Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Aljaž Simonič

ANALIZA ODZIVNOSTI KRIPTOSISTEMOV Z DIGITALNIMI POTRDILI

Diplomsko delo univerzitetnega študija

Mentor: prof. dr. Aleksandar Jurišić

Ljubljana, 2008

Povzetek

V kriptografiji vse bolj prihaja v ospredje kriptosistem javnih ključev, ki temelji na teoriji eliptičnih krivulj, imenovan EC kriptosistem in postaja resna alternativa danes najbolj razširjenemu kriptosistemu javnih ključev RSA.

Naloga diplomskega dela je bil izdelati primerjavo povprečnih odzivnih časov spletnega strežnika pri uporabi protokola SSL z digitalnimi potrdili različnih dolžin ključev obeh kriptosistemov. V ta namen je bila izdelana aplikacija, ki je na spletni strežnik izvajala zahteve ter za vsako zahtevo posebej merila odzivni čas njene strežbe. Obremenitveni testi so bili narejeni za vsako v programski opremi podprto dolžino ključev obeh kriptosistemov, testom pa je sledila analiza rezultatov.

Kot se je izkazalo, je EC kriptosistem hitrejši od RSA pri daljših dolžinah ključev, kar je bilo tudi v skladu z teoretičnimi analizami. V nasprotju s pričakovanji, pa se je RSA kriptosistem izkazal za hitrejšega pri krajših dolžinah ključev, kar bi lahko bila tudi posledica slabše optimizirane implementacije EC algoritmov. Za podrobnejšo primerjavo obeh kriptosistemov bi bili potrebni odzivni časi vseh primerljivih dolžin ključev, ki pa jih zaradi omejitev programske opreme ni bilo mogoče izmeriti.

Ključne besede: kriptografija, kriptosistem, digitalno potrdilo, RSA, ECC, Apache, OpenSSL, analiza odzivnosti, eliptična krivulja

Abstract

In the world of public key cryptography elliptic curve cryptosystem, also known as EC cryptosystem is gaining momentum. It is becoming a serious alternative to today's most popular public key cryptosystem RSA

The goal of this graduate thesis was to produce a comparison of average response times of a web server with SSL protocol using digital certificates of different key sizes of both cryptosystems. For this reason a computer application that could measure response time of each request to a web server was constructed. Workload tests have been executed for each of the software supported cryptosystem key sizes and afterwards results analysis followed.

As it turns out EC cryptosystem proved to be faster than RSA at longer key sizes which was in accordance with theoretical analysis. However RSA was the faster one at shorter key sizes. The cause for that might lie in less optimized EC algorithms. For a more detailed comparison of the two cryptosystems all of the response time tests would have been required. That was due to the software limitations impossible to produce.

Keywords: cryptography, cryptosystem, digital certificate, RSA, ECC, Apache, OpenSSL, response analysis, elliptic curve

Kazalo

Kazalo slik

Kazalo tabel

Poglavje 1

Uvod

Ameriška agencija za državno varnost (National Security Agency - NSA) je leta 2005 izdala dokument z naslovom Suite B (glej [6]), v katerem za javno uporabo priporoča uporabo kriptografijo z eliptičnimi krivuljami oziroma EC (Elliptic Curve) kriptosistema kot nadomestilo danes najbolj razširjenega kriptosistema RSA, ki so ga razvili Rivest, Shamir in Adleman.

Prednost EC kriptosistema pred RSA kriptosistemom je v tem, da omogoča višjo stopnjo varnosti pri enaki dolžini ključa. Trenutno najučinkovitejši algoritem, ki omogoča razbitje RSA šifre, ima podeksponentno časovno zahtevnost, medtem ko ima najučinkovitejši algoritem za razbitje EC šifre eksponentno časovno zahtevnost (glej [5]). Posledično je šifriranje in odšifriranje z uporabo EC kriptosistema učinkovitejše, saj lahko uporabljamo ključe krajših dolžin kot pri RSA in obdržimo enako stopnjo varnosti.

Kljub temu, da izum EC kriptosistema (Miller, Koblitz) sega v leto 1985, pa se je ta v praksi v večjem obsegu začel uporabljati šele pred kratkim. To lahko vidimo tudi v naraščajoči podpori programske opreme EC kriptosistemu. Sun Microsystems je podporo EC dodal v javanskem razvojnem kompletu SDK 6 in spletnem strežniku Sun Java System Web Server 7.0, Microsoft v razvojnem ogrodju .NET Framework 3.5 ter v svojih najnovejših operacijskih sistemih Microsoft Windows Vista ter Microsoft Windows Server 8. Prav tako je bila podpora EC dodana v več odprtokodnih kriptografskih projektih, med katerimi najbolj izstopa OpenSSL, na katerem slonijo kriptografski komunikacijski protokoli spletnega strežnika Apache ter brskalnik Mozilla Firefox 3. V prihodnosti lahko pričakujemo, da bo EC kriptosistem izpodrinil RSA kriptosistem.

V skladu s trendi v kriptografiji so se tudi na Fakulteti za računalništvo in informatiko v Ljubljani (FRI), natančneje v Laboratoriju za kriptografijo in računalniško varnost (LKRV) pod

vodstvom profesorja dr. Aleksandra Jurišića odločili, da bo njihova certifikatna agencija z imenom @friCA v prihodnosti izdajala digitalna potrdila za Fakulteto za računalništvo in informatiko ter za Fakulteto za elektrotehniko izključno z uporabo kriptografije eliptičnih krivulj. Kljub temu, da programska knjižnica OpenSSL na kateri temelji Apachejeva podpora protokolu HTTPS, že podpira EC kriptosistem, pa v tem trenutku oviro predstavljajo predvsem spletni brskalniki, saj nobeden izmed najpogosteje uporabljanih spletnih brskalnikov, z izjemo Mozille Firefox 3, te podpore ne vsebuje (podporo EC je mogoča tudi v Mozilla Firefox 2, vendar le v Linuxu, kjer je izvorno kodo brskalnika potrebno prevesti s posebnimi stikali). Posledično @friCA trenutno izdaja RSA digitalna potrdila, vendar pa lahko pričakujemo, da bo tudi ta ovira v bližnji prihodnosti premagana.

Ker je temelj vsakega uspešno izpeljanega projekta dober načrt, so se v LKRV odločili izdelati analizo odzivnosti sistema, ki bo za varno izmenjavo sporočil med uporabniki in strežnikom ter za avtentikacijo uporabnikov uporabljal digitalna potrdila z uporabo RSA in EC kriptosistema. Analizirati je namreč potrebno, kako se spreminja odzivni čas sistema pri različnih obremenitvah strežnika z uporabo obeh kriptosistemov z ozirom na gostoto zahtev in število uporabnikov. Glede na teoretične analize je pričakovati, da se bo za hitrejšega izkazal EC kriptosistem. Najbolj znana praktična primerjava je bila izvedena pod okriljem podjetja Sun Microsystems (glej [3]), kjer so ugotovili, da se pri nižjih stopnjah varnosti oziroma krajših dolžinah ključev (160 bitov pri EC oziroma 1024 pri RSA) odzivnost spletnega strežnika Apache z uporabo kriptosistema EC izboljša za 10%-30%, pri višjih stopnjah varnosti (224 bitov pri EC oziroma 2048 pri RSA) pa se prednost pred RSA kriptosistemom poveča kar za 110%-280%. Prednosti EC pred RSA pa se ne kažejo samo v večji hitrosti izvajanja, temveč tudi v manjši porabi energije, delovnega spomina in pasovne širine, kar se kot velika prednost pokaže predvsem pri uporabi manjših mobilnih naprav, ki imajo te vire zelo omejene ter pri delovanju strežnikov, kjer se šifrirne operacije izvršujejo zelo pogosto.

Cilj diplomskega dela je bil izdelati aplikacijo, ki bi znala opraviti obremenitveni test spletnega strežnika. Aplikacija bi na podlagi parametrov izvajala zahteve na spletni strežnik ter merila odzivnost strežnika pri uporabi digitalnih potrdil različnih dolžin ključev obeh kriptosistemov.

Za izvajanje obremenitvenega testa ima LKRV na voljo dve računalniški učilnici, v kateri se nahaja več osebnih računalnikov, ki jih je moč uporabiti za izvajanje obremenitev strežnika. Ker je spletni strežnik Apache visoko optimiziran in prenese velike obremenitve, je potrebnih več računalnikov, ki simultano izvajajo obremenitveni test. Ob večjih obremenitvah postanejo spremembe v hitrosti delovanja bolj izrazite (glej [3]), kar omogoča natančnejšo analizo rezultatov. Posledično mora biti aplikacija za izvajanje obremenitvenega testa porazdeljena; izvajati se mora na več računalnikih naenkrat ter imeti centraliziran nadzor, ki usklajuje izvajanje testne aplikacije na vseh računalnikih.

