. Facebook, ki nam zagotavljajo, da so naše objave vidne zgolj skupini naših prijateljev. Podobno so v vlogi tretje osebe tudi banke, ki nam zagotavljajo, da je bil naš denar nakazan v pravi meri in na pravi račun izbranega naslovnika. Vse to pa nakazuje, da so vsa naša dejanja v digitalnem svetu osnovana na zaupanju tretji osebi, predvsem, ko govorimo o dejanjih, za katera želimo, da se izvajajo varno. Kljub temu, da so te tretje osebe pogosto večje organizacije, katerim zaupamo, pa obstaja dejstvo, da so lahko tudi takšne organizacije manipulirane, ali morda tarča hekerskih napadov, kar ogrozi stopnjo zaupnosti, ki jo pričakujemo [2]. Tu pa nastopi

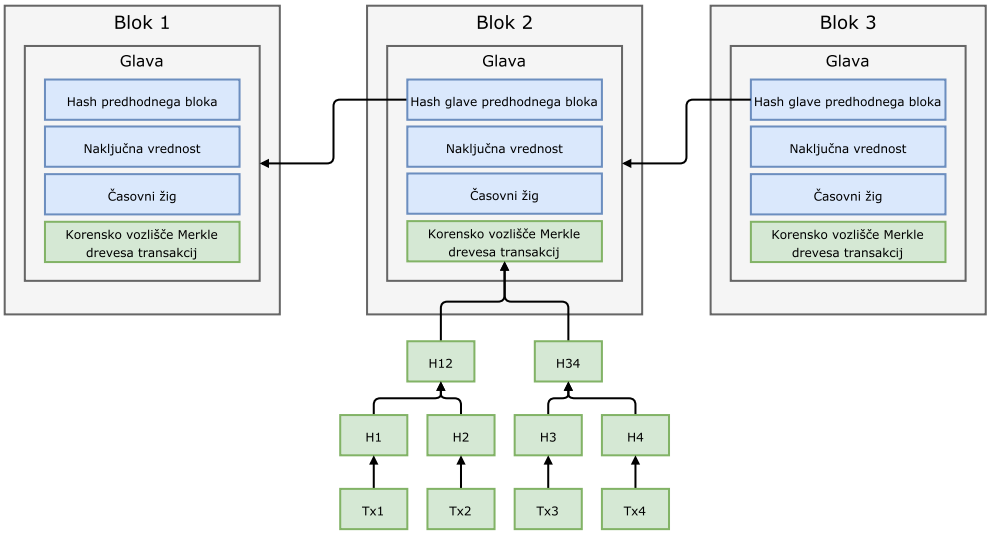
tehnologija verige blokov, ki nam z uporabo porazdeljenega soglasja vseh vpletenih vozlišč omogoča, da za vsako izvedeno transakcijo, ki se je kdaj zgodila, kadarkoli v prihodnosti enostavno preverimo njeno pravilnost. Če tej lastnosti porazdeljenega soglasja dodamo še anonimnost vpletenih vozlišč, pa dobimo dve glavni lastnosti verige blokov, zaradi katerih je veriga blokov postala tako priljubljena.

2.1.1 Kako deluje veriga blokov

Tehnično gledano je veriga blokov neke vrste porazdeljena podatkovna baza, prvotno namenjena knjiženju digitalnih transakcij. Veriga blokov je porazdeljena med vozlišči (ang. node), ki so računalniki, na katerih je nameščena programska oprema, s pomočjo katere vozlišča preverjajo pravilnost vseh transakcij, vključenih v verigi blokov in hkrati sodelujejo pri gradnji novega bloka, v katerega vključijo nove transakcije. Pomembno je, da vsako izmed vozlišč pri sebi hrani celotno verigo blokov.

Ker zaradi porazdeljenosti lahko pride do situacije, v kateri dve ali več vozlišč hkrati v omrežje pošljejo nov blok, se lahko zgodi, da del omrežja gradi eno vejo, medtem ko drugi del omrežja gradi drugo vejo verige blokov, odvisno od tega, kateri blok je po omrežju do njih pripotoval hitreje. Da bi rešili problem, v verigi blokov velja pravilo, da vozlišča vedno dodajajo nove bloke na konec najdaljše verige. Verjetnost, da bo omrežje naslednji blok na vsaki izmed vej ponovno zgradilo ob istem času, se manjša s številom dodanih blokov, in posledično bo vedno (običajno že z naslednjim blokom po vejitvi) prevladala ena izmed vej, ki bo postala glavna, s tem pa se bodo transakcije v drugi veji razveljavile.

Sodelovanje v verigi blokov je sicer javno in prostovoljno, pri čemer je glavni motiv sodelovanja v takšnem omrežju predvsem možnost dobitka nagrade za opravljeno delo. Vsa vozlišča, vključena v verigi blokov, namreč tekmujejo pri izdelavi novega veljavnega bloka, pri tem pa je potrebno rešiti računsko zahtevno uganko, ter s tem dokazati, da je bilo v izdelavo novega bloka vloženo delo, kar podrobneje opišemo v nadaljevanju.



Slika 2.1: Primer poenostavljene verige blokov.

Zgradba verige blokov

Dejanska zgradba verige blokov je sicer odvisna od same implementacije verige blokov, kar pomeni da verige blokov različnih ponudnikov med seboj niso identične. Kljub vsemu pa v principu vse verige blokov delujejo na enak način. V nadaljevanju predstavimo osnovno zgradbo verige blokov, pri čemer za zgled vzamemo verigo blokov Bitcoin, ki najbolj odraža verigo blokov, ki je bila predlagana v prvotnem članku [1].

Veriga je sestavljena iz tako imenovanih blokov, znotraj vsakega pa je zabeleženih več transakcij. Za vse transakcije znotraj bloka se predpostavi, da so se zgodile ob istem času. Bloki se v verigo vedno le dodajajo in verige za nazaj ni mogoče spreminjati. Vsak blok v glavi vsebuje časovni žig, zgoščeno (hash) vrednost glave predhodnega bloka, naključno vrednost, ki jo pri rudarjenju vozlišča z ugibanjem izberejo tako, da se zgoščena vrednost tako sestavljenega bloka začne z vnaprej določenim številom ničel, ter korensko vozlišče tako imenovanega *Merkle* drevesa transakcij. Primer poenostavljene verige blokov je prikazan na sliki 2.1.

Transakcije, ki so vključene v blok, niso shranjene v glavi, pač pa se v glavi

nahaja zgolj korensko vozlišče Merkle drevesa transakcij. Merkle drevo [3][4] je zgrajeno tako, da najprej izračunamo zgoščeno vrednost za vsako izmed transakcij, nato paroma združujemo zgoščene vrednosti transakcij, tako da izračunamo novo zgoščeno vrednost združenih transakcij, in to ponavljamo vse dokler ne pridemo do enega korenskega vozlišča (glej sliko 2.1), ki se shrani v glavo bloka. Namen takšnega shranjevanja transakcij je v prvotni zasnovi [1] opisan kot način optimizacije prostora v primeru vozlišč, ki hranijo zgolj glave vseh blokov. Z uporabo Merkle drevesa pa je možno tudi v logaritemskem času dokazati, da je določena transakcija vsebovana v bloku.

Ključna lastnost, zaradi katere je veriga varna, je vsebovanost zgoščene vrednosti prejšnjega bloka. Tako je v primeru, da napadalec želi spremeniti vrednost katere izmed transakcij za nazaj, potrebno na novo izračunati vse bloke od tistega trenutka dalje.

Glavna in revolucionarna prednost verige blokov pa ni v sami strukturi verige, pač pa v dejstvu, da je le ta varovana s strani omrežja, kjer je mehanizem za zagotavljanje varnosti zasnovan na principu nagrajevanja - bodisi na principu mehanizma Proof of Work ali Proof of Stake.

Mehanizem Proof of Work (PoW)

Zagotavljanje varne verige blokov je bilo prvotno predlagano na osnovi mehanizma Proof of Work. Pri tem se predpostavi, da je za gradnjo vsakega bloka potrebno vložiti določeno, dovolj veliko količino dela, kar kasneje onemogoča enostavno spreminjanje blokov, že vsebovanih v verigi, saj bi bilo za takšen poseg potrebno ponovno vložiti ogromne količine dela. V omrežju Bitcoin vozlišča pri gradnji novega bloka skušajo rešiti kriptografsko uganko (glej enačbo 2.1), pri kateri k samemu bloku naključno dodajajo poljubno vrednost N, tako da je zgoščena vsebine bloka B in poljubne vrednosti N po dvakratnem zgoščevanju z zgoščevalno funkcijo $SHA256$, manjša od vnaprej določene vrednosti T, imenovane težavnost bloka. Obraten proces preverjanja (ali izbrana vrednost N ustreza pogoju) je enostaven, samo iskanje takšne vrednosti pa je računsko zelo zahtevno in se po potrebi prilagaja s

spreminjanjem vrednosti T, tako da se doseže konstantno dodajanje novih blokov na približno 10 minut. Procesu iskanja vrednosti N, ki ustreza pogoju, pravimo tudi rudarjenje (ang. mining), njegov rezultat dokazuje, da je bila v gradnjo bloka vložena velika količina dela, vozlišča v omrežju pa pri reševanju računsko zahtevne operacije sodelujejo predvsem zaradi možnosti dobitka nagrade za zgrajen blok, ki se na približno štiri leta razpolovi in v času pisanja znaša $12.5BTC$.

$$SHA256(SHA256(B||N)) < T \tag{2.1}$$

Mehanizem Proof of Stake (PoS)

Kljub temu, da se mehanizem "proof of work" v praksi dobro obnese, pa obstajajo pri uporabi le-tega resni pomisleki. Za gradnjo novih blokov se namreč porabljajo ogromne količine električne energije. Po podatkih iz leta 2014[5] je že samo pri gradnji blokov v omrežju Bitcoin poraba električne energije za gradnjo blokov presegala porabo električne energije celotne Irske. Od takrat se je računska moč močno povečala, prav tako pa se je povečalo število verig blokov, ki uporabljajo mehanizem proof of work. Z namenom reševanja te problematike se že nekaj let v raznih verigah blokov skuša vpeljati nov mehanizem, imenovan proof of stake [6][7][8], ki ne temelji na dokazu vloženega dela, pač pa se naslednji blok izbere med vozlišči naključno in sorazmerno glede na količino valute, ki jo le-ta poseduje. Varnost takšnega mehanizma temelji na dejstvu, da več kot ima vozlišče vloženo v samo verigo blokov, manj mu je v interesu uničiti takšen sistem.

2.1.2 Področja uporabe verige blokov

Uporaba verige blokov ni omejena zgolj na finančni svet, temveč je to zgolj eno izmed področij, kjer je uporaba takšne tehnologije smiselna. V nadaljevanju so našteta bolj razširjena področja uporabe verige blokov, skupaj s primeri že razvitih aplikacij [2]:

Vrednostni papirji Ena izmed najbolj razširjenih uporab verige blokov je gotovo trgovanje z elektronskimi valutami, ki pa ga lahko razširimo tudi na področje trgovanja delnic in ostalih vrednostnih papirjev. Primeri storitev, ki tehnologijo verige blokov uspešno uporabljajo v ta namen so npr. Chain.com in Funderbeam, vključevanja verige blokov v obstoječe sisteme pa se poslužujejo tudi večje organizacije, kot so NASDAQ in ASX [9].

Zavarovalništvo Na področju zavarovalništva lahko veriga blokov igra pomembno vlogo pri preverjanju zahtev za izplačilo odškodnine - npr. lažni ali podvojeni zahtevki bi bili avtomatično zavrnjeni s strani omrežja v primeru, da je bila za določen dogodek odškodnina že povrnjena. Poleg tega bi lahko z uporabo pametnih pogodb samodejno izvrševali povračilo odškodnine v primerih, ko bi prišlo do dogodka in bi le tega omrežje tudi potrdilo.

Notarske storitve Veriga blokov lahko olajša delo na področju notarskih storitev, saj nam omogoča, da za dokumente izdelamo dokaz o avtorstvu (kdo je lastnik dokumenta), dokaz o obstoju dokumenta v točno določenem času, ter dokaz o nespremenjenosti dokumenta. Primeri takšnih storitev so Block Notary, Stampery, Crypto Public Notary in Viacoin, ki omogočajo izvajanje notarskih storitev brez v ta namen pooblaščenih oseb.

Glasbena industrija Področje avtorskih pravic, ki je zaradi popularnosti interneta in pretočnih storitev postalo občutno bolj oteženo, bi bilo prav tako možno reševati s pomočjo porazdeljene verige blokov. Še več, z uporabo pametnih pogodb, bi bilo možno avtorjem, glasbenikom in produkcijskim hišam avtomatično plačevati njihove avtorske pravice, kot tudi določati razmerja med njimi.