Obremenitveni test mora biti nastavljiv. Vsebovati mora naslednje parametre: trajanje testa, število zahtev, ki naj se izvedejo v tem času, delež veljavnih in delež neveljavnih zahtev, verjetnostno porazdelitev izvajanja zahtev ter seznam digitalnih potrdil, ki se uporabljajo za avtentikacijo uporabnika in izmenjavo tajnih ključev za simetrično šifriranje povezave. Glede na to ali se med testom uporabljajo RSA ali EC digitalna potrdila, se testira obnašanje strežnika pri uporabi enega ali drugega kriptosistema. Obremenitveni test se mora začeti istočasno na vseh testnih računalnikih, po koncu izvajanja pa mora vsak testni računalnik vrniti rezultate testa, kjer je za vsako zahtevo navedeno, kdaj se je začela, kdaj je bil prejet odgovor, naslov zahteve uspešnost odgovora na zahteve ter digitalno potrdilo, ki je bilo uporabljeno. Na podlagi prejetih rezultatov vseh testnih računalnikov se izdela statistična analiza odzivnosti strežnika, ki služi kot izhodišče za primerjavo hitrosti uporabe obeh kriptosistemov.

Testirati ter primerjati je potrebno RSA kriptosistem z uporabo dolžin ključev 1024, 2048 in 3072 bitov ter EC kriptosistem z dolžinami ključev 160, 224 in 256 bitov, saj so te dolžine ključev paroma primerljive glede na stopnjo varnosti (glej poglavje 5).

Organizacija diplomske naloge

V naslednjem poglavju so predstavljeni temeljni koncepti kriptografije, kot so šifriranje in odšifriranje, zaupnost in celovitost podatkov, simetrični in asimetrični kriptosistemi, zasebni in javni ključi ter drugo. Poglavje RSA kriptosistem prikaže, kako v RSA kriptosistemu poteka generiranje ključev, šifriranje in odšifriranje, dogovor o ključu in digitalno podpisovanje. Poglavje EC kriptosistem govori o tem, kako so prej omenjene operacije realizirane v EC

kriptosistemu. Poglavje 6 prikaže razlike med lastnostmi obeh kriptosistemov ter njuni uporabi v komunikacijskih protokolih. Poglavje Spletni strežnik opisuje računalnik, na katerem se izvaja Apache spletni strežnik, ki je bil uporabljen v praktičnem preizkusu, poglavje Aplikacija pa aplikacijo, ki je bila izdelana za analizo odzivnosti obeh kriptosistemov. Poglavje Analiza odzivnosti predstavi teste, ki so bili izvedeni in analizira rezultate, pridobljene pri praktičnem preizkusu.

Poglavje 2

Osnove kriptografije

Kriptografija je v osnovi matematično-računalniška veda, ki se ukvarja predvsem z zaupnostjo podatkov, njihovo integriteto in avtentikacijo izvora podatkov.

Kriptografski sistem ali krajše kriptosistem je množica kriptografskih metod, ki zagotavljajo vsaj enega izmed omenjenih ciljev kriptografije.

Procesu pretvarjanja originalnega sporočila v šifrirano sporočilo ali šifro pravimo šifriranje, obratnemu procesu, torej pretvorbi šifre nazaj v originalno sporočilo, pa odšifriranje. Uporaba šifriranja ima poleg pozitivnega učinka na zaupnost sporočil negativen učinek na hitrost komuniciranja, saj zahteva izvajanje dveh dodatnih operacij med komunikacijo: šifriranja na eni in odšifriranja na drugi strani.

Varnost šifre temelji na predpostavki, da je metoda pretvorbe originalnega sporočila v šifrirano obliko in nazaj ob poznavanju posebnega parametra, ki mu pravimo ključ in ni nič drugega kot zaporedje naključno izbranih bitov, hitra in enostavna, v nasprotnem primeru pa je zaradi nepoznavanja ključa ta proces kljub poznavanju šifriranega sporočila časovno tako potraten, da tudi najbolj učinkovita poznana metoda za razbijanje šifre potrebuje več časa, kot ga ima napadalec na voljo. Vkolikor to ne velja, se kriptografska metoda ne smatra za varno, njeni uporabi pa se je potrebno izogibati.

Kompromis med stopnjo varnosti in učinkovitostjo kriptografske metode predstavlja dolžina ključa, ki mora biti dovolj velika, da ustrezno oteži razbijanje šifre. Istočasno pa dolžina ključa ne sme biti prevelika, saj želimo da je šifriranje oziroma odšifriranje čim hitrejše. Prav tako je zaželena čim manjša dolžina ključa pri majhnih napravah (pametne kartice, mobilni telefoni), ki imajo količino spomina zelo omejeno. Z naraščanjem procesorske zmogljivosti računalnikov so za vzdrževanje tega razmerja potrebne večje dolžine ključev.

Zaupnost podatkov

Kadar si entiteti izmenjujeta občutljive podatke, za katere želita, da bi vsebina sporočila ostala znana samo njima in nikomur drugemu, je potrebno zagotoviti tajnost sporočil. En tak način je uporaba varnega komunikacijskega kanala, ki je fizično zaščiten pred neželenim prisluškovanjem, vendar pa je ta le redko na voljo. Druga možnost je uporaba nezaščitenega komunikacijskega kanala ter šifriranje sporočil, kar preseli varnost iz nivoja komunikacijskega kanala na nivo sporočila. Takšen pristop omogoča varno komunikacijo brez posega v strukturo komunikacijskega kanala, potrebno je le dodati mehanizem za šifriranje na strani oddajnika, torej ob vhodu v komunikacijski kanal ter mehanizem za odšifriranje pri sprejemniku ob izhodu iz komunikacijskega kanala.

Celovitost podatkov

Celovitost podatkov pomeni, da sporočilo ni bilo spremenjeno vse naprej od trenutka, ko je zapustilo pošiljatelja, četudi se sporočilo prenaša po komunikacijskem kanalu v originalni obliki oziroma nešifrirano. V primeru, da celovitost podatkov ni zagotovljena, lahko napadalec sporočilo med prenosom po kanalu prestreže in ga spremeni brez vednosti pošiljatelja in naslovnika (tako imenovani *man in the middle attack*). Takrat se mora naslovnik sam odločiti, ali bo vsebini sporočila verjel, saj nima nobenega zagotovila o njeni verodostojnosti.

Kriptografija za zagotavljanje integritete podatkov predlaga uporabo digitalnih izvlečkov oziroma digitalnih prstnih odtisov sporočil; metodo, ki za vsako sporočilo z deterministčnim algoritmom izračuna njegov izvleček. Izvleček je pravzaprav povzetek sporočila, ki je enoličen vsakemu sporočilu ne glede na njegovo dolžino in je konstantne dolžine (npr. 128 bitov). Ob pošiljanju se originalnemu sporočilu doda tudi njegov šifriran izvleček, prejemnik pa ponovno izračuna izvleček prejetega sporočila in če se izračunani izvleček in odšifriran prejeti izvleček ujemata, je celovitost vsebine sporočila zagotovljena, v nasprotnem primeru pa lahko prejemnik sklepa, da je bilo sporočilo med prenosom spremenjeno.

Zaželeno je, da metoda za generiranje izvlečkov sporočil omogoča izdelavo izvlečka iz sporočila enostavna in hitra, obratno pa je v doglednem času nemogoče najti pravo sporočilo

z enakim izvlečkom. Metodi za računanje digitalnih izvlečkov pravimo tudi zgoščevalna funkcija.

Primeri zgoščevalnih funkcij so:

- MD5 (Message-Digest algorithm 5, dolžina izvlečkov je 128 bitov)
- SHA-1 (Secure Hash Algorithm, dolžina izvlečkov je 160 bitov)
- SHA-256(Secure Hash Algorithm, dolžina izvlečkov je 256 bitov)

Avtentikacija pošiljatelja

Avtentikacija pošiljatelja skuša zagotavljati identifikacijo pošiljatelja sporočila in s tem onemogočati napad, kjer se napadalec izdaja za avtorja sporočila.