Dokaz o obstoju Obstoječi mehanizmi dokazovanja obstoja določenega dokumenta v času, ali pripadnost dokumenta določeni osebi v danem času

temeljijo na centralni avtoriteti, ki potrjuje obstoj določenega dokumenta v danem času. Problem takšnih sistemov pa ni zgolj v zaupanju v pravilnost podatkov, posredovanih s strani tretje osebe, temveč tudi v tem, da takšni sistemi predpostavljajo, da tretja oseba hrani bodisi originale, bodisi kopije teh dokumentov, kar lahko predstavlja varnostni problem. Z verigo blokov je takšen sistem enostaven, hkrati pa ne temelji na hranjenju vsebine dokumentov, pač pa zgolj izvlečkov. Primer enostavne storitve, ki omogoča takšen mehanizem je storitev Proof of Existence [10].

Porazdeljena shramba Shramba v oblaku je postala v zadnjih letih nepogrešljiva stvar večine uporabnikov interneta, naj bo to uporaba storitev Google, Dropbox, One Drive in ostalih. Podobno kot pri ostalih storitvah pa tudi tu ostaja problem zaupanja tretji osebi. V ta namen Storj ponuja platformo za decentralizirano shrambo večjih količin podatkov [11], kjer pa se vsi podatki ne shranjujejo v eni ogromni verigi blokov, pač pa se veriga blokov uporablja zgolj za shranjevanje meta podatkov datotek, datoteke same pa so ločeno shranjene na prostoru, ki ga sodelujoča vozlišča nudijo v zameno za plačilo v kripto valuti Bitcoin.

Porazdeljen internet stvari IoT je ena izmed novejših tehnologij, ki v zadnjih letih postaja vse bolj priljubljena. Večina takšnih sistemov deluje na principu centraliziranega sistema, kjer en vmesnik skrbi za večjo količino nanj povezanih naprav - posledično je varnost naprav in varna komunikacija med napravami odvisna od tega vmesnika. Filament je storitev, ki omogoča da postavimo sistem IoT, kjer naprave komunicirajo neposredno ena z drugo, pri tem pa veriga blokov služi kot zaupanja vredna shramba identifikatorjev naprav [12].

Sistemi proti ponaredkom Boj proti ponaredkom ostaja eden izmed večjih izzivov današnjih časov. Storitev BlockVerify ponuja rešitev, ki temelji na verigi blokov. Njihov cilj je reševanje problema ponarejanja s pomočjo decentraliziranega sistema, katerih del so tako proizvajalci,

prodajalci in posredniki, kot tudi končni uporabniki, predvsem na področju farmacije, diamantov, elektronskih naprav ter luksuznih predmetov [13].

Storitve na področju interneta Pojavljajo se aplikacije, ki sicer dobro vpeljane storitve implementirajo še z uporabo verige blokov. Primer takšne storitve je Namecoin, ki je porazdeljena imenska storitev DNS [14]. Obstoječi DNS strežniki so namreč nadzirani s strani držav in večjih korporacij, z uporabo verige blokov pa se izgubi potreba po takšnih organizacijah.

2.2 Pametne pogodbe

Princip pametnih pogodb je sprva predstavil Nick Szabo že leta 1997 [15] z idejo o pogodbah, ki temeljijo na programski kodi in se lahko samodejno izvajajo, vendar pa takratna tehnologija ni omogočala izvajanja takšnih pogodb brez posredovanja centralne avtoritete. Ideja o pametnih pogodbah se je tako ponovno prebudila s prihodom verige blokov. Pametne pogodbe namreč potrebujejo nepristranskega posrednika, za kar se veriga blokov izkaže kot primeren kandidat prav zaradi lastnosti distribuiranega omrežja ter z njim povezane lastnosti odločanja. Pametne pogodbe namreč lahko preverjamo in izvajamo na enak način, kot poteka preverjanje digitalnih transakcij v omrežju Bitcoin, kar pomeni, da bo vsako izvedeno akcijo znotraj pogodbe izvedlo in preverilo vsako vozišče v omrežju, s tem pa dosežemo pošteno izvajanje pogodb brez potrebe po tretji osebi.

Projekt Ethereum je prvi ponudnik javne verige blokov, ki omogoča izvajanje pametnih pogodb, sama veriga blokov pa je bila zasnovana prav za podporo pametnim pogodbam. V ta namen so namreč uvedli poseben virtualni stroj (EVM), ki je zmožen izvajanja programske kode na verigi blokov in deluje kot neke vrste decentraliziran računalnik. Poleg tega so omogočili pisanje pametnih pogodb v uporabniku prijaznem visoko nivojskem programskem jeziku Solidity.

Pametno pogodbo se ob kreiranju shrani v verigo blokov, tako kot katerokoli drugo transakcijo. Takšna pogodba je vidna vsem, kar pomeni, da lahko vsak uporabnik preveri, kaj se bo izvedlo ob klicu pogodbe. Poleg tega je takšno pogodbo, kot vse druge transakcije v verigi blokov, nemogoče spremeniti, niti ni možno spreminjati poteka izvajanja. Uporabnik je tako lahko prepričan, da se bo takšna pogodba vedno izvedla natanko na način, kot je zapisana. Izvajanje ukazov pa se znotraj pametne pogodbe izvaja tako, da se izvede zaporedje transakcij, ki spreminjajo vrednost pametne pogodbe. Preverjanje pravilnosti izvedenih operacij potem poteka kot preverjanje vseh ostalih transakcij v verigi blokov - vsa vključena vozlišča v verigi blokov potrjujejo transakcije, tako običajne transakcije, kot tudi transakcije, ki spreminjajo stanje pametne pogodbe [16].

Princip delovanja pametnih pogodb je najlažje prikazati na primeru. Vzemimo šolski primer pametne pogodbe, ki predstavlja platformo za množično kampanjo zbiranja sredstev (ang. crowdfunding):

```
contract Kampanja{

    uint public cilj;
    address public avtor;

    Uporabnik[] public seznamPrispevkov;

    struct Uporabnik{
        address naslovZaVracilo;
        uint znesek;
    }

    function Kampanja(uint _cilj){
        avtor = msg.sender;
        cilj = _cilj;
    }
```

```
    function prispevaj(){
        seznamPrispevkov.push(Uporabnik(
            msg.sender,msg.value));
        preveriDosezenCilj();
    }

    function preveriDosezenCilj(){
        if(this.balance >= cilj){
            suicide(avtor);
        }
    }

    function preklici(){
        if(msg.sender != avtor) return;

        for(uint i=0;i<seznamPrispevkov.length;i++){
            seznamPrispevkov[i].naslovZaVracilo.send(
                seznamPrispevkov[i].znesek);
        }
    }
}
```

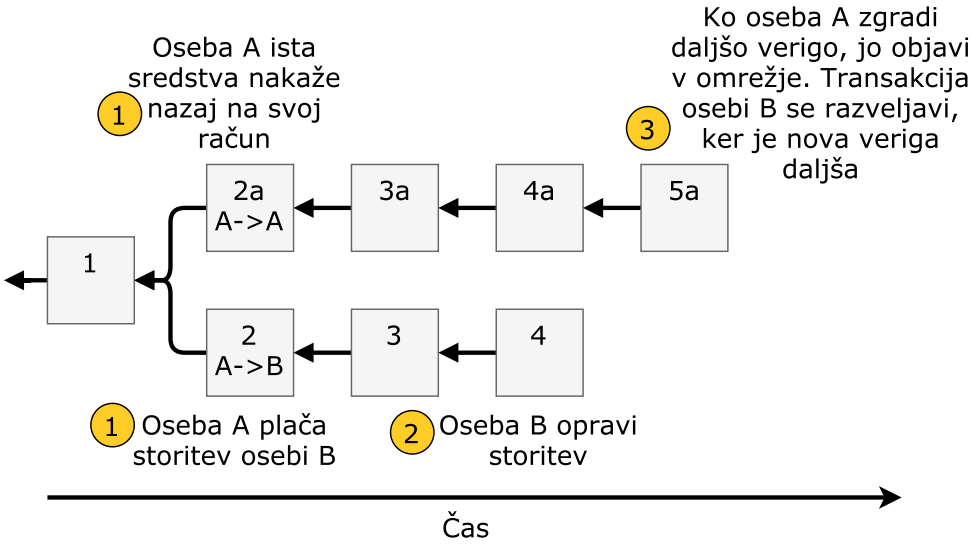
Pametna pogodba vsebuje spremenljivko *cilj*, ki določa višino zneska, pri katerem se kampanjo smatra za uspešno in se sredstva nakažejo avtorju. Naslov avtorja hranimo v spremenljivki *avtor*. Poleg teh dveh spremenljivk hranimo še seznam vseh uporabnikov, ki so prispevali h kampanji. Seznam darovalcev hranimo v spremenljivki *seznamPrispevkov*, ki je sestavljena iz elementov tipa *Uporabnik*, ki hkrati prikazuje kako lahko ustvarimo lastne podatkovne strukture. Za vsakega uporabnika hranimo znesek njegovega prispevka, in pa podatek o njegovem naslovu, ki ga bomo po-

trebovali v primeru preklica kampanje in vračila sredstev. Ob kreiranju pametne pogodbe se pokliče metoda $Kampanja(uint_cilj)$, v kateri se nastavita cilj ter avtor pogodbe. Ob klicu metode *prispevaj*() se na seznam prispevkov zabeleži uporabnika in njegov prispevek. Hkrati se pokliče metoda *preveriDosezenCilj*(), ki preveri, ali je cilj že dosežen, in v primeru da je, s klicem vgrajene funkcije *suicide*() vsa zbrana sredstva prenese na avtorjev naslov. V primeru, da avtor želi v določenem trenutku preklicati kampanjo, to naredi s klicem metode *preklici*(), ki vsem uporabnikom, ki so prispevali sredstva, vrne njihove prispevke.

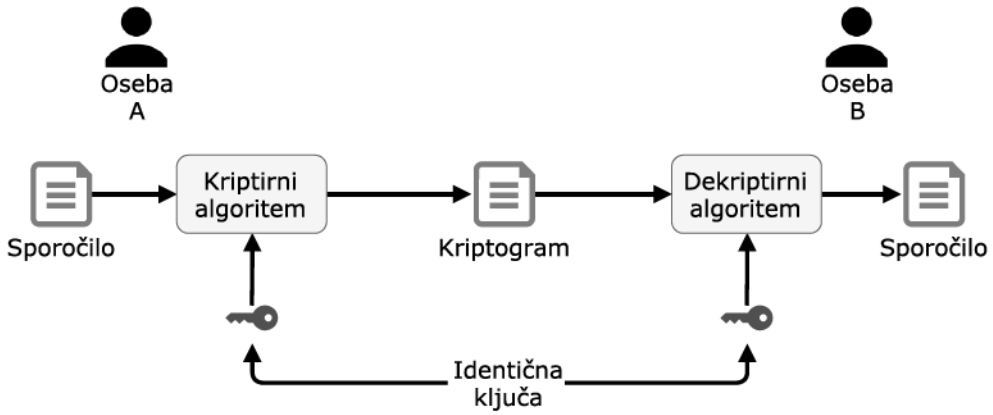
2.3 Varnostne pomanjkljivosti verige blokov

Kljub vsem prednostim, ki jih prinaša uporaba verige blokov, pa le-ta v sami zasnovi vsebuje tudi resne varnostne pomanjkljivosti. Ker je večina verig blokov primarno namenjenih digitalnim valutam, so razlogi za napade predvsem želja po pridobitvi denarne valute. V ta namen se na verigi blokov največkrat izvajajo napadi, kjer je cilj dvakratna poraba istih sredstev (ang. double spending). Najpogosteje je tu govora o večinskem napadu (ang. majority attack) ali napadu 51%, kjer predpostavimo, da ima napadalec vsaj polovico celotne računske moči omrežja. Ker je veriga blokov zasnovana tako, da omrežje vozlišč nove bloke vedno dodaja v najdaljšo verigo, lahko napadalec po opravljenem plačilu in izvršeni storitvi ustvari novo vejo v verigi, kjer ista sredstva porabi še enkrat, takšno verigo pa dopolnjuje z novimi bloki tako, da le ta preseže dolžino prvotne veje. Nato takšno verigo pošlje v omrežje, vozlišča pa jo, ker je daljša, vzamejo za primarno vejo, ter razveljavijo transakcije, ki so se zgodile na prvotni veji. Primer takšnega napada prikazuje slika 2.2.