V primeru, da si isti ključ deli več entitet, je na podlagi izvlečka sporočila nemogoče določiti avtorja sporočila, z gotovostjo lahko sklepamo le, da gre za eno izmed teh entitet. Problem je rešljiv tako, da vsaka entiteta uporablja unikaten ključ, katerega izvor je mogoče ugotoviti. Ta pristop omogoča uporaba avtoritete, ki je zadolžena za izdajanje ključev. Tej avtoriteti pravimo certifikatna agencija. Ko entiteta zaprosi za ključ, ji certifikatna agencija na podlagi veljavnega dokumenta, s katerim entiteta dokazuje svojo identiteto, izda javni digitalni ključ. Pravzaprav entiteta prejme par ključev: zasebnega, ki ga mora hraniti na varnem ter javnega, ki mora biti dostopen vsem, ki bi radi komunicirali s to entiteto. Javni ključ lahko entiteta generira tudi sama. Javni ključ je digitalno podpisan s strani certifikatne agencije in vsebuje tudi podatke o imetniku ključa, obdobju veljavnosti in agenciji, ki ga je izdala. Takemu javnemu ključu pravimo digitalno potrdilo. Entiteta sporočilo digitalno podpiše z zasebnim ključem, zato je njena dolžnost, da zasebni ključ hrani na varnem, prejemnik pa preveri avtentikacijo pošiljatelja z njegovim digitalnim potrdilom. Ta pristop je asimetrični, obstaja tudi simetrični, ki je sicer hiter, a zaradi velikega števila ključev neučinkovit.

Enkripcijska shema

Enkripcijska shema je množica metod, ki omogočajo šifriranje in odšifriranje. Za določitev enkripcijske sheme potrebujemo naslednje množice:

- množico A (abeceda sporočila, npr.: slovenska abeceda $\{a, b, c, \dots, \text{ž}\}$, binarna abeceda $\{0, 1\}$, itd),
- množico M (množica vseh sporočil, ki jih lahko sestavimo z abecedo A, torej velja za vsak $m \in M = \{a_1, \dots, a_n\}$; $a_i \in A$),
- množico C (množica vseh šifriranih sporočil, abeceda množice C je lahko enaka A, vendar ni nujno) ter
- množico K (množica vseh ključev).

ter dve funkciji:

- $E_e: M \to C$ (šifrirna funkcija, kjer je $e \in K$ šifrirni ključ, M množica originalnih sporočil in C množica šifriranih sporočil) in
- $D_d: C \to M$ (odšifrirna funkcija, kjer je $d \in K$ odšifrirni ključ).

Če želimo, da je proces šifriranja obrnljiv, torej da lahko šifrirano sporočilo pretvorimo nazaj v njegovo originalno obliko oziroma ga odšifriramo, morata biti obe funkciji injektivni (vsako sporočilo se lahko pretvori v natanko eno šifrirano sporočilo ter vsako šifrirano sporočilo se lahko pretvori nazaj v natanko eno sporočilo, ki je enako originalnemu). Iz povedanega sledi, da mora veljati $D_d = E_e^{-1}$ oziroma $D_d\left(E_e\ (m)\right) = m$ za vsak $m \in M$ (glej [1]).

Zaupnost sporočil dosežemo na dva načina: tako, da izberemo tajno šifrirno in odšifrirno funkcijo, ali pa da izberemo znani parametrizirani funkciji in ključa *e* in *d* (parametra funkcij E in D, ki morata ostati tajna). Slabost prvega pristopa je ta, da moramo vedno, kadar želimo spremeniti način šifriranja in odšifriranja, spremeniti tudi enkripcijsko shemo, medtem ko ob uporabi drugega pristopa preprosto zamenjamo ključa, sami funkciji pa obdržimo. Napadalec kljub temu, da pozna funkciji E in D brez poznavanja ključev ne bo mogel odšifrirati šifriranih podatkov. Kadar velja, da sta ključa e in d enaka, ali pa je mogoče iz enega enostavno izračunati drugega, govorimo o simetrični kriptografiji, v nasprotnem primeru, ko sta ključa e in d različna, pa o asimetrični kriptografiji.

Potreben, ne pa tudi zadosten pogoj za varnost kriptosistema je, da so množice M, C in K čim večje, saj na ta način dosežemo, da je preiskovanje vseh možnih kombinacij (m_i, c_j) ob nepoznavanju ključev (e, d) časovno zelo potratno in posledično neučinkovito (pri hitrosti

delovanja današnjih računalnikov, velja, da temu ustreza množica z vsaj 2^{80} elementi), po drugi strani pa morata biti funkciji E in D čim hitrejši, zato da čim manj vplivata na zakasnitve pri prenosu sporočil.

Poleg šifriranja in odšifriranja mora enkripcijska shema vsebovati tudi algoritem za generiranje ključev, ki ni nič drugega kot generator psevdo naključnih števil.

Simetrična kriptografija

V primeru, da sta šifrirni in odšifrirni ključ enaka, oziroma da je mogoče enega izmed ključev izpeljati iz drugega, govorimo o simetrični kriptografiji, temu ključu pa pravimo *tajni ključ*. Vsak, ki pozna tajni ključ, lahko sporočilo šifrira in odšifrira. Prednost simetrične kriptografije je v majhni računski zahtevnosti in posledično visoki hitrosti šifriranja in odšifriranja, njeno največjo pomanjkljivost pa predstavlja dogovor o ključu, saj se morata stranki pred začetkom komuniciranja dogovoriti o ključu preko varnega kanala, ki ni vedno na voljo. Pomanjkljivost predstavljata tudi pogosta menjava tajnih ključev, po navadi kar za vsako sejo ter dejstvo, da je v komunikacijski mreži za vsak komunikacijski kanal potreben svoj ključ. Toliko ključev je namreč težko hraniti na primer pametnih karticah.

Najbolj razširjena simetrična kriptosistema sta:

- DES (Data Encryption Standard), ki je bil najpogosteje uporabljan način šifriranja po drugi svetovni vojni in je uporabljal 56-bitne dolžine ključa.
- AES (American Encription Standard), ki je leta 2000 nadomestil DES.

Kriptosistemi z uporabo javnih ključev

Asimetrična kriptografija, znana tudi kot kriptografija javnih ključev, je oblika kriptografije, kjer ima vsak uporabnik par različnih ključev: šifrirni ključ, ki ga v asimetrični kriptografiji imenujemo *javni ključ* ter odšifrirni oziroma *zasebni ključ*. Kar se šifrira z javnim ključem, se lahko odšifrira le z zasebnim ključem in obratno. Veljati mora, da je zasebni ključ neodvisen od javnega, torej iz javnega ključa ni mogoče pridobiti nobene informacije zasebnem ključu. To je v praksi težko doseči, zato se zadovoljimo s tem, da je to računsko prezahtevno. Javni ključ mora biti na voljo vsem, ki želijo komunicirati z njegovim lastnikom, medtem ko mora

biti zasebni ključ na varnem, dostop do njega pa sme imeti le njegov lastnik. Vedno kadar nekdo želi lastniku ključev poslati šifrirano sporočilo, z njegovim javnim ključem izvede šifriranje sporočila ter ga pošlje lastniku, ta pa ga odšifrira s svojim zasebnim ključem.

Prednost asimetrične kriptografije pred simetrično je predvsem v enostavni izmenjavi ključev ter manjši potrebi po menjavi ključev (v primeru, da izberemo dovolj dolgo dolžino ključev, lahko isti par ključev lahko obdržimo tudi po nekaj let), njena slabost pa v večji računski zahtevnosti.

V praksi se uporablja kombinacija asimetrične in simetrične kriptografije: asimetrična se uporabi za izmenjavo tajnih ključev med obema komunicirajočima strankama, za nadaljnjo komunikacijo pa se uporablja simetrična kriptografija. Ta pristop ima najboljše lastnosti obeh svetov: enostavna in varna izmenjava ključev ter hitro šifriranje in dešifriranje.

Najbolj razširjeni asimetrični kriptosistemi so:

- RSA (Rivest, Shamir in Adleman),
- ElGamal nad $\mathbb{Z}_p$ oziroma
- ECC (Elliptic Curve Cryptography).

Poglavje 3

RSA kriptosistem

RSA kriptosistem je danes najbolj razširjen asimetrični kriptosistem. Razvit je bil leta 1977 na Inštitutu za tehnologijo na MIT (Massachusetts Institute of Technology), svoje ime pa je dobil po njegovih stvariteljih: Rivest, Shamir in Adleman.

RSA kriptosistem poleg asimetričnega kriptografskega protokola RSAES (RSA Encryption Scheme) za generiranje digitalnih podpisov uporablja RSA Signing Scheme, za izmenjavo tajnega ključa pa DH (Diffie-Hellman) protokol.