Ker običajno pri plačevanju ponudnik storitve za potrditev plačila čaka, da je potrjenih vsaj nekaj naslednjih blokov, napadalec za takšen napad potrebuje precej več kot 51% celotne računske moči, saj verjetnost gradnje zaporednih m blokov hitreje kot preostanek omrežja hitro pada [17].



Slika 2.2: Primer napada na verigi blokov, kjer napadalec A ista sredstva porabi dvakrat - prvič jih nakaže ponudniku storitve B, drugič pa nazaj na svoj račun. Napadalec nato novo ustvarjeno vejo dopolnjuje z bloki hitreje od ostalega omrežja, kar povzroči razveljavitev transakcije od osebe A k osebi B.

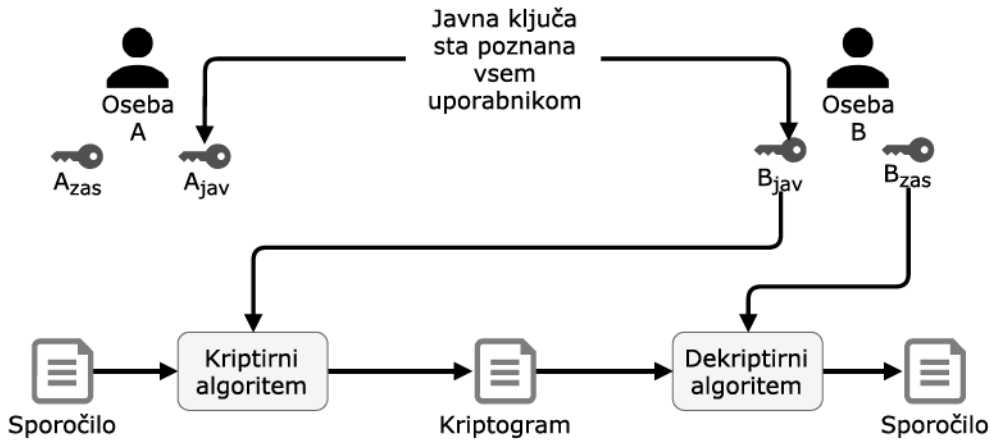


Slika 2.3: Kriptiranje in dekriptiranje sporočila z uporabo simetrične kriptografije.

2.4 Simetrična kriptografija

Pod pojem simetrične kriptografije uvrščamo kriptografske tehnike kriptiranja sporočila, pri katerem sta ključa tako za kriptiranje podatkov kot tudi dekriptiranje identična. Da je takšen način kriptiranja varen, mora biti ključ skrit in poznan le končnima uporabnikoma. Pri kriptiranju sporočila uporabnik z uporabo vnaprej dogovorjenega kriptirnega algoritma (ki je lahko javno znan) in ključa, ki je poznan le pošiljatelju in sprejemniku, kriptira sporočilo, prejemnik pa prejeto sporočilo z obratnim kriptirnim algoritmom in istim ključem spremeni nazaj v čistopis. Primer uporabe kriptiranja z uporabo simetrične kriptografije je prikazan na sliki 2.3.

Pri uporabi simetrične kriptografije je potrebno zagotoviti varno izmenjavo ključev, bodisi preko osebne izmenjave med uporabnikoma, bodisi preko drugega varnega kanala. Glavna prednost uporabe simetričnih kriptirnih algoritmov pa je predvsem hitrost kriptiranja, ter dejstvo, da je kriptirano sporočilo enake dolžine kot original.



Slika 2.4: Kriptiranje in dekriptiranje sporočila z uporabo kriptografije javnih ključev.

2.5 Kriptografija javnih ključev

Poleg simetrične kriptografije poznamo tudi asimetrično kriptografijo, pri kateri ključ, namenjen kriptiranju sporočila, ni enak ključu, namenjenem dekriptiranju sporočila. Najbolj znana oblika asimetrične kriptografije je kriptografija z javnimi ključi. Vsak uporabnik U ima dva ključa: javnega U_{jav} in zasebnega U_{zas}. Zasebni ključ je poznan le lastniku ključa, medtem ko je javni ključ poznan vsem ostalim uporabnikom. Če želi oseba A kriptirati sporočilo tako, da ga bo lahko prebrala le oseba B, to stori tako, da sporočilo kriptira z javnim ključem osebe B (B_{jav}) in tako kriptirano sporočilo pošlje osebi B. Oseba B, ki edina pozna svoj zasebni ključ, lahko sporočilo odkriptira z uporabo svojega zasebnega ključa B_{zas}. Postopek kriptiranja na takšen način prikazuje slika 2.4.

Glavni problem kriptografije z javnimi ključi predstavlja verodostojnost ključev - dokaz o tem, da javni ključ pripada osebi, za katero je ustvarjen, in da, le-ta ni bil spremenjen s strani tretje osebe. V ta namen uporabljamo mehanizem infrastrukture javnih ključev (ang. public key infrastructure), ki s pomočjo centralne avtoritete (CA) omogoča varno distribucijo certifikatov, namenjenih preverjanju javnih ključev.

2.6 Zgoščevalne funkcije

Zgoščevalne funkcije ali kriptografske zgoščevalne funkcije so matematični algoritmi, ki za vhodni podatek poljubne dolžine vrnejo rezultat fiksne dolžine imenovan izvleček ali zgoščena vrednost. Takšna preslikava je enosmerna, kar pomeni, da za dan izvleček ni mogoče ugotoviti kakšen je bil vhodni podatek. Kakovost oziroma varnost zgoščevalnih funkcij je določena z naslednjimi lastnostmi:

Odpornost na prasliko (ang. pre-image resistance) Med takšne zgoščevalne funkcije sodijo tiste, za katere je pri poznanem izvlečku h težko ali nemogoče najti sporočilo m, ki ustreza $h = H(m)$, pri čemer H predpostavlja zgoščevalno funkcijo.

Odpornost na drugo prasliko (ang. second pre-image resistence) Med te uvrščamo tiste zgoščevalne funkcije, za katere velja, da je za dano vhodno sporočilo m_1 težko ali nemogoče najti sporočilo m_2, ki bi ustrezalo pogoju $H(m_1) = H(m_2)$

Odpornost na trke Med takšne zgoščevalne funkcije sodijo tiste, za katere velja, da je težko ali nemogoče najti poljubni dve sporočili m_1 in m_2, za katere bi veljalo $H(m_1) = H(m_2)$.

V povezavi z verigo blokov bomo v nalogi največkrat omenjali zgoščevalno funkcijo $SHA-256$, saj se le-ta uporablja pri reševanju matematične uganke, na kateri je osnovan sistem nagrajevanja.

2.7 Integriteta podatkov

Integriteta podatkov je pojem, ki opisuje celovitost ali nespremenljivost podatkov. O integriteti podatkov govorimo, ko želimo pokazati, da je podatek ostal nespremenjen in celovit, med prenosom ali med hranjenjem podatka.

Za zagotavljanje integritete podatkov se v računalništvu najpogosteje uporabljajo zgoščevalne funkcije, ki imajo lastnost, da za dan podatek poljubne dolžine vedno vračajo enak rezultat fiksne dolžine. Preverjanje celovitosti ali integritete podatka lahko torej preverimo tako, da za dan podatek izračunano zgoščeno vrednost pred in po operaciji (npr. prenos po omrežju) in preverimo, ali se zgoščeni vrednosti ujemata.

2.8 Sorodne raziskave

Pregled raziskav s področja verige blokov je sistematično prikazan v nedavno objavljenem članku [18], kjer avtorji pregledajo vse relevantne članke tega področja ter podajo mnenje o tem, na katera področja naj bi se nadaljnje raziskave verige blokov posebej osredotočale. Med drugim menijo, da je razvoj novih aplikacij, ki uporabljajo verigo blokov, ključnega pomena za razvoj ne zgolj kripto valut pač pa tudi verige blokov kot tehnologije. Iz članka je razvidno, da se sicer velik del raziskav s tega področja posveča predvsem odkrivanju varnostnih in tehnoloških problemov elektronske valute Bitcoin, ter nadgrajevanju le-te.

Za naše delo so predvsem pomembni članki, ki se dotikajo samih zmožnosti in tehnoloških omejitev verige blokov [19][20], še posebej članek [21], ki se sicer osredotoča na tehnološke raziskave kripto valute Bitcoin, vendar je večji del raziskav namenjen verigi blokov. Rezultati raziskav kažejo, da trenutna veriga blokov Bitcoin še ni pripravljena za ogromne količine transakcij, vendar pa so mnenja, da bo, glede na razvoj v tej smeri, veriga blokov Bitcoin omogočala večji pretok transakcij še preden se dejansko pojavi potreba po tem.

Med potencialnimi področji uporabe verige blokov se sicer pogosto navaja varnost osebnih podatkov [22][23][24], vendar pa nihče ni raziskoval na področju zaupnosti in integritete podatkov v obstoječih sistemih. V članku [25] sicer avtorji predlagajo sistem za dodeljevanje pravic vpogleda v osebne podatke, ki temelji na verigi blokov, vendar pa se pri tem osredotočajo na

konkreten primer nameščanja mobilnih aplikacij, kjer ponudniku aplikacije omogočimo vpogled v naše osebne podatke. S podobnim problemom upravljanja z osebnimi podatki se srečujejo tudi v delu [26], kjer predlagajo zasnovo sistema za upravljanje z medicinskimi podatki, ki ne le da uporablja verigo blokov, pač pa v ta namen pripravijo tudi zasnovo lastne verige blokov, vključno s sistemom potrjevanja. Pri tem sicer velja poudariti, da je za varnost verige blokov ključnega pomena dovolj veliko omrežje [1][27], kar bo potrebno upoštevati v primeru, da se odločimo za lastno implementacijo verige blokov.

Pri pregledu obstoječih aplikacij, ki temeljijo na verigi blokov, smo zasledili seznam ponudnika javno dostopne verige blokov Ethereum, na katerem so navedene vse aplikacije, ki uporabljajo njihovo verigo blokov [28]. Med razvitimi projekti so npr. igre na srečo, porazdeljene shrambe podatkov, sistemi za upravljanje z identitetami, itd., vendar pa projekta, ki bi reševal problem zaupnosti in integritete podatkov v obstoječih sistemih, nismo zasledili.

Poglavje 3

Načrt sistema

Cilj te naloge je torej narediti sistem, ki bo zagotavljal zaupnost in integriteto podatkov v že obstoječih sistemih ali storitvah. V tem poglavju tako razvijemo načrt in predlog vpeljave sistema za zagotavljanje zaupnosti in integritete podatkov. V prvem podpoglavju najprej predstavimo metodologijo dela, nato pa razvijemo dva ločena načrta - v prvem se posvetimo zagotavljanju integritete podatkov, v drugem pa predstavimo načrt načina uporabe verige blokov še za zagotavljanje zaupnosti.

3.1 Metodologija

V prvem koraku izdelave magistrske naloge smo pregledali širše področje verig blokov, ter izdelali idejo za nov način uporabe verige blokov, kjer bi obstoječe storitve nadgradili z vpeljavo verige blokov ter jim s tem omogočili višjo stopnjo zaupnosti ali integritete podatkov. V drugem koraku bomo izvedli študijo izvedljivosti, ter na primeru spletne aplikacije družabnega omrežja razvili predlagan sistem, sam razvoj pa razdelili v dve fazi. V prvi bomo uporabili verigo blokov za hranjenje izvlečkov vsebine, ki jo uporabniki družabnega omrežja objavljajo. S tem bomo zagotovili integriteto podatkov, saj bo vsak uporabnik lahko preveril, ali zapis, ki ga prikazuje ponudnik storitve, ustreza tistemu, ki je bil objavljen. V drugi fazi bomo rešitev razširili

z mehanizmom, ki bo zagotavljal zaupnost. Veriga blokov bo v tem primeru uporabljena kot sistem dodeljevanja pravic do vpogleda v uporabnikovo vsebino, ki bo temeljila izključno na verigi blokov. S tem bo uporabnik vedno edini, ki lahko določi pravice za ogled vsebine. V tretjem koraku bomo za razvit sitem identificirali omejitve, ki jih prinaša uporaba verige blokov, ter izmed javno dostopnih verig blokov izbrali najprimernejšo, oziroma pokazali, ali bi bila uporaba in razvoj lastne verige blokov za razvito rešitev bolj primerna.

3.2 Načrt sistema za zagotavljanje integritete podatkov

Pri načrtu sistema za zagotavljanje integritete podatkov v obstoječih sistemih za primer obstoječega sistema vzamemo spletno aplikacijo družabnega omrežja. Izbira takšne aplikacije morda ni najboljši primer za dejansko implementacijo, kot to kasneje tudi prikažemo v poglavju 6, služi pa kot dober primer za razumevanje predlaganega sistema. Namen je tako izdelati rešitev, ki bo omogočala višjo stopnjo integritete podatkov vsebine, ki jo uporabniki na takšnem družabnem omrežju objavljajo. V poglavjih, ki sledijo, načrt razvijemo po korakih, od prvotne enostavne idejne zasnove, do končnega, izpopolnjenega načrta.