Določitev javnega in zasebnega ključa

Par ključev se določi po naslednjem algoritmu (glej [1]):

1. Naključno generiramo dve veliki praštevili p in q, ki sta približno enake dolžine ter sta si različni.
2. Izračunamo $n = pq$.
3. Izračunamo $\varphi(n) = (p-1)(q-1)$.
4. Izberemo naključno naravno število e, kjer je $1 < e < \varphi$ in za katerega velja, da sta si števili e in φ tuji, oziroma da je njun največji skupni delitelj enak 1.
5. Z uporabo Evklidovega algoritma (glej [1]) izračunamo od e različno naravno število d, kjer je $1 < d < \varphi$ tako da velja $ed \bmod \varphi = 1$.
6. Javni ključ predstavlja par števil (n, e), zasebni ključ pa števila (d, p, q).

Pri izbiri praštevil p in q je potrebno biti pazljiv, saj je v določenih primerih lahko šifriranje podvrženo nekaterim napadom (glej [1]).

Šifriranje in odšifriranje

Varnost enkripcijske sheme RSA temelji na računski zahtevnosti faktorizacije velikih naravnih števil, kjer je za dve veliki praštevili p in q izračun števila $n = pq$ enostavna operacija z majhno računsko zahtevnostjo, obraten postopek, to je ugotovitev pravilne kombinacije praštevil p in q, če poznamo le število n, pa je težak matematični problem (glej [1]).

Šifriranje z uporabo kriptosistema RSA (glej [1]):

1. Pridobimo javni ključ (n, e).
2. Originalno sporočilo spremenimo v naravno število m na intervalu $[0, n-1]$. V primeru, da je originalno sporočilo daljše od $n-1$, ga razdelimo v bloke ter kriptiramo vsak blok posebej.
3. Izračunamo šifrirano sporočilo $c = m^e \bmod n$.
4. Sporočilo pošljemo lastniku zasebnega ključa.

Odšifriranje z uporabo kriptosistema RSA (glej [1]):

1. Za odšifriranje uporabimo zasebni ključ (d, p, q) in iz šifriranega sporočila pridobimo originalno sporočilo z enačbo $m = c^d \bmod n$.

Dogovor o ključu

Pri kriptosistemu RSA poteka dogovor o tajnem ključu za simetrično šifriranje sporočil med dvema komunicirajočima strankama po Diffie-Hellmanovem protokolu (DH protokol), katerega varnost temelji na problemu diskretnega logaritma. Zanj je značilno, da je izračun tajnega ključa $k = g^{ba} = g^{ab}$, kjer je g generator ciklične grupe, časovno zelo potraten ob poznavanju dveh javnih vrednosti $g^a \bmod p$ in $g^b \bmod p$ in nepoznavanju parametrov a in b, pod pogojem, da je p zadosti veliko število (glej Poglavje 2). Dogovor o ključu poteka po naslednjemu algoritmu (glej [10]):

1. Stranki določita parameter p in generator g, kjer velja: p je praštevilo, $p > g$ za vsak $n \in [1, p-g]$ obstaja nek $k \epsilon \mathbb{N}_0$ za katerega velja $n = g^k \bmod p$.
2. Stranka A določi naključno celo število a, ki mora ostati skrito.
3. Stranka B določi naključno celo število b, ki mora ostati skrito.

4. Stranka A pošlje stranki B g^a mod p.
5. Stranka B pošlje stranki A g^b mod p.
6. Stranka A izračuna $g^{ba} = (g^a)^b$ mod p.
7. Stranka B izračuna $g^{ab} = (g^b)^a$ mod p.
8. Ker velja $g^{ba} = g^{ab} = k$, sta si stranki uspešno izmenjali tajni ključ.

Digitalno podpisovanje

Digitalni podpis se uporablja v digitalnih dokumentih, njegov namen pa je podoben tistemu pri navadnem podpisu na papirju in to je avtentikacija avtorja dokumenta. Da je avtentikacija entitete mogoča, mora veljati, da digitalnega podpisa ni mogoče ponarediti. Poleg avtentikacije pa digitalni podpis zagotavlja tudi celovitost sporočila. Digitalni podpis mora v kriptografiji javnih ključev vsebovati dva učinkovita algoritma: algoritem za digitalno podpisovanje in algoritem za preverjanje veljavnosti digitalnega podpisa.

Postopek digitalnega podpisovanja v RSA kriptosistemu poteka tako, da avtor sporočilo digitalno podpiše z veljavnim zasebnim ključem. Ker je sporočilo pogosto predolgo in bi bilo digitalno podpisovanje preveč časovno potratno, se z uporabo zgoščevalne funkcije izračuna njegov digitalni izvleček, ki mu pravimo tudi prstni odtis. Ta se digitalno podpiše, prejemnik pa preveri pristnost digitalnega podpisa z avtorjevim javnim ključem. Če prejemnik ugotovi, da je podpis veljaven, običajno lahko zaupa, da je avtor sporočila res njegov pošiljatelj, v nasprotnem primeru pa to ne velja.

RSA kriptosistem za digitalno podpisovanje uporablja lastno podpisovalno shemo (RSA Signature Scheme), ki zamenja vlogi šifriranja in odšifriranja, torej ne potrebuje novih algoritmov. Šifriramo oziroma podpišemo z zasebnim ključem, odšifriramo oziroma preverimo podpis pa z javnim ključem. Javni in zasebni ključ sta (n, e) in (d, p, q) (glej poglavje 3, »Določitev javnega in zasebnega ključa«).

Generiranje digitalnega podpisa (glej [1]):

1. Izračunamo $\widetilde{m} = h(m)$, kjer je m originalno sporočilo, $h: M \to Z_n$ zgoščevalna funkcija (npr. SHA-1) in $\widetilde{m}$ naravno število na intervalu $[0, n-1]$.
2. Izračunamo $s = \widetilde{m}^d$ mod n. Digitalni podpis sporočila m je vrednost s.

3. Prejemniku pošljemo originalno sporočilo, zraven pa pripnemo njegov digitalni podpis.

Preverjanje veljavnosti digitalnega podpisa (glej [1]):

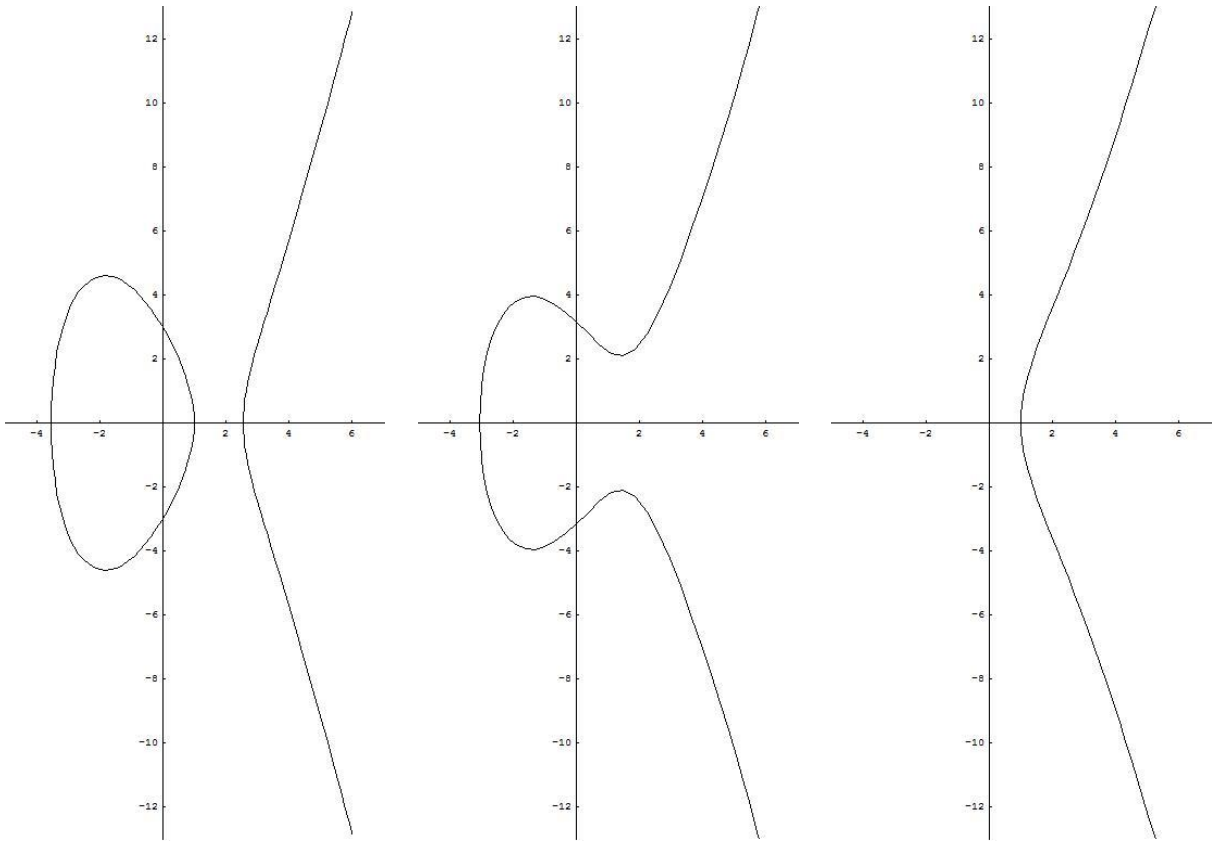
1. Pridobimo javni ključ pošiljatelja (n, e).
2. Iz prejetega digitalnega podpisa izračunamo izvleček sporočila $\tilde{m}' = s^e \bmod n$.
3. Izračunamo $\tilde{m} = h(m)$. Če je izračunani izvleček sporočila enak izvlečku, ki smo ga pridobili iz digitalnega podpisa ($\tilde{m} = \tilde{m}'$), potem je podpis veljaven, v nasprotnem primeru pa to ne drži.