3.2.1 Idejna zasnova

Prvotna ideja je uporabiti verigo blokov za hranjenje celotne vsebine, ki jo uporabniki objavljajo. Na ta način dosežemo, da je vsaka sprememba vsebine zabeležena v verigi blokov, in ker je veriga blokov vidna vsem, lahko vsak uporabnik sistema preveri, kdaj je bila vsebina dodana v sistem, in ali se je v tem času morda spreminjala. Takšen sistem sicer omogoča popoln pregled nad celotno zgodovino vsebine, vendar pa hitro opazimo največjo težavo takšnega pristopa - vsa vsebina in njene spremembe so na tak način dosto-

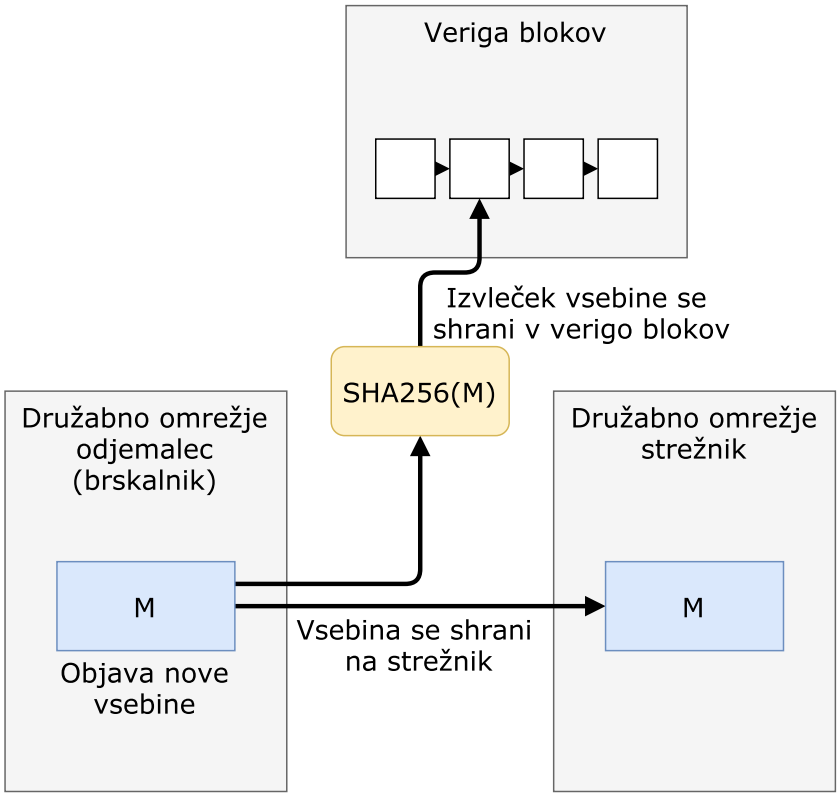
pne vsakomur, ki ima dostop do verige blokov. Ni nujno, da to vzamemo kot slabo lastnost sistema - v določenih primerih bi bila takšna zasnova sistema povsem smiselna in zadostna, vendar pa se tu pojavi tudi drug problem - kot smo prikazali v poglavju 2, veriga blokov ni namenjena shranjevanju ogromnih količin podatkov, saj mora vsako vozlišče v omrežju pri sebi hraniti celotno verigo blokov.

Prvotno idejo torej spremenimo tako, da znotraj verige blokov vedno hranimo le izvleček vsebine. Na ta način lahko za določeno vsebino še vedno določimo, ali se njen izvleček nahaja v verigi blokov, in ali se je morda vsebina spremenila, po drugi strani pa same vsebine ni mogoče določiti - seveda ob predpostavki, da izberemo primerno in dovolj močno zgoščevalno funkcijo. Preprost načrt takšnega sistema je prikazan na sliki 3.1.

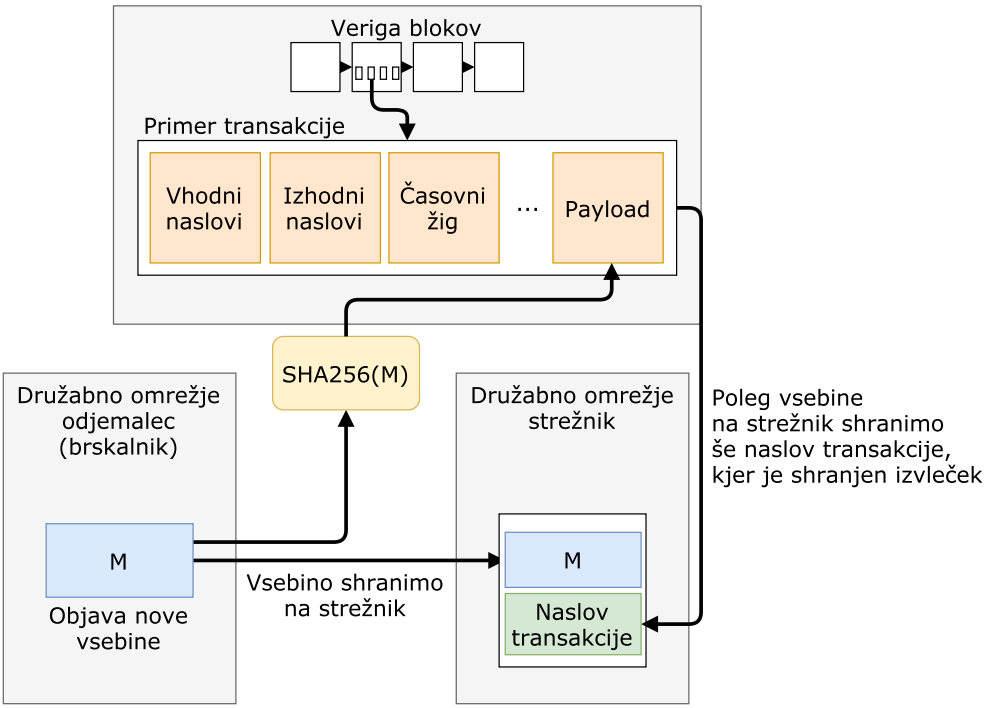
Veriga blokov nam bo v tem primeru služila kot podatkovna shramba izvlečkov vsebine. Ker pa je v osnovi veriga blokov namenjena shranjevanju transakcij, moramo sistem prilagoditi tako, da bodisi izvlečke na nek način pretvorimo v transakcije, ali pa uporabimo pametne pogodbe, ki takšno shrambo v verigi blokov že omogočajo. V nadaljevanju predlagamo načrt dveh rešitev. Prvega z uporabo enostavne verige blokov, drugega pa z uporabo pametnih pogodb, ter s tem omogočimo razvoj sistema za poljubno verigo blokov.

3.2.2 Načrt sistema z uporabo verige blokov

Pri uporabi enostavne verige blokov je edini način shranjevanja podatkov ta, da jih na nek način pretvorimo v transakcijo. Večina verig blokov je namreč zasnovanih tako, da v strukturi same transakcije poleg podatkov, ki so obvezni za veljavno transakcijo (vhodni in izhodni naslovi, časovni žig, itd.) vsebujejo tudi dodatna polja, ki niso obvezna, ter so namenjena bodisi opcijskim podatkom (npr. sporočilo ob poslani transakciji), bodisi kasnejšim razširitvam protokola. Veriga blokov Ethereum na primer v svoji zasnovi vsaki transakciji dodeljuje polje imenovano *payload*, kamor lahko uporabnik zapiše poljubno vsebino ob ustvarjanju transakcije. Veriga blokov Bitcoin



Slika 3.1: Načrt idejne zasnove uporabe verige blokov za zagotavljanje integritete podatkov.



Slika 3.2: Načrt uporabe verige blokov za zagotavljanje integritete podatkov, ki temelji na osnovi shranjevanja izvlečkov vsebine znotraj transakcij.

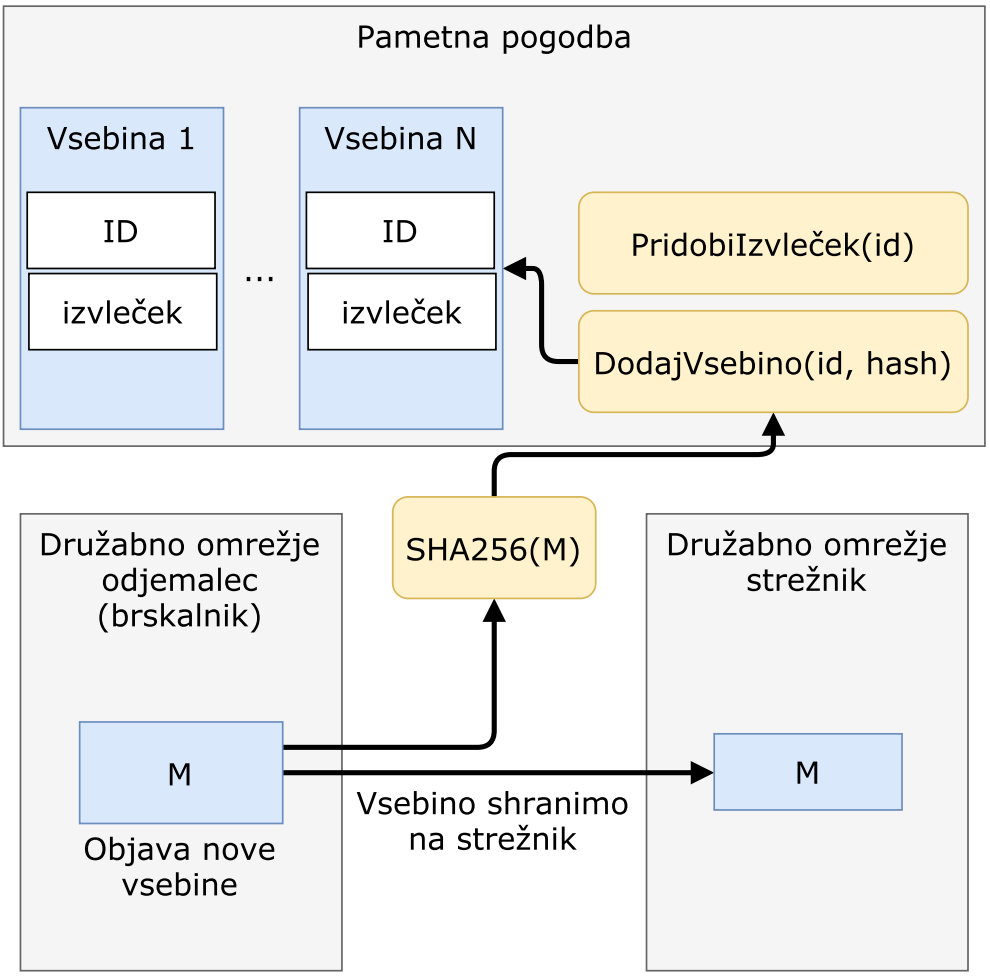
vsebuje polje *coinbase*, ki ga prav tako lahko izkoristimo za vnos sicer krajše vsebine, vendar v primeru shranjevanja izvlečkov to ne predstavlja ovire. V nadaljevanju predpostavimo, da verige blokov vsebujejo takšno polje, pri samem načrtu pa za zgled vzamemo polje *payload*, kakršnega najdemo v verigi blokov Ethereum.

Načrt, prikazan na sliki 3.2, prikazuje načina shranjevanja izvlečkov tako, da se le-ti shranijo v polju *paylod* pri vsaki transakciji. Tako shranjen izvleček pa uporabniku ne koristi, če le-ta ni obveščen o naslovu, kjer se transakcija nahaja. Možno bi bilo sicer preiskati celotno verigo blokov ter preverjati ali kateri izmed zapisov ustreza uporabnikovemu izvlečku, vendar pa to v praksi zaradi velikosti verige blokov ni uporabno. Načrt torej razširimo tako, da se podatek o naslovu, kjer se izvleček nahaja, prikaže uporabniku, hkrati pa ta podatek pripnemo še vsebini, preden se le-ta zapiše na strežnik storitve.