Poglavje 4

EC kriptosistem

Ameriška matematika Miller in Koblitz sta prišla na idejo, da bi bilo mogoče realizirati kriptosistem z uporabo eliptičnih krivulj in tako je leta 1985 nastal EC kriptosistem. Varnost EC temelji na problemu diskretnega logaritma eliptičnih krivulj (glej [7]), kjer je ob poznavanju dveh točk na eliptični krivulji P in Q, tako da velja $kP = Q$, izračun vrednosti skalarnega parametra k časovno izredno potraten, pod pogojem da je k dovolj veliko število.

Eliptična krivulja nad obsegom realnih števil $\mathbb{R}$ je določena z enačbo $y^2 = x^3 + ax + b$, kjer mora veljati, da je njena diskriminanta različna od 0; to je $4a^3 - 27b^2 \neq 0$. V tem primeru so vse ničle polinoma na desni med seboj različne (glej [8]). Parametra a in b določata krivuljo. Za eliptične krivulje je značilno, da so simetrične glede na abscisno os, torej za vsako vrednost x dobimo dve vrednosti y, ki se razlikujeta le v predznaku (glej [8]).



$y^2 = x^3 - 10x + 9$ $\quad$ $\boldsymbol{y^2 = x^3 - 6x + 10}$ $\quad$ $y^2 = x^3 + 6x - 7$

Slika 1: Primeri eliptičnih krivulj nad $\mathbb{R}$ (glede na to, koliko realnih ničel ima polinom na desni strani; tri različne, ena sama, ena trojna)

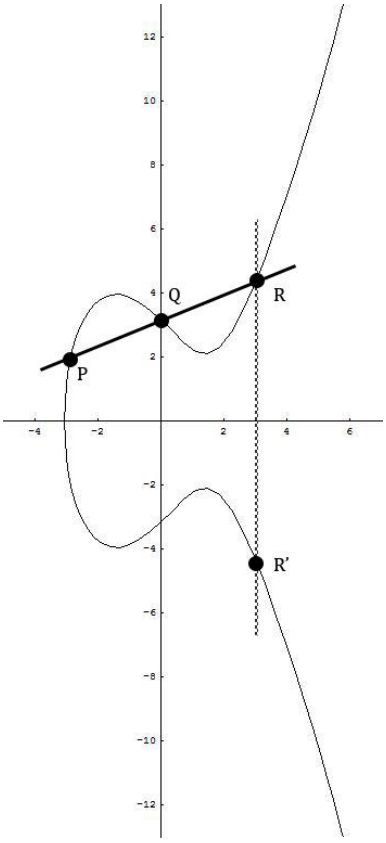
Primer eliptičnih krivulj nad $\mathbb{R}$ si je mogoče ogledati na spletnem naslovu http://lkrv.fri.uni-lj.si/~ajurisic/tec3/gradiva/ECClassroom/Files/EC_contents.htm, kjer se nahaja interaktivna aplikacija za njihovo vizualizacijo.

Nad točkami eliptične krivulje, katerim pridružimo še točko v neskončnosti O, je mogoče definirati binarno operacijo, kjer je rezultat operacije med dvema točkama na krivulji tudi točka te krivulje. Tej operaciji pravimo seštevanje točk, kjer velja: $P + Q = R'$ in je komutativna (Abelova) grupa (glej [8]), saj zanjo velja, da je:

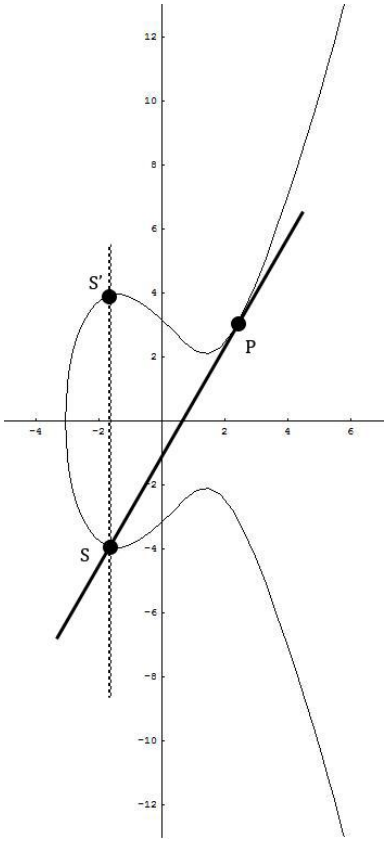
- Komutativna: $P + Q = Q + P$, torej vrstni red operandov ni pomemben.
- Asociativna: $(A + B) + C = A + (B + C)$, torej lahko operacijo izvajamo v poljubnem vrstnem redu.
- Ima nevtralni element O: $P + O = P$.

- Vsak element P ima nasprotni element P': $P + (P') = O$, saj je eliptična krivulja simetrična glede na abscisno os, torej je nasprotni element P' preko abscise prezrcaljena točka P.

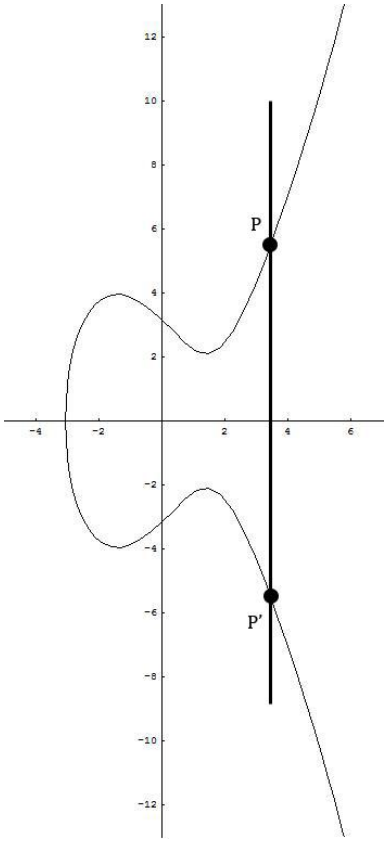
Seštevanje različnih točk na eliptični krivulji je mogoče grafično ponazoriti tako, da skozi dve točki, ki jih seštevamo, potegnemo premico, rezultat pa je nasprotna točka (glede na x os) novo dobljenega presečišča premice in eliptične krivulje. Seštevanje istih točk $P + P = S'$ grafično prikažemo tako, da v točki P potegnemo njeno tangento. Kjer ta seka eliptično krivuljo, se nahaja točka S, S' pa je njena nasprotna točka. Če seštevamo nasprotni si točki $P + P' = O$, je rezultat točka v neskončnosti O, ker je v temu primeru premica skozi točki P in P' navpična in krivulje ne seka v tretji točki.



Slika 2: Seštevanje točk $P + Q = R'$



Slika 3: Podvajanje točk $2P = P + P = S'$



Slika 4: Seštevanje nasprotnih točk $P + P' = O$

Seštevanje istih točk nad eliptično krivuljo lahko izrazimo tudi kot množenje točk s skalarjem, kjer množenje točke P na krivulji s skalarjem vrne novo točko Q, ki se nahaja na isti krivulji ($kP = Q$). Množenje točk je mogoče izvajati na dva načina:

- S seštevanjem dveh točk, tako da dobimo tretjo točko na krivulji: $L = J + K$.
- S podvajanjem točk, kjer seštejemo dve enaki točki: $L = 2J$.

Tako je mogoče $Q = 10P$ izračunati kot $Q = P + P + P + P + P + P + P + P + P + P$ ali kot $Q = 2(2(2P) + P)$. V praksi se uporablja slednji način (podvoji in seštej), saj je zaradi logaritemske časovne odvisnosti Hornerjevega algoritma hitrejši od prvega, ki ima eksponentno časovno odvisnost glede na dolžino zapisa. Interaktivno aplikacijo za vizualizacijo računanja s točkami nad eliptično krivuljo je mogoče preizkusiti na spletnem naslovu http://www.certicom.ca/ ecc_tutorial/ecc_javaCurve.html.

Ker je izvajanje operacij nad eliptičnimi krivuljami na realnih številih časovno potratno in zaradi zaokrožitvenih napak tudi nezanesljivo, se v EC kriptosistemu za računanje uporabljata dva končna obsega: celoštevilski obseg F_p in obseg binarnih vektorjev F_{2^m}. Ne glede na to, katero izberemo, mora biti njuna moč dovolj velika, s čimer je zagotovljen zadosti obsežen nabor točk.

Domenske parametre EC kriptosistema predstavlja 6-terica (F_{p^m}, a, b, P, n, h) (glej [9]):

- V primeru $m = 1$, je obseg $F_{p^m} = \mathbb{Z}_p$, v primeru $p = 2$ pa $F_{p^m} = GF(2^m)$.
- a in b sta koeficienta krivulje in definirata uporabljeno krivuljo.
- P je generator ciklične grupe nad to eliptično krivuljo (vsak element grupe lahko zapišemo kot večkratnik točke P).
- red te ciklične podgrupe je n ($nP = O$), ki mora biti neko veliko praštevilo ter določa velikost ciklične podgrupe.
- $h = |E(F_{p^m})|/n$. Želimo, da je h majhno število (običajno 2 ali 4).

EC kriptosistem ima poleg asimetričnega kriptografskega protokola tudi lasten protokol za generiranje digitalnih podpisov Elliptic Curve Digital Signature Algorithm (ECDSA) ter lasten protokol za izmenjavo tajnih ključev Elliptic Curve Diffie-Hellman (ECDH).

Določitev javnega in zasebnega ključa

Generiranje para ključev (glej [9]):

1. Izberemo naključno naravno število d na intervalu $[1, n-1]$.

2. Izračunamo $Q = dP$.
3. Število d je zasebni ključ, točka Q pa javni ključ.

Šifriranje in odšifriranje

Šifriranje z uporabo EC (glej [9]):

1. Pošiljatelj izbere naključno število k.
2. Pošiljatelj izračuna šifrirano sporočilo po enačbi $(kP; (kQ)_x + m)$, kjer je Q javni ključ prejemnika, m pa originalno sporočilo, . $(kQ)_x$ pa x koordinata izračunane točke na eliptični krivulji.

Odšifriranje z uporabo ECC (glej [9]):

1. Prejemnik pridobi originalno sporočilo iz šifriranega sporočila z izračunom: $m + (kQ)_x - (dkP)_x = m + (kdP)_x - (dkP)_x = m$, kjer je d zasebni ključ prejemnika.

Dogovor o ključu

EC kriptosistem za izmenjavo tajnega ključa uporablja ECDH protokol, ki je za eliptične krivulje prirejen Diffie-Hellman protokol. Tako kot varnost DH protokola temelji na problemu diskretnega logaritma, tudi varnost ECDH protokola temelji na problemu diskretnega logaritma eliptičnih krivulj. Stranki v protokolu si preko komunikacijskega kanala izmenjata domenske parametre $(F_{p^m}, a, b\ P, n, h)$ in svoja javna ključa, kar so vse podatki javnega značaja in morebitnemu napadalcu ne omogočajo rešitve problema diskretnega logaritma. Dogovor o ključu poteka po naslednjem algoritmu, kjer velja, da je (d_a, Q_a) zasebni in javni ključ stranke A ter (d_b, Q_b) par ključev stranke B (glej [9]):

1. Stranka A izračuna $K = (K_x, K_y) = d_a Q_b = d_a d_b P$.
2. Stranka B izračuna $L = (L_x, L_y) = d_b Q_a = d_b d_a P$.
3. Ker velja $d_a d_b P = d_b d_a P = K = L = (K_x, K_y) = (L_x, L_y)$, sta si stranki uspešno izmenjali tajni ključ $k = K_x = L_x$.

Digitalno podpisovanje

EC kriptosistem za digitalno podpisovanje uporablja ECDSA, ki je za eliptične krivulje prirejen DSA protokol. Javni ključ pošiljatelja je Q, njegov zasebni ključ pa d.

Generiranje digitalnega podpisa (glej [9]):

1. Izberemo naključno število k na intervalu $[1, n-1]$.
2. Izračunamo $kP = (x_1, y_1)$.
3. Izračunamo $r = x_1 \bmod n$.
4. Če je $r = 1$, potem se vrnemo na točko 1.
5. Izračunamo $k^{-1} \bmod n$ z uporabo razširjenega Evklidovega algoritma.
6. Izračunamo $\tilde{m} = h(m)$, kjer je h zgoščevalna funkcija.
7. Izračunamo $s = k^{-1}(\tilde{m} + dr) \bmod n$.
8. Če velja $s = 0$, potem se vrnemo na točko 1. Digitalni podpis predstavlja par vrednosti (r, s)

Preverjanje veljavnosti digitalnega podpisa (r, s) z uporabo javnih vrednosti (glej [9]):

1. Preverimo, ali sta s in r naravni števili na intervalu $[1, n-1]$.
2. Izračunamo $\tilde{m} = h(m)$.
3. Izračunamo $\omega = s^{-1} \bmod n$.
4. Izračunamo $u_1 = \tilde{m}\omega \bmod n$.
5. Izračunamo $u_2 = r\omega \bmod n$.
6. Izračunamo $u_1 P + u_2 Q = (x_1, y_1)$.
7. Izračunamo $v = x_1 \bmod n$.
8. Če velja $v = r$, potem je podpis veljaven, saj je izračunani izvleček enak prstnemu odtisu sporočila, v nasprotnem primeru pa to ne velja.

Poglavje 5

Primerjava EC in RSA kriptosistemov

Dva kriptosistema imata enako stopnjo varnosti pri neki dolžini ključa, če velja, da najučinkovitejša algoritma za razbijanje ene in druge šifre potrebujeta približno enako količino časa. Če imamo na izbiro več kriptosistemov, je bolj optimalna uporaba tistega, ki ima pri enaki stopnji varnosti najmanjšo dolžino ključa, saj ta običajno zahteva najmanjše število operacij pri šifriranju oziroma odšifriranju. Kot je mogoče videti iz spodnje razpredelnice, simetrična kriptografija uporablja najkrajše ključe in zato povzroča najmanj zakasnitev pri komunikaciji, vendar pa jo zaradi težav pri izmenjavi tajnih ključev vedno uporabljamo v kombinaciji z enim izmed asimetričnih kriptosistemov. V primerjavi s simetrično kriptografijo pri enaki stopnji varnosti dolžina ključa pri EC narašča s faktorjem dva, dolžina ključa pri RSA pa z nelinearno odvisnostjo.

Dolžine ključev [bit]			Razmerje med dolžinami ključev	Čas za razbitje šifre [MIPS let]	Predvidena doba uporabe
Simetrična kriptografija	EC	RSA			
80	160	1024	1 : 2 : 12,8	10^{12}	do leta 2010
112	224	2048	1 : 2 : 18,3	10^{24}	do leta 2030
128	256	3072	1 : 2 : 24	10^{28}	po letu 2031
192	384	7680	1 : 2 : 40	10^{47}	
256	521	15360	1 : 2 : 60	10^{66}	

Tabela 1: Primerjava varnosti EC in RSA kriptosistemov glede na dolžine ključev (glej [3])

Kljub temu, da sta EC in RSA oba asimetrična kriptosistema, pa se v mnogo pogledih razlikujeta:

	EC	RSA

Leto nastanka	1985	1977
Avtorji	Miller in Koblitz	Rivest, Shamir in Adleman
Matematični problem, na katerem temelji varnost šifriranja	Problem diskretnega logaritma na eliptični krivulji	Problem faktorizacije naravnih števil, problem diskretnega logaritma
Časovna zahtevnost trenutno najučinkovitejšega algoritma za razbijanje šifre	Eksponentna (Pollardova ρ metoda)	Podeksponentna (metoda kvadratnega sita)
Matematični problem, na katerem temelji varnost digitalnega podpisa	Problem diskretnega logaritma na eliptični krivulji	Problem faktorizacije naravnih števil
Matematični problem, na katerem temelji varnost izmenjave tajnega ključa	Problem diskretnega logaritma na eliptični krivulji	Problem diskretnega logaritma
Enkripcijska shema	ECIES (Elliptic Curve Integrated Encryption Scheme)	RSAES (RSA Encryption Scheme)
Protokol za digitalno podpisovanje	ECDS (Elliptic Curve Digital Signature)	RSA Signature Scheme
Protokol za izmenjavo tajnega ključa	ECDH (Elliptic Curve Diffie-Hellman)	DH (Diffie-Hellman)
Avtoritete	Certicom, SECG (Standards for Efficient Cryptography Group)	RSA Laboratories
PKCS (Public Key Cryptography Standards) standard	PKCS #13	PKCS #1
Standard formata za digitalna potrdila	PKCS#12	PKCS#12