Pomembno je, da uporabnik komunicira neposredno z verigo blokov, saj bi v primeru, da shranjevanje izvlečkov v verigo blokov prepustimo ponudniku storitve, varnost takšnega sistema temeljila na zaupanju ponudniku, čemur se želimo izogniti. Na sliki je tako nazorno prikazano, da komunikacija z verigo blokov poteka neposredno med verigo blokov in končnim uporabnikom - preko klicev, ki se izvajajo na strani odjemalca. Podatek o naslovu izvlečka na strani storitve shranimo z namenom, da se le-ta lahko prikaže vsem ostalim uporabnikom - ponudnik storitve bi ta podatek skupaj s samo vsebino sicer lahko spremenil, vendar pa bi v tem primeru avtor vsebine vseeno imel dostop do dejanskega naslova izvlečka, ter bi s tem lahko pokazal, da ponudnik storitve ni pravičen. Da bi povsem preprečili, da ponudnik storitve spreminja vsebino ni naslov izvlečka, bi lahko sicer sam izvleček zaščitili z digitalnim podpisom, vendar je namen tega dela prikazati uporabo verige blokov, morebitne izboljšave sistema pa pregledamo v poglavju 6.

3.2.3 Načrt sistema z uporabo pametnih pogodb

Drug pristop k problemu je uporaba pametnih pogodb. Takšna rešitev je sicer uporabna zgolj pri ponudnikih verig blokov, ki podpirajo pametne pogodbe, vendar pa prinaša prednosti. Načrt je podoben prejšnjemu, le da bomo v tem primeru izkoristili lastnosti pametnih pogodb, ter omogočili pregled zgodovine sprememb za vsebino, v primeru, da jo uporabnik spreminja. Zgodovina sprememb je sicer v verigi blokov shranjena tudi v prejšnjem primeru, vendar pa se ob spremembi vsebine izgubijo naslovi prejšnjih stanj in nam je na voljo zgolj naslov izvlečka zadnje spremembe. Načrt rešitve, ki uporablja pametne pogodbe, prikazuje slika 3.3. Tudi v tem primeru je pomembno, da uporabnik komunicira neposredno s pametno pogodbo. V primeru uporabe pametnih pogodb prav tako ni potrebno shranjevati naslova izvlečka na strani ponudnika storitve, saj lahko znotraj pogodbe implementiramo metode, ki za dano objavo (določeno z identifikatorjem) vrne vrednost izvlečka. Uporabniki storitve lahko tako za vsako objavljeno vsebino preverijo, ali je vrednost izvlečka prava, neposredno s klicem metode pametne pogodbe.

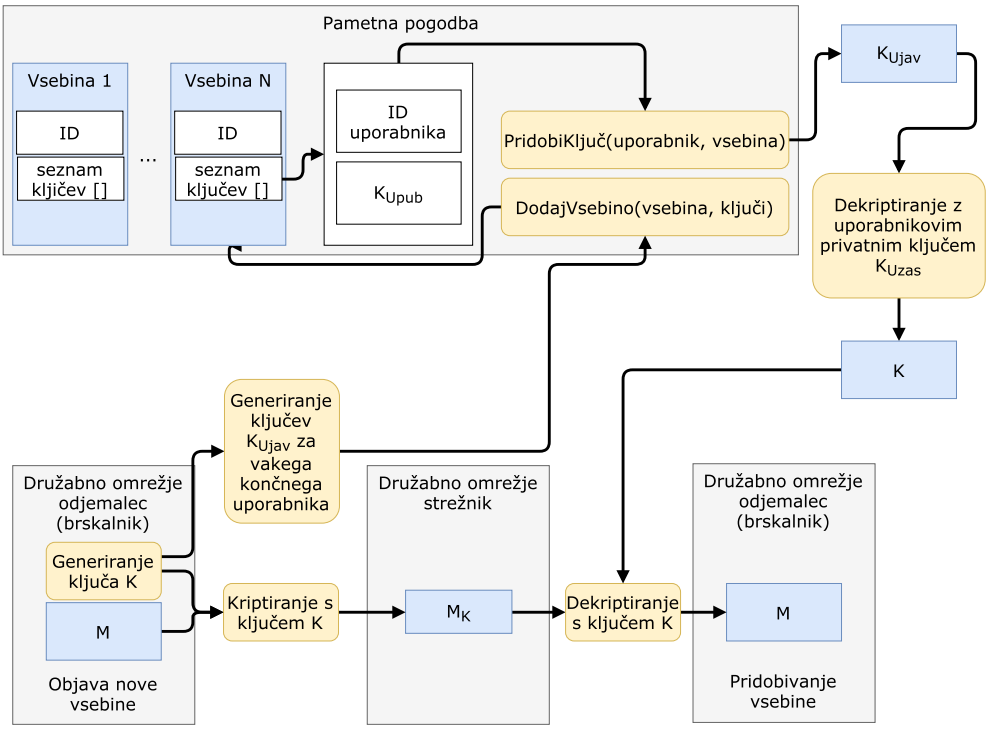


Slika 3.3: Načrt uporabe verige blokov za zagotavljanje integritete podatkov, ki temelji na pametnih pogodbah.

3.3 Načrt sistema za zagotavljanje zaupnosti

Kot smo že omenili, bomo za študijo izvedljivosti uporabe verige blokov za zagotavljanje zaupnosti razvili sistem za dodeljevanje pravic vpogleda. Tudi v tem primeru za zgled aplikacije, ki jo želimo nadgraditi s takšnim mehanizmom vzamemo aplikacijo spletnega družabnega omrežja, čeprav, tako kot v primeru zagotavljanja integritete ni najboljši primer dejanske uporabe (glej poglavje 6), je pa na tem primeru enostavneje razumeti delovanje celotnega sistema. Ponovno torej govorimo o skupini uporabnikov, ki na spletnem družabnem omrežju delijo različne tipe vsebin, le da se tokrat osredotočimo tudi na skupine uporabnikov, katerim so te vsebine namenjene. Vsak uporabnik namreč lahko določi skupino uporabnikov, katerim bo objavljena vsebina vidna. Obstoječi sistemi sicer to že omogočajo, vendar pa v našem primeru želimo pokazati, da lahko vzpostavimo sistem, kjer nad dodeljevanjem pravic vpogleda nima nadzora ponudnik storitve, pač pa je možnost vpogleda dodeljena s pomočjo verige blokov. Prav tako želimo pokazati, da lahko vzpostavimo sistem, katerega zaupnost ni osnovana na zaupanju ponudniku storitve.

V tem primeru skušamo rešitev izdelati s pomočjo pametnih pogodb, saj ne želimo zgolj shranjevati podatkov v verigo blokov, pač pa imeti možnost avtomatičnega izvajanja metod, ki bi dodeljevale pravice vpogleda. Ker ne želimo, da ponudnik storitve v kakršnemkoli primeru omogoči pogled vsebine tretji osebi, kateri vpogled ni bil dovoljen s strani avtorja, bomo sistem zasnovali tako, da bo vsebina, ki bo shranjena na strani ponudnika storitve, kriptirana. Predpostavimo, da vsebino M kriptiramo s simetričnim ključem K, ki je generiran na strani uporabnika, ki objavi vsebino. Vsebina, ki se dejansko shrani na strežnik ponudnika storitve, je kriptirana in jo označimo z M_K. Poleg kriptiranja vsebine avtor hkrati doda zapis v verigo blokov, kjer za ustvarjeno vsebino ustvari seznam parov $(id\_uporabnika, K_{Ujav})$, kjer *id_uporabnika* predstavlja enoličen identifikator uporabnika, kateremu želimo dodeliti pravico vpogleda, K_{Ujav} pa predstavlja ključ K, kriptiran z uporabnikovim(določenim z *id_uporabnika*) javnim ključem. Končni uporabnik bo



Slika 3.4: Načrt uporabe verige blokov za dodeljevanje pravic vpogleda, ki temelji na pametnih pogodbah.

na ta način od pametne pogodbe lahko pridobil K_{Ujav}, ga s svojim zasebnim ključem $Uzas$ dekriptiral, ter na koncu z dobljenim ključem K dekriptiral še vsebino M_K, ki mu jo bo ponudnik storitve posredoval kriptirano. Pametna pogodba bo v tem primeru poleg seznama objav in njim pripadajočih ključev vsebovala še metodo, s katero bo uporabnik lahko objavil novo vsebino (dodal seznam ključev), in pa metodo, ki bo končnemu uporabniku glede na njegov enolični identifikator vrnila pripadajoč ključ. Načrt takšnega sistema prikazuje slika 3.4. Kot tudi pri načrtu rešitve za zagotavljanje integritete podatkov je tudi tu pomembno, da uporabnik komunicira neposredno z verigo blokov - pametno pogodbo.

Poglavje 4

Izbira tehnologij in orodij

V tem poglavju se posvetimo predvsem izbiri glavnega orodja, na katerem temelji naše delo - verigi blokov. Pri izbiri verige blokov se moramo sprva odločiti, ali bomo uporabili obstoječo verigo blokov, ali se bomo lotili izdelave lastne verige blokov. V nadaljevanju na kratko prikažemo zakaj vpeljava lastne verige blokov z naš primer ni najboljša izbira, pregledamo obstoječe ponudnike verige blokov, ter se odločimo za verigo blokov, ki jo bomo uporabili pri implementaciji predlagane rešitve.

4.1 Implementacija lastne verige blokov

Pri implementaciji lastne verige blokov se lahko odločimo med implementacijo verige blokov znotraj organizacije, ali pa vpeljavo nove javne verige blokov. V prvem primeru gre zgolj za obliko porazdeljene podatkovne strukture, saj je veriga blokov povsem nadzorovana s strani organizacije. V primeru javne verige blokov pa gre za implementacijo verige blokov, kakršne so npr. Bitcoin, Ethereum, itd.

Glede na to, da je cilj, ki ga želimo doseči, vpeljava mehanizma, ki bi omogočal višjo stopnjo zaupnosti in integritete podatkov, nam implementacija lastne verige blokov znotraj organizacije ne koristi, saj bi bila v tem primeru stopnja zaupnosti in integritete podatkov odvisna od organizacije,

naš cilj pa je ravno obraten - vpeljava mehanizma, ki za zagotavljanje zaupnosti in integritete podatkov ne potrebuje centralne avtoritete, ki bi ji bilo potrebno zaupati. Izbira javno dostopne verige blokov, v katero se lahko vključi kdorkoli, je tako primerna in smiselna rešitev, vendar le pod pogojem, da takšni verigi blokov lahko zaupamo. Kot smo omenili v poglavju 2, je ena izmed glavnih slabosti verige blokov občutljivost na napad v primeru, da si katerokoli izmed vključenih vozlišč (ang. node) lasti več kot polovico celotne računske moči verige blokov. Pri implementaciji lastne verige blokov bi bilo tako potrebno poskrbeti za dovolj veliko omrežje tako imenovanih računskih vozlišč (ang. miner), kar pripelje do vpeljave nove kripto valute, saj je ravno princip nagrajevanja glavna motivacija za sodelovanje pri potrjevanju blokov. Sistem nagrajevanja (po principu PoW - proof of work) je zgolj eden izmed načinov zagotavljanja zaupanja vredne verige blokov. Drug način, ki bi deloval je princip PoS (proof of stake), a jasno je, da brez načina nagrajevanja veriga blokov ni varna.

To nas pripelje do ugotovitve, da je za zagotavljanje varnosti potrebna predvsem dobro uveljavljena veriga blokov z visoko računsko močjo vključenih vozlišč - večja kot je računska moč, težje je izvesti napad na takšno verigo blokov. Ker je naš cilj vpeljava verige blokov v obstoječe sisteme in je velik del takšnih sistemov premajhen, da bi bilo zanj možno ustvariti zanesljivo verigo blokov, je tako enostavneje uporabiti že uveljavljeno verigo blokov [29], kar pa ne pomeni, da vpeljava lastne verige blokov ne bi bila primerna - bi pa bilo potrebno najprej poskrbeti za varno verigo blokov, šele nato bi lahko z uporabo le-te zagotavljali zaupnost in integriteto podatkov na način, ki smo ga prikazali v poglavju 3.

4.2 Pregled ponudnikov verige blokov

V prejšnjem poglavju smo pokazali, da je vpeljava nove verige blokov možna, vendar pa je za zagotavljanje višje stopnje varnosti, ter posledično zagotavljanje želene stopnje zaupnosti ter integritete podatkov, kakršno navajamo

v poglavju 3, potrebna veriga blokov z dovolj veliko računsko močjo vseh vključenih vozlišč. V nadaljevanju tako pregledamo nekaj največjih javno dostopnih verig blokov, ter ovrednotimo smiselnost uporabe vsake izmed pri implementaciji zastavljene rešitve. Glavne metrike, na podlagi katerih vrednotimo verige blokov so:

- Računska moč omrežja, merjena v številu izvlečkov (ang. hash) na sekundo [H/s]. Večja kot je računska moč omrežja, težje je izvesti "napad 51" na takšni verigi, saj bi napadalec potreboval vsaj polovico celotne računske moči. Podatek o računski moči je seveda pomemben le za verige blokov, ki uporabljajo model Proof of Work.