Tabela 2: Primerjava lastnosti EC in RSA kriptosistemov

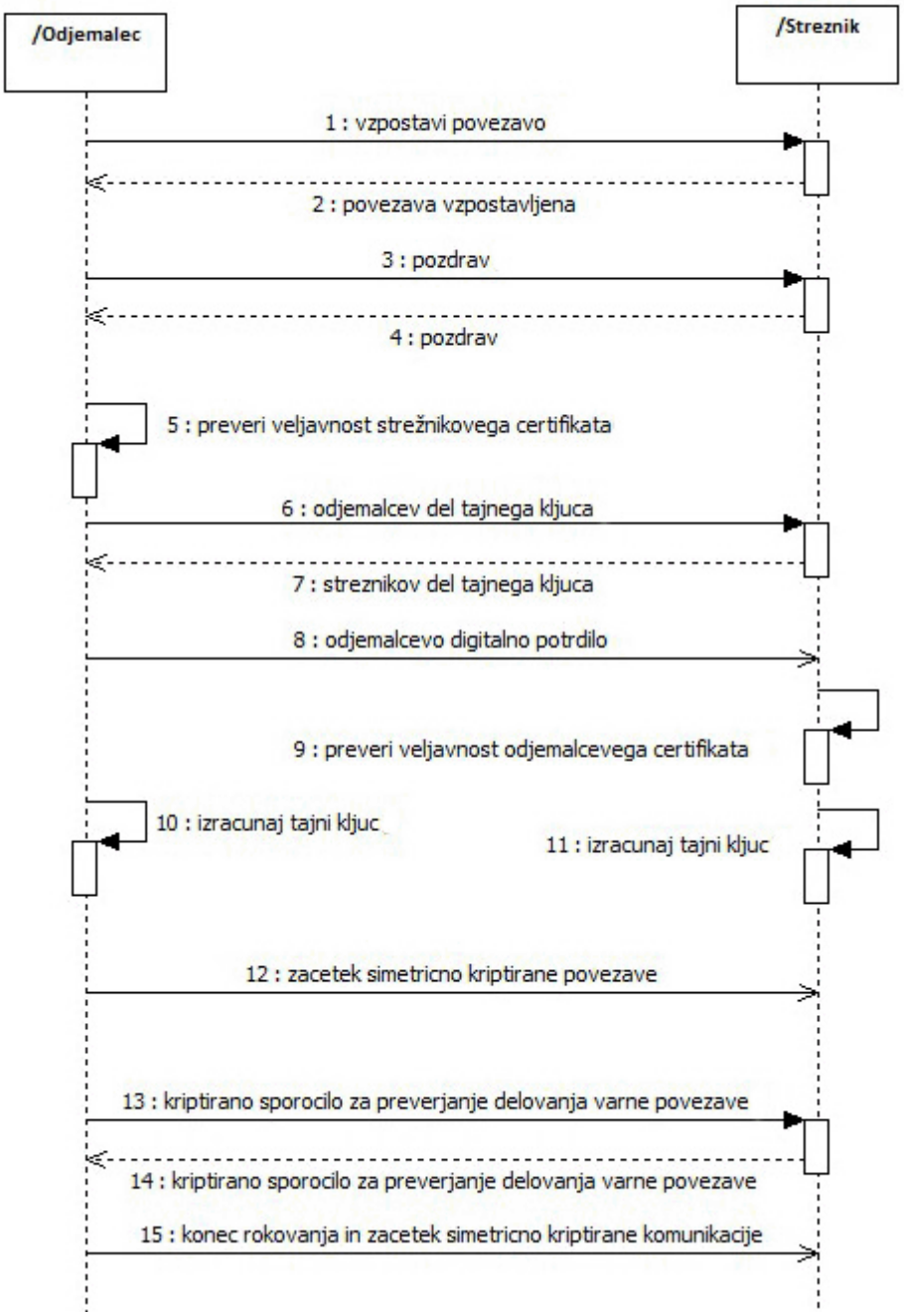
Komunikacijski protokoli z uporabo javnih ključev

SSL

Protokol SSL (Secure Sockets Layer) so razvili leta 1993 pri podjetju Netscape Communications Corporation. Od takrat je doživel več nadgradenj. Trenutna različica se imenuje TLS (Transport Layer Security), verzija 1.2.

Protokol SSL je platformno neodvisen kriptografski komunikacijski protokol, ki temelji na TCP/IP (Transmition Control Protocol / Internet Protocol) protokolu in omogoča aplikacijam varno izmenjavo sporočil tipa odjemalec-strežnik preko svetovnega spleta. Omogoča tudi avtentikacijo ene ali obeh entitet v protokolu.

Izmenjava sporočil med odjemalcem in strežnikom se začne s tako imenovanim rokovanjem (handshake), to je fazo v kateri se stranki dogovorita o parametrih komunikacije. V tej fazi se lahko izvrši tudi avtentikacija ene in druge strani. Ko je rokovanje zaključeno, se prične simetrično šifrirana izmenjava sporočil med odjemalcem in strežnikom. Tak pristop izkorišča prednosti tako asimetrične kot simetrične kriptografije. Prva omogoča avtentikacijo komunicirajočih strank in varen dogovor o ključu, druga pa varno in predvsem hitro izmenjavo sporočil.



Slika 5: Diagram zaporedja poteka vzpostavitve varne povezave preko SSL protokola, ki mu pravimo rokovanje

Potek rokovanja (glej [12]):

1. Odjemalec preko omrežja vzpostavi povezavo s strežnikom.
2. Strežnik sprejme povezavo.

3. Odjemalec pošlje strežniku pozdravno sporočilo, v katerem navede najvišjo verzijo SSL protokola, ki ga odjemalec podpira, seznam kombinacij šifrirnih algoritmov (vsaka kombinacija vsebuje algoritem za izmenjavo ključev, simetrično kriptografijo ter izračun izvlečkov sporočil) in načinov kompresije, ki jih odjemalec podpira ter javne parametre za izmenjavo tajnega ključa.
4. Strežnik odgovori na pozdravno sporočilo.
   4.1. V odgovoru navede izbrano verzijo protokola SSL (vedno izbere najvišjo verzijo, ki jo podpirata oba, tako strežnik kot odjemalec), izbrano kombinacijo šifrirnih algoritmov in način kompresije.
   4.2. Strežnik pošlje svoje digitalno potrdilo, ki ustreza izbranemu šifrirnemu algoritmu .
   4.3. Strežnik pošlje signal za zaključek odgovora na pozdrav.
   4.4. Strežnik pošlje zahtevo po avtentikaciji odjemalca, če je ta prisotna.
5. Odjemalec lahko preveri veljavnost strežnikovega digitalnega potrdila tako, da preveri digitalni podpis certifikatne agencije, ki je podpisala strežnikovo digitalno potrdilo. Če ugotovi, da je bilo strežnikovo digitalno potrdilo resnično podpisano s strani certifikatne agencije, ki ji odjemalec zaupa, potem lahko zaupa tudi strežniku. Ker je v strežnikovem certifikatu zapisan tudi domenski naslov strežnika, se mora ta ujemati z naslovom strežnika, s katerim je odjemalec vzpostavil povezavo. S tem je avtentikacija strežnika zaključena.
6. Odjemalec pošlje strežniku svoj del tajnega ključa (osnovo za izračun tajnega ključa).
7. Strežnik pošlje odjemalcu svoj del tajnega ključa (osnovo za izračun tajnega ključa).
8. Če je strežnik zahteval avtentikacijo odjemalca, slednji pošlje svoje digitalno potrdilo, ter s svojim zasebnim ključem podpisan podatek, ki je znan obema (npr. odjemalčev del tajnega ključa), na podlagi česar lahko strežnik preveri veljavnost odjemalčevega digitalnega potrdila.
9. Odjemalec na podlagi parametrov za generiranje tajnega ključa, ki si jih je izmenjal s strežnikom, izračuna tajni ključ, ki se bo uporabljal za simetrično šifriranje/odšifriranje v nadaljnji izmenjavi sporočil.
10. Strežnik po podobnem postopku kot odjemalec izračuna tajni ključ.
11. Odjemalec pošlje strežniku posebno sporočilo, s katerim oznani začetek simetrično šifrirane komunikacije.