- Namen uporabe verige blokov - določene verige blokov služijo zgolj točno določenim namenom in jih ni možno, ali pa ni smiselno uporabljati v splošne namene. Kljub temu, da bi bila uporaba takšne verige blokov teoretično možna, se lahko omrežje na podlagi sumljivih transakcij odloči, da takšne transakcije neha potrjevati.

- Podpora pametnim pogodbam bo predvsem igrala pomembno vlogo pri implementaciji rešitve za zagotavljanje zaupnosti, saj smo načrt izdelali na osnovi pametne pogodbe.

V tabeli 4.1 se nahaja seznam javno dostopnih verig blokov, za vsako od njih pa tudi ovrednotene zgoraj navedene metrike. Med podrobnejšim pregledom ponudnikov smo opazili, da je velik del ponudnikov verige blokov še v začetnih fazah razvoja in uporaba takšnih storitev še ni možna. Poleg tega velik del ponudnikov, predvsem tistih, ki se osredotočajo na storitev pametnih pogodb, ne temelji na metodi Proof of Work.

Glede na nabor možnosti smo se pri izdelavi implementacije naše rešitve odločili za izbiro ponudnika *Ethereum*, ki nudi dovolj veliko in dobro vzpostavljeno omrežje, podpira pametne pogodbe in je s stališča uporabe njihovih storitev dobro dokumentiran.

Tabela 4.1: Pregled ponudnikov verige blokov.

Veriga blokov	Računska moč	Namen uporabe	Pametne pogodbe	Opombe
Bitcoin	$5,012,285\ TH/s$	valuta	ne	
Ethereum	$64\ TH/s$	splošno	da	Razširjena uporaba, dobro dokumentirana
Ripple		valuta	ne	Ni povsem javna veriga blokov
Hyperledger		splošno	da	Ni javno dostopna
Hedgy		splošno	da	V fazi razvoja, uporaba še ni možna
RSK	Uporablja omrežje Bitcoin	splošno	da	
R3:Corda		finančne storitve	da	V fazi razvoja, uporaba še ni možna
Blockstream		splošno	da	V fazi razvoja, uporaba še ni možna
SmartContract		splošno	da	Omogoča zgolj pametne pogodbe

4.3 Projekt Ethereum

Projekt *Ethereum* predstavlja decentralizirano platformo, na kateri je možno izvajati pametne pogodbe. Projekt je leta 2014 predstavil avtor Vitalik Buterin [29] kot prvo platformo, osnovano na verigi blokov, ki omogoča izvajanje pametnih pogodb znotraj verige blokov. Danes projekt *Ethereum* predstavlja najbolj razširjeno platformo za izvajanje pametnih pogodb.

V namen izvajanja pametnih pogodb so razvili tako imenovan Ethereum Virtual Machine (EVM), ki omogoča izvajanje pametnih pogodb zapisanih v jeziku *EVM bytecode* v samem omrežju verige blokov. Poleg tega so razvili tudi visoko-nivojski programski jezik *Solidity*, ki omogoča enostavno programiranje pametnih pogodb in ga je mogoče enostavno prevesti v *EVM bytecode*.

Zaradi velike razširjenosti projekta *Ethereum* in predvsem dobre uradne dokumentacije, kot tudi ogromne zbirke resursov s strani vse večje skupnosti, smo se odločili, da za samo implementacijo predlagane rešitve uporabimo projekt *Ethereum*.

4.4 Naivechain

Med pregledom ponudnikov verig blokov smo zasledili tudi zanimiv odprtokodni projekt *Naivechain* (`https://github.com/lhartikk/naivechain`). Gre za enostavno implementacijo poenostavljene verige blokov, katere namen je predvsem na enostaven način prikazati delovanje verige blokov. Vsebuje le najbolj osnovne elemente, ki so potrebni za prikaz principa delovanja verige blokov. Projekt je sicer zastavljen bolj v izobraževalne namene in ni namenjen dejanski uporabi v nekem realnem primeru, vendar pa smo se odločili, da prvo implementacijo naše rešitve izdelamo z uporabo te verige, in na ta način hitreje izdelamo prvi prototip implementacije.

Naivechain je v osnovi sestavljen iz blokov, vsak izmed blokov pa vsebuje zgolj naslednje parametre: indeks, zgoščeno vrednost prejšnjega bloka, zgoščeno vrednost trenutnega bloka, časovni žig in pa polje za poljubne po-

datke. Transakcij veriga blokov ne vsebuje, po želji pa jih lahko uporabnik v blok vključi znotraj polja za poljubne podatke. Ker v našem načrtu od verige blokov pričakujemo, da vsebuje vsaj eno polje, kjer je možno shraniti poljuben podatek manjše velikosti, nam takšen koncept verige povsem ustreza. *Naivechain* je seveda zasnovan tako, da omogoča dejansko postavitev omrežja tipa vsak z vsakim, ter s tem kar se le da posnema obnašanje prave verige blokov.

Poglavje 5

Implementacija

V tem poglavju podrobneje predstavimo implementacijo rešitev predlaganih v poglavju 3. Sprva po korakih razvijemo rešitev za zagotavljanje integritete podatkov - od uporabe preproste verige blokov, pa do uporabe pametnih pogodb. Nato z uporabo pametnih pogodb razvijemo še rešitev za dodeljevanje pravic vpogleda. V zadnjem delu poglavja prikažemo, kako je možno razvite rešitve integrirati v obstoječe sisteme.

5.1 Implementacije rešitve za zagotavljanje integritete podatkov

Rešitev, predlagano v poglavju 3.2, razvijemo v treh korakih:

- V prvem koraku se odločimo za uporabo odprtokodne poenostavljene implementacije verige blokov imenovane Naivechain. Namen tega koraka je na enostaven način preveriti delovanje predlagane rešitve. Pri uporabi prave verige blokov se lahko hitro pojavijo težave ob nepravilni uporabi, zato želimo delovanje rešitve preveriti v okolju, kjer se ni potrebno posvečati konfiguraciji odjemalcev, vključenih v verigo blokov, pač pa zgolj implementaciji rešitve. Drugi razlog za uporabo testne verige blokov je tudi ta, da pri takšni verigi nismo del pravega omrežja,

kar pomeni, da s testiranjem pri implementaciji rešitve v verigo blokov ne vnašamo nepotrebnih podatkov in po nepotrebnem širimo velikost same verige blokov. Poleg tega je potrebno v pravem omrežju vsako izmed transakcij tudi plačati, kar prinaša dodatne stroške, ki se jim z uporabo testne verige blokov izognemo.

- V drugem koraku se lotimo implementacije rešitve še na pravi verigi blokov. V ta namen izberemo verigo blokov Ethereum. Tudi v tem primeru pa testiranja ne izvajamo znotraj pravega omrežja, pač pa uporabimo sprva zgolj odjemalec *testrpc*, ki znotraj lokalnega omrežja simulira pravo Ehtereum verigo blokov.

- V zadnjem koraku po načrtu iz poglavja 3.2 rešitev implementiramo še z uporabo pametnih pogodb, kjer zopet uporabimo verigo blokov Ethereum.

V vseh treh korakih v nadaljevanju opišemo zgolj implementacijo komunikacije z verigo blokov oziroma implementacijo pametne pogodbe in komunikacijo z le-to. Integracijo v obstoječo aplikacijo podrobneje opišemo v poglavju 5.3, za zdaj pa predpostavimo, da vsa komunikacija poteka s strani odjemalca - v spletnem brskalniku. Z ozirom na to vse klice tako izvajamo v programskem jeziku *JavaScript*.

5.1.1 Uporaba verige blokov Naivechain

V poglavju 4.4, kjer smo verigo blokov *Naivechain* podrobneje opisali, smo pokazali, da nam veriga blokov omogoča vnos podatkov v polje *data*, ki lahko vsebuje poljuben podatek. V našem primeru bomo to polje uporabili za shranjevanje izvlečka vsebine. Predpostavimo, da se račun izvlečka vsebine izvede v uporabnikovem brskalniku. Naš cilj je torej zgolj vnos izvlečka v blok. Za zdaj lahko predpostavimo, da izvleček izračunamo s pomočjo *JavaScript* knjižnice *jsSHA* z naslednjim zaporedjem ukazov:

```
var shaObj = new jsSHA("SHA-256", "TEXT");
```

```
shaObj.update("<vsebina>");
var hash = shaObj.getHash("HEX");
```

Ker se veriga blokov *Naivechain* izvaja zgolj znotraj lokalnega okolja in ne predstavlja javno dostopne storitve, jo moramo najprej zagnati na lokalnem računalniku. Najprej je potrebno prenesti izvorno kodo z javno dostopnega repozitorija, nato pa verigo blokov namestimo in zaženemo z naslednjim zaporedjem ukazov:

```
npm install

# v prvem oknu terminala pozenemo:
HTTP_PORT=3001 P2P_PORT=6001 npm start

# v drugem oknu terminala pozenemo:
HTTP_PORT=3002 P2P_PORT=6002 PEERS=ws://localhost:6001\
npm start
```

Na ta način smo pognali verigo blokov v simuliranem okolju, kjer omrežje predstavljata dve vozlišči. Delovanje verige blokov lahko enostavno preverimo preko brskalnika na naslovu `http://localhost:3001/blocks`. Na ekranu se nam izpiše seznam vseh blokov, v katerem se na začetku nahaja zgolj en blok.

Naslednji korak je vnos izvlečka v verigo blokov, kar preprosto naredimo s klicem HTTP zahtevka na eno izmed vozlišč v omrežju *Naivechain* (ponovno predpostavimo, da je bila predhodno naložena knjižnica *jQuery*):

```
$.post("http://localhost:3001/mineBlock",
    {data: "<vrednost_izvlecka>"}
);
```

V kolikor ponovno preverimo seznam vseh blokov (`http://localhost:3001/blocks`). Da preverimo bolj realen scenarij, dodajanje izvlečkov izvedemo iz več različnih virov (poženemo z različnih sej v brskalniku) in pa z uporabo različnih vozlišč (na portih 3001 in 3002). V omrežje nato dodamo

še nekaj novih vozlišč in preverimo, ali so v verigah blokov na vseh vozliščih shranjeni zapisi o izvlečkih vsebin. Na enostaven način tako preverimo, da je implementacija predlagane rešitve možna na poenostavljeni verigi blokov. V nadaljevanju skušamo na enak način rešitev implementirati še z uporabo prave verige blokov.

5.1.2 Uporaba verige blokov Ethereum

Da pokažemo, da je rešitev možno implementirati tudi z uporabo prave verige blokov, v tem koraku uporabimo verigo blokov Ethereum, v katero nameravamo, glede na predlagan načrt, izvlečke vsebine shranjevati znotraj transakcij, v posebno polje *paylod*. Tudi tokrat predpostavimo, da komunikacija z verigo blokov poteka neposredno s strani brskalnika.

V namen testiranja in razvijanja rešitve pri verigi blokov *Ethereum* povezava v pravo omrežje ni potrebna, saj nam je na voljo odjemalec *testrpc*, ki je navzven identičen pravemu odjemalcu, le da pri tem vzdržuje svojo lastno verigo blokov kar v pomnilniku lokalnega računalnika. Poleg tega omogoča tudi takojšnje potrjevanje blokov, kar pospeši razvojni čas. Kasneje, ko rešitev razvijemo, enostavno zgolj spremenimo naslov odjemalca na pravega, in naša rešitev nato teče na pravi verigi blokov. Za uporabo je najprej potrebna namestitev odjemalca:

```
npm install -g ethereumjs-testrpc
```

Odjemalca nato poženemo z ukazom:

```
testrpc
```

Odjemalec nam ob zagonu kreira prazno verigo blokov, hkrati pa ustvari 10 vnaprej pripravljenih računov, na katerih so že naložena sredstva v digitalni valuti ETH. Ta sredstva so seveda uporabna zgolj v namen testiranja in jih v pravem omrežju ne moremo koristiti. Pustimo, da odjemalec teče v ozadju.

Spletni brskalnik za komunikacijo z odjemalcem *testrpc* potrebuje knjižnico

*web*3, ki ni namenjena le komunikaciji s *testrpc*, pač pa gre za uradno knjižnico, namenjeno komunikaciji z verigo blokov *Ethereum*. Edina razlika med uporabo pravega odjemalca in uporabo *testrpc* je pri začetni vzpostavitvi povezave z odjemalcem, kjer nastavimo naslov odjemalca. V našem primeru bo to naslov, ki ga *testrpc* izpiše ob zagonu (`localhost:8545`). Za vzpostavitev povezave v brskalniku poženemo naslednji ukaz, ki ustvari objekt *web*3, katerega kasneje uporabljamo za komunikacijo z verigo blokov:

```
web3 = new Web3(new Web3.providers.HttpProvider(
    "http://localhost:8545"
));
```

V načrtu smo predvideli shranjevanje izvlečkov vsebine znotraj transakcije. Za kreiranje nove transakcije nam je na voljo metoda *sendTransaction*(), ki se nahaja v objektu *web*3.*eth*. Pri kreiranju nove transakcije je potrebno definirati parametra *from* in *to*, ki predstavljata vhodne in izhodne naslove. Ker nam *testrpc* ob zagonu ustvari 10 vnaprej pripravljenih računov, bomo v namen testiranja uporabil kar dva izmed teh naslovov. K vsaki transakciji je potrebno pripeti tudi *gas*, ki predstavlja neko majhno količino virtualnega denarja in služi kot neke vrsto plačilo provizije za uspešno potrjevanje transakcije. Poleg tega nam, kot smo že omenili *Ethereum* omogoča, da transakciji pripnemo poljubne podatke. To storimo z definiranjem parametra *data*. Transakcijo, v kateri je vključen naš izvleček izvedemo z naslednjim klicem:

```
web3.eth.sendTransaction({
    from:web3.eth.accounts[0],
    to:web3.eth.accounts[1],
    gas: 1000000,
    data:"0x<vrednost_izvlecka>"
});
```

Uspešen klic transakcije nam vrne naslov izvedene transakcije. Kot smo omenili že v načrtu, bomo ta naslov shranili na strani ponudnika storitve

hkrati z objavo vsebine, ter s tem zagotovili prikaz naslova izvlečka tudi vsem ostalim uporabnikom. Da preverimo, ali je transakcija vsebovana v verigi blokov, in da le-ta vsebuje tudi izvleček vsebine, uporabimo naslov transakcije, ki je rezultat prejšnjega ukaza, ter pokličemo metodo, ki pridobi informacije o transakciji:

```
web3.eth.getTransaction("0x<naslov_transakcije>");
```

Izvleček, ki smo ga transakciji podali pod parametrom *data*, se nahaja v polju *input*, poleg tega pa nam metoda vrne tudi podatek o tem, v katerem bloku je transakcija vsebovana.

Ponovno testiramo takšen način uporabe z različnih sej brskalnikov, ter preverimo, ali so znotraj verige blokov vsebovani vsi izvlečki. S tem smo prikazali, da je rešitev izvedljiva tudi z uporabo prave verige blokov, kjer v samo transakcijo vgradimo dodatne podatke.

5.1.3 Uporaba pametne pogodbe

Rešitev v tem koraku skušamo implementirati še z uporabo pametne pogodbe. Tudi v tem primeru bomo implementacijo izvedli z uporabo verige blokov *Ethereum*, ki pametne pogodbe omogoča. Tako kot v prejšnjem koraku bo tudi tokrat spletni brskalnik vso komunikacijo izvajal preko objekta *web*3, vendar pa je v tem primeru potrebno najprej ustvariti pametno pogodbo in jo shraniti v verigo blokov. V ta namen razvijemo sledečo pametno pogodbo:

```
pragma solidity ^0.4.0;

contract ShrambaIzvleckov {

    mapping(uint => string) public izvlecki;

    function dodajVsebino(string izvlecek,
            uint id_vsebine) public {
```

```
        izvlecki[id_vsebine] = izvlecek;
    }
}
```

Podoba, imenovana *ShrambaIzvleckov*, vsebuje spremenljivko *izvlecki*, ki je neke vrste slovar, pri katerem za ključ uporabljamo celo število, vrednost pa je tekstovne oblike. Kot smo zastavili v načrtu, pogodba vsebuje metodo *dodajVsebino*, namenjeno dodajanju novega izvlečka, metode za pridobivanje izvlečka za dano vsebino pa v našem primeru ne potrebujemo, saj je spremenljivka *izvlecki* definirana kot *public* in lahko do njene vrednosti dostopamo neposredno.

Za vnos pametne pogodbe v verigo blokov je pametno pogodbo, napisano v jeziku *solidity*, potrebno najprej prevesti, in prevedeno kodo objaviti v verigi blokov:

```
var izvornaKoda = '<izvorna_koda>'
var prevedenaPogodba =  web3.eth.compile.solidity(
    izvornaKoda
);
var ShrambaIzvleckov = web3.eth.contract(
    prevedenaPogodba.info.abiDefinition
);
var objavljenaPogodba = ShrambaIzvleckov.new({
    data: prevedenaPogodba.code,
    from: web3.eth.accounts[0],
    gas: 5000000
});
```

Končni uporabnik objavljeno pogodbo uporablja tako, da najprej ustvari instanco pogodbe, ki se nahaja na naslovu, kjer smo pogodbo objavili:

```
InstancaShrambaIzvleckov = ShrambaIzvleckov.at(
    objavljenaPogodba.address
);
```

Izvleček vsebine v pogodbo shranimo s klicem metode *dodajVsebino*, ki pa jo pokličemo tako, da nad metodo izvedemo transakcijo, kateri podamo vhodne parametre metode in pa zadostno količino *gas* denarja, da bo transakcija uspešno potrjena s strani omrežja:

```
InstancaShrambaIzvleckov.dodajVsebino.sendTransaction(
    <izvlecek>,
    <id_vsebine>,
    {from:web3.eth.accounts[0], gas: 1000000}
);
```

Tako objavljen izvleček nato pridobimo tako, da neposredno dostopamo do spremenljivke *izvlecki* znotraj instance naše pogodbe:

```
InstancaShrambaIzvleckov.izvlecki(<id_vsebine>);
```

Delovanje zopet preverimo z uporabo različnih sej v brskalnikih, ter pokažemo, da je implementacija rešitve možna tudi z uporabo pametne pogodbe. Prednost uporabe pametne pogodbe je tudi v tem, da na strani ponudnika storitve ni potrebno shranjevati naslova, kjer se izvleček nahaja, saj nam pametna pogodba omogoča pridobivanje izvlečka glede na poznan enolični identifikator vsebine *id_vsebine*.

5.2 Implementacija rešitve za dodeljevanje pravic vpogleda

Rešitev sistema za dodeljevanje pravic vpogleda razvijemo izključno z uporabo pametnih pogodb. Tudi v tem primeru uporabimo verigo blokov Ethereum in programski jezik *Solidity*:

```
pragma solidity ^0.4.0;

contract DodeljevanjePravic {

```

```
    struct Vsebina {
        mapping(uint => byte[100]) kljuci;
    }

    mapping(uint => Vsebina) senzamVsebin;

    function dodajVsebino(uint id_vsebine,
                uint[] seznam_uporabnikov,
                byte[100][] seznamKljucev) public {
        senzamVsebin[id_vsebine] = Vsebina();
        Vsebina v = senzamVsebin[id_vsebine];
        v.kljuci[seznam_uporabnikov[0]] =
            seznamKljucev[0];
    }

    function dodajPravico(uint id_vsebine,
                uint id_uporabnika,
                byte[100] kljuc) public {
        Vsebina v = senzamVsebin[id_vsebine];
        v.kljuci[id_uporabnika] = kljuc;
    }

    function pridobiKljuc(uint id_vsebine,
                uint id_uporabnika) public {
        Vsebina v = senzamVsebin[id_vsebine];
        v.kljuci[id_uporabnika];
    }
}
```

Skladno z načrtom za hranjenje vsebin ustvarimo nov podatkovni tip imenovan *Vsebina*, ki vsebuje slovar ključev končnih uporabnikov, katerim želimo omogočiti pogled v vsebino. Pametna pogodba nato vsebuje slo-

var vsebin, kjer enoličnemu identifikatorju vsebine pripada podatkovni tip $Vsebina$. Za zapis nove vsebine in pripadajočih ključev v verigo blokov imamo na voljo metodo $dodajVsebino$, kateri je potrebno podati enolični identifikator vsebine $id\_vsebine$, seznam enoličnih identifikatorjev uporabnikov, katerim želimo omogočiti vpogled v vsebino, in pa seznam ključev za vsakega izmed uporabnikov. Pomembno je, da pri klicu seznam uporabnikov in seznam ključev ustvarimo tako, da zapisi seznamov paroma ustrezajo uporabnikom. Končni uporabnik ključ za vpogled v vsebino pridobi s klicem metode $pridobiKljuc$, kateri je potrebno podati enolični identifikator vsebine in pa uporabnika. Na tem mestu je potrebno poudariti, da so vse metode pametne pogodbe javne, in do njih lahko dostopa vsak uporabnik, prav tako pa lahko vsak uporabnik metode pokliče s poljubnimi podatki - npr. z identifikatorjem drugega uporabnika. Takšen klic sicer ne predstavlja varnostne pomanjkljivosti sistema, saj so vsi ključi predhodno kriptirani z uporabnikovim javnim ključem in je le uporabnik, kateremu je ključ namenjen, tisti, ki lahko s pomočjo svojega zasebnega ključa pridobi ključ K.

V zgoraj prikazani pametni pogodbi se nahaja še metoda $dodajPravico$, ki je namenjena naknadnemu dodajanju pravic vpogleda. Avtor, ki pozna ključ K, lahko ustvari nov zapis v seznamu ključev, tako da ključ kriptira z javnim ključem uporabnika, kateremu želi omogočiti vpogled v vsebino, ter le-tega posreduje pametni pogodbi skupaj z identifikatorjem vsebine, za katero vpogled omogoča. Podobno razvijemo še metodo $odstraniPravico$, ki omogoča odstranitev ključa s sezama.

Pametno pogodbo je na enak način kot v prejšnjem primeru potrebno najprej prevesti ter objaviti v verigi blokov. Ko je pogodba objavljena, lahko do nje dostopa vsak, ki pozna naslov, na katerem je objavljena. Komunikacijo s pametno pogodbo ponovno izvajamo preko knjižnice *web*3 in pa z uporabo testnega odjemalca *testrpc*, primer izvajanja klicev metod pa je prikazan v prejšnjem poglavju.

5.3 Integracija predlaganih rešitev v obstoječe sisteme

V prejšnjih poglavjih smo se osredotočili zgolj za implementacijo rešitve na strani uporabe verige blokov, v tem poglavju pa prikažemo na kakšen način lahko komunikacijo z verigo blokov integriramo v nek obstoječ sistem. Kot smo večkrat poudarili že v prejšnjih poglavjih, je pomembno, da komunikacija z verigo blokov poteka neposredno s strani končnega uporabnika, kar v praksi pomeni, da komunikacijo izvaja uporabnikov brskalnik. Za integracijo bo torej ponudnik storitve moral omogočiti vgradnjo dodatne funkcionalnosti na strani odjemalca, za kar je najbolj preprosto uporabiti programski jezik *JavaScript*, katerega podpira večina današnjih brskalnikov. V namen testiranja integracije smo razvili preprosto storitev spletnega družabnega omrežja, ki omogoča kreiranje uporabnikov in objavljanje vsebin. Na strani odjemalca je nato potrebno naložiti knjižnico *web*3 in vzpostaviti povezavo z vozliščem povezanim v verigo blokov, ki je v našem primeru kar odjemalec *testrpc*:

```
<script src="web3.min.js"></script>
<script>
    web3 = new Web3(new Web3.providers.HttpProvider(
        "http://localhost:8545"
    ));
</script>
```

Po vzpostavitvi povezave z verigo blokov pa klice izvajamo glede na to, katero rešitev želimo integrirati, način komunikacije pa je že v prejšnjih poglavjih podrobneje opisan za vsako izmed rešitev. V vsakem primeru se integrirana rešitev izvaja le s strani odjemalca in ponudniku storitve sistema na strani strežnika ni potrebno dodatno spreminjati, kar omogoča enostavno vključitev. Edina izjema pri tem je eden izmed korakov implementacije za shranjevanje izvlečkov, ki deluje na osnovi vgrajevanja izvlečka v transakcijo. Pri tej rešitvi je namreč potrebno na strani ponudnika storitve shraniti

naslov transakcije, kjer je shranjen izvleček. Izvedljivost integracije takšnega sistema smo na poenostavljenem primeru aplikacije sicer preverili, vendar pa morda takšen način ni primeren za uporabo v splošnem primeru, saj je pri tem potrebno omogočiti shranjevanje naslovov na strani ponudnika, kar morda v nekaterih okoljih ni izvedljivo. Kljub vsemu lahko za shranjevanje izvlečkov še vedno uporabimo rešitev, ki uporablja pametne pogodbe.

Poglavje 6

Analiza razvite rešitve

V tem poglavju sledi pregled implementacij rešitev, predlaganih v prejšnjih poglavjih, kjer pokažemo predvsem potencialne nevarnosti, ki jih razvite rešitve prinašajo in pa predlagamo izboljšave, ki bi takšne nevarnosti odpravile v kolikor je to možno. Poleg potencialnih nevarnosti prikažemo tudi stroškovno analizo dejanske uporabe predlaganega sistema in za konec predstavimo še eno večjih tehničnih omejitev, ki jih vpeljava takšnega sistema prinaša - čakanje na potrditev transakcije, preden je le ta vključena v verig blokov.

6.1 Pomanjkljivosti predlaganih rešitev

Med testiranjem implementiranih rešitev smo opazili pomanjkljivosti, ki bi utegnile predstavljati varnostne luknje. V nadaljevanju sledi pregled pomanjkljivosti, za vsako pa tudi predlagamo rešitev, v kolikor je ta možna:

Spreminjanje naslova pogodbe Kljub temu, da končni uporabnik v vseh predlaganih rešitvah komunicira neposredno z verigo blokov, je integracija zasnovana tako, da je naslov pametne pogodbe shranjen na strani ponudnika. To dopušča možnost za napad, pri katerem ponudnik storitve namerno preusmeri komunikacijo na lastno pametno pogodbo. Temu se sicer uporabniki lahko izognejo tako, da pred vsako

uporabo ročno izvedejo povezavo na želeno pametno pogodbo, vendar s tem otežimo uporabo aplikacije. Problem lahko delno rešimo tako, da v kolikor uporabnik želi, sam ustvari povezavo, v kolikor pa uporabnik zaupa ponudniku, pa izbiro naslova pogodbe prepusti njemu. Tudi v primeru, ko bi ponudnik storitve izbral drug naslov pogodbe, lahko uporabnik vidi, s katero pogodbo poteka komunikacija, ter s tem pokaže na nepoštenost ponudnika.

Neželeni vnosi (spam) Ker komunikacija z verigo blokov poteka neposredno od končnega uporabnika, lahko le-ta v verigo blokov vnaša neželene vnose ter s tem ponudniku povzroča dodatne stroške. Temu se lahko izognemo tako, da ponudnik storitve uporabniku za vsak željen vnos dodeli žeton, s katerim uporabniku omogoči vnos vsebine. Drug način reševanja problema je omejitev števila vnosov neposredno znotraj pametne pogodbe - če bi pametna pogodba zaznala, da uporabnik vnaša ogromne količine podatkov, takšnih akcij ne bi izvedla.

Nepooblaščeno spreminjanje vsebine Pri obeh implementacijah rešitev, kjer smo uporabili pametne pogodbe, lahko vsebino le-te spreminja kdorkoli. Tako lahko nepooblaščena oseba spremeni vrednost izvlečka in posledično zavaja uporabnike, saj bo sistem kazal na to, da vsebina, ki jo prikazuje ponudnik storitve ni skladna s tem, kar je objavil avtor. V primeru dodeljevanja pravic prav tako lahko nepooblaščena oseba spremeni ključe, s katerimi je končnim uporabnikom omogočen pogled v vsebino - pri tem sicer ne more dodajati novih pravic, saj ne pozna ključa K, vendar pa lahko obstoječim uporabnikom onemogoči vpogled s spremembo ključa. V obeh primerih je rešitev v shranjevanju podatka o avtorju in kasneje v preverjanju, ali transakcijo izvaja avtor vsebine.

6.2 Stroškovni vidik uporabe implementiranih rešitev

V nadaljevanju predpostavimo, da pri implementaciji rešitev uporabljamo izključno verigo blokov *Ethereum* in stroškovno analizo izdelamo za uporabo le-te.Za vsako uspešno potrjeno transakcijo je v verigi blokov potrebno plačati nekaj enot, imenovanih *gas*. Za vsako operacijo znotraj verige blokov je določeno, koliko *gas* enot stane. Cena ene *gas* enote se spreminja in je v času pisanja približno $0,000000005 ETH$, kar je približno $0,00000133 EUR$. V tabeli 6.1 je za vsako izmed implementacij predlaganih rešitev prikazano število *gas* enot, ki so potrebne za uspešno izvedbo, hkrati pa je glede na trenutno vrednost *gas* prikazana še strošek v valuti EUR.

Opazimo, da cene uporabe verige blokov niso zanemarljivo majhne. V primeru integracije v obstoječ sistem bi tako bilo potrebno poskrbeti za sredstva, namenjena uporabi verige blokov. Pri implementaciji smo za zgled obstoječe aplikacije uporabili aplikacijo spletnega družabnega omrežja, a smo dodali, da integracija v tak sistem ni najbolj smiselna. Ob ogromnih količinah podatkov, ki jih uporabniki takih omrežij objavljajo, bi takšen sistem predstavljal opazen strošek, prav tako pa tega stroška ne moremo prenesti na končnega uporabnika, saj je uporaba takšnih aplikacij v večini primerov brezplačna. Po drugi strani uporaba rešitve za shranjevanje izvlečkov ne bi predstavljala finančne težave v primeru sistema, ki nudi dokaz o obstoju določenega dokumenta, saj bi strošek poravnal končni uporabnik v zameno za opravljeno storitev.

6.3 Tehnične omejitve

Tehnična omejitev, ki za vpeljavo predlagane rešitve o obstoječ sistem lahko predstavlja problem, je predvsem čas, ki je potreben, da je transakcija uspešno potrjena s strani omrežja. Čas, potreben za potrditev transakcije, je pogojen s časom, ki je potreben za kreiranje novega bloka, kar pa je odvisno od

Tabela 6.1: Predvideni stroški uporabe implementiranih rešitev.

Rešitev	*gas* enot	strošek v EUR
Shranjevanje izvlečkov v transakcijo	23176	$0,03$
Klic metode $dodajVsebino()$ pri shranjevanju izvlečkov z uporabo pametne pogodbe	107431	$0,14$
Pridobivanje vrednosti izvlečka pri uporabi pametne pogodbe	0	0
Klic metode dodajVsebino() pri dodeljevanju pravic vpogleda (dodajanje enega ključa)	756252	$1,00$
Klic metode dodajVsebino() pri dodeljevanju pravic vpogleda (dodajanje dveh ključev)	758596	$1,01$
Klic metode dodajPravico() pri dodeljevanju pravic vpogleda	739514	$0,98$
Klic metode pridobiKljuc() pri dodeljevanju pravic vpogleda	21907	$0,03$

izbire ponudnika verige blokov. V omrežju *Bitcoin* je ta čas približno 10 minut[1], v omrežju *Ethereum* pa se novi bloki kreirajo na približno 15 sekund, vendar v prihodnje nameravajo ta čas zmanjšati na 12 sekund [30]. V primeru, da želimo povečati stopnjo varnosti, je potrebno počakati, da je poleg obstoječega bloka potrjenih še nekaj naslednjih blokov.

Glede na to, da smo pri implementaciji rešitve uporabili verigo blokov *Ethereum*, lahko torej pričakujemo, da za uspešen vnos vsebine v verigo blokov čakamo v povprečju 8 sekund. Čakanje na potrditev ne zadeva zgolj implementacij, kjer za hranjene izvlečkov uporabljamo transakcije, pač pa zadeva tudi vsak klic pametne pogodbe, ki je prav tako transakcija. Zaradi te omejitve integracija takšne rešitve ni možna v primerih, kjer želimo, da so informacije zapisane v realnem času. Zakasnitev do 15 sekund na primer ni sprejemljiva v aplikacijah, ki so namenjene izmenjavi sporočil, po drugi strani pa na primer pri storitvi za objavljanje novic ali člankov takšna zakasnitev ne bi imela večjih posledic na celoten sistem.

Poglavje 7

Sklepne ugotovitve

Tekom dela smo uspešno zastavili načrt rešitve tako za zagotavljanje integritete podatkov, kjer smo predlagali sistem za shranjevanje izvlečkov, kot tudi za zagotavljanje zaupnosti, kjer smo predlagali sistem za dodeljevanje pravic vpogleda. Obe predlagani rešitvi smo implementirali z uporabo verige blokov *Ethereum* in jih nato integrirali v obstoječo rešitev, za kar smo izbrali enostavno implementacijo spletnega družabnega omrežja. S tem smo izvedli študijo izvedljivosti uporabe verige blokov v namen zagotavljanja tako zaupnosti kot tudi integritete podatkov. Za predlagane implementacije smo na koncu opredelili pomanjkljivosti in omejitve ter prišli do zaključka, da je uporaba predlagane rešitve možna zgolj v primerih, kjer krajše časovne zakasnitve pri vnosu v verigo blokov ne predstavljajo težav in pa kjer so obstoječe rešitve zasnovane tako, da je možno upravičiti visoko ceno izvajanja operacij v verigi blokov. Ker je veriga blokov relativno nova tehnologija, razvoj le-te pa vse bolj v porastu, je sicer pričakovati, da bo v prihodnje možno odpraviti ali vsaj zmanjšati vpliv teh omejitev.

Literatura

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf (2008).

[2] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, Blockchain technology: Beyond bitcoin, Applied Innovation 2 (2016) 6–10.

[3] R. C. Merkle, A Digital Signature Based on a Conventional Encryption Function, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 369–378. doi:10.1007/3-540-48184-2_32.
URL http://dx.doi.org/10.1007/3-540-48184-2_32

[4] R. C. Merkle, Protocols for public key cryptosystems, in: Security and Privacy, 1980 IEEE Symposium on, IEEE, 1980, pp. 122–122.

[5] K. J. O'Dwyer, D. Malone, Bitcoin mining and its energy footprint, in: 25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014), 2014, pp. 280–285. doi:10.1049/cp.2014.0699.

[6] S. King, S. Nadal, Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, self-published paper, August 19.

[7] I. Bentov, C. Lee, A. Mizrahi, M. Rosenfeld, Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y, ACM SIGMETRICS Performance Evaluation Review 42 (3) (2014) 34–37.

[8] A. Kiayias, A. Russell, B. David, R. Oliynykov, Ouroboros: A provably secure proof-of-stake blockchain protocol, Tech. rep., Cryptology ePrint Archive, Report 2016/889, 2016. http://eprint. iacr. org/2016/889 (2016).

[9] M. Pilkington, Blockchain Technology: Principles and Applications, Handbook of Research on Digital Transformations, 2016, pp. 225–364.

[10] Proof of existence, `https://proofofexistence.com/`.

[11] Filament, `https://storj.io/`.

[12] Filament, `https://filament.com/`.

[13] Blockverify, `http://www.blockverify.io/`.

[14] Namecoin, `https://namecoin.org/`.

[15] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9).

[16] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151.

[17] F. Tschorsch, B. Scheuermann, Bitcoin and beyond: A technical survey on decentralized digital currencies, IEEE Communications Surveys & Tutorials 18 (3) (2016) 2084–2123.

[18] J. Yli-Huumo, D. Ko, S. Choi, S. Park, K. Smolander, Where is current research on blockchain technology?—a systematic review, PloS one 11 (10) (2016) e0163477.

[19] G. Karame, On the security and scalability of bitcoin's blockchain, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 1861–1862.

[20] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün, On scaling decentralized blockchains, in: Proc. 3rd Workshop on Bitcoin and Blockchain Research, 2016.

[21] F. Tschorsch, B. Scheuermann, Bitcoin and beyond: A technical survey on decentralized digital currencies, IEEE Communications Surveys Tutorials 18 (3) (2016) 2084–2123. doi:10.1109/COMST.2016.2535718.

[22] G. Hurlburt, Might the blockchain outlive bitcoin?, IT Professional 18 (2) (2016) 12–16.

[23] K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the internet of things, IEEE Access 4 (2016) 2292–2303. doi:10.1109/ACCESS.2016.2566339.

[24] A. Bahga, V. K. Madisetti, Blockchain platform for industrial internet of things, Journal of Software Engineering and Applications 9 (10) (2016) 533.

[25] G. Zyskind, O. Nathan, et al., Decentralizing privacy: Using blockchain to protect personal data, in: Security and Privacy Workshops (SPW), 2015 IEEE, IEEE, 2015, pp. 180–184.

[26] A. Azaria, A. Ekblaw, T. Vieira, A. Lippman, Medrec: Using blockchain for medical data access and permission management, in: 2016 2nd International Conference on Open and Big Data (OBD), 2016, pp. 25–30. doi:10.1109/OBD.2016.11.

[27] L. Bahack, Theoretical bitcoin attacks with less than half of the computational power (draft), arXiv preprint arXiv:1312.7013.

[28] Ethereum apps, http://http://dapps.ethercasts.com/.

[29] V. Buterin, et al., A next-generation smart contract and decentralized application platform, https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_

generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (2014).

[30] Toward a 12 second block time, https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/.