12. Odjemalec pošlje strežniku šifrirano sporočilo, ki vsebuje izvleček vseh dosedanjih odjemalčevih sporočil. Strežnik sporočilo odšifrira in preveri veljavnost izvlečka.
13. Če se vse ujema, potem strežnik prav tako pošlje odjemalcu šifrirano sporočilo z izvlečkom vseh dosedanjih strežnikovih sporočil, ki jih odjemalec odšifrira in preveri njihovo veljavnost.
14. Rokovanje se zaključi. Začne se izmenjava sporočil, šifrirana s simetrično kriptografijo Kot tajni ključ je uporabljen ključ, ki sta ga tekom rokovanja določila strežnik in odjemalec.

Glavne razlike pri rokovanju z uporabo EC in RSA kriptosistema so predvsem pri avtentikaciji in izmenjavi tajnega ključa:

- Pri uporabi EC kriptosistema odjemalec vedno izvede preverjanje strežnikovega digitalnega potrdila (preveri digitalni podpis certifikatne agencije, ki je podpisala digitalno potrdilo), ECDH operacijo za generiranje odjemalčevega dela tajnega ključa in opcijsko tudi ECDSA podpisovanje za svojo avtentikacijo. Strežnik v primeru avtentikacije odjemalca preveri digitalni podpis certifikatne agencije (ECDSA preverjanje), ki je podpisala odjemalčevo digitalno potrdilo in odjemalčev digitalni podpis (ECDSA preverjanje), v vsakem primeru pa izvede ECDH operacijo za generiranje strežnikovega dela tajnega ključa.
- Pri uporabi RSA kriptosistema odjemalec prav tako izvede RSA preverjanje strežnikovega digitalnega potrdila, DH operacijo za generiranje odjemalčevega dela tajnega ključa (ki je enaka RSA šifriranju) in če strežnik zahteva avtentikacijo uporabnika tudi RSA podpisovanje. Strežnik za avtentikacijo odjemalca (če je ta potrebna) preveri digitalni podpis certifikatne agencije in odjemalčev podpis (RSA preverjanje) ter DH operacijo za generiranje strežnikovega dela javnega ključa (ki je enaka RSA odšifriranju).

Tabela 3 prikazuje operacije, ki se izvedejo v primeru avtentikacije strežnika, tabela 4 pa operacije v primeru avtentikacije strežnika in odjemalca.

	ECC	RSA
Odjemalec	$ECDSA_{preverjanje}$ +	$RSA_{prevrerjanje}$ +

	$ECDH_{operacija}$	$RSA_{preverjanje}$
Strežnik	$ECDH_{operacija}$	$RSA_{odšifriranje}$

Tabela 3: Primerjava rokovanja z uporabo EC in RSA kriptosistema z avtentikacijo strežnika (glej [3])

	ECC	RSA
Odjemalec	$ECDSA_{preverjanje}$ + $ECDH_{operacija}$ + $ECDSA_{podpisovanje}$	$RSA_{preverjanje}$ + $RSA_{šifriranje}$ + $RSA_{podpisovanje}$
Strežnik	2 * $ECDSA_{preverjanje}$ + $ECDH_{operacija}$	2 * $RSA_{preverjanje}$ + $RSA_{odšifriranje}$

Tabela 4: Primerjava rokovanja z uporabo EC in RSA kriptosistema z avtentikacijo strežnika in odjemalca (glej [12])

HTTPS

Protokol HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) so prav tako kot SSL razvili pri podjetju Netscape Communications Corporation z namenom, da bi omogočili varno uporabo spleta ter avtenticiranje tako strežnikov kot odjemalcev.

HTTPS ni samostojen protokol ampak gre za kombinacijo protokolov HTTP (HyperText Transfer Protocol) protokola in SSL oziroma TLS protokola. Uporabo HTTPS protokola spoznamo po drugačni shemi omrežnih naslovov, saj se URL (Uniform Resource Locator) začne z besedo https (npr. https://www.gmail.com). Prav tako so privzeta vrata za HTTPS protokol 443 in ne 80, kot je to značilno za protokol HTTP.

Če želimo postaviti spletni strežnik, ki bo uporabljal HTTPS protokol, moramo v nastavitvah podati tudi pot do veljavnega strežnikovega digitalnega potrdila, ki ga bo strežnik uporabljal za avtentikacijo ter šifrirano izmenjavo sporočil. Priporočljivo, ne pa tudi nujno je, da strežnik

uporablja digitalno potrdilo, podpisano s strani priznane certifikatne agencije, saj se na ta način izognemo napadu z oponašanjem.

HTTPS omogoča tudi avtentikacijo uporabnikov oziroma odjemalcev, tako da sam vzdržuje seznam digitalnih potrdil uporabnikov, katerim je dostop do spletnega strežnika dovoljen. V primeru, da je uporabnikov veliko, lahko strežnik vsebuje kar seznam digitalnih potrdil certifikatnih agencij, ki jim zaupa. Tako lahko strežnik sam izdaja digitalna potrdila svojim uporabnikom, namesto seznama veljavnih digitalnih potrdil uporabnikov pa vzdržuje tako imenovano črno listo oziroma seznam preklicanih digitalnih potrdil, ki je načeloma krajša.

Uporabnik, ki želi dostopati do spletne strani preko HTTPS protokola, ki zahteva avtentikacjo uporabnikov, mora imeti v spletnem brskalniku nameščeno svoje digitalno potrdilo ter zasebni ključ, strežnik pa mora imeti uporabnikovo digitalno potrdilo oziroma digitalno potrdilo certifikatne agencije, ki je to potrdilo podpisala, na svojem seznamu. Uporabnik svojo identiteto dokaže z digitalnim podpisom, strežnik pa svojo s svojim digitalnim potrdilom, v katerem je zapisan tudi domenski naslov strežnika, ki se mora ujemati z naslovom strani, do katere odjemalec dostopa.

Poglavje 6

Spletni strežnik

Strojno opremo računalnika, na katerem je nameščen spletni strežnik, ki se bo uporabljal v testu, predstavlja osebni računalnik z operacijskim sistemom Linux. Spletni strežnik je Apache HTTP Server 2.2.8. z OpenSSL kriptografsko knjižnico 0.9.8g. Podpira RSA kriptosistem z dolžinami ključev 1024, 2048, 3072 in 4096 bitov ter EC kriptosistem z dolžinami ključev 256, 384 in 521 bitov. Za vključitev podpore v Apache-ju in OpenSSL je potrebno prevesti izvorno kodo obeh programov z vključitvijo posebnih zastavic.

Apache HTTP strežnik

Spletni strežnik Apache je nastal kot nadaljevanje projekta imenovanega NCSA HTTPd, ki so ga razvijali na Univerzi v Illinoisu. Ko je razvoj projekta leta 1994 zastal zaradi odhoda glavnega programerja iz omenjene univerze, je naslednje leto pobudo prevzela manjša skupina programerjev, ki so ta spletni strežnik uporabljali za razvoj svojih spletnih aplikacij. Odločili so se, da bodo nadaljevali razvoj projekta in tako je nastal Apache spletni strežnik, ki je po slabem letu postal najbolj priljubljen spletni strežnik na svetu.

Leta 1999 je skupina razvijalcev ustanovila Apache Software Fundation – fundacijo, ki predstavlja organizacijsko, pravno in finančno podporo vsem projektom, ki se odvijajo pod njenim okriljem.

Spletni strežnik Apache danes uživa približno petdeset odstotni delež med spletnimi strežniki po svetovnem spletu, predvsem na račun odprto-kodne licence, podpore več operacijskim sistemom, obsežne funkcionalnosti, robustnosti in prilagodljivosti.

Delovanje spletnega strežnika Apache

Spletni strežnik Apache poleg protokola HTTP (Hypertext Transfer Protocol) protokola podpira tudi protokol HTTPS (Hypertext Transfer Protocol over Secure Socket Layer)

protokol, ki predstavlja kombinacijo HTTP in SSL (Secure Sockets Layer) protokolov, torej omogoča varno strežbo spletnih strani. Pravzaprav spletni strežnik Apache omogoča hkratno uporabo obeh, tako HTTP kot HTTPS protokola na nekem spletnem mestu, tako da je vsebina vsake datoteke oziroma imenika spletnega mesta uporabnikom na voljo preko enega ali drugega protokola. Torej lahko dosežemo, da je del spletišča prosto dostopen komurkoli, ki bi ga rad uporabljal, del spletnega mesta pa je namenjen le pooblaščenim uporabnikom.

OpenSSL

Open SSL je odprtokodna kriptografska knjižnica, ki vsebuje tudi implementacijo protokola SSL oziroma protokola TLS. Na voljo je za različne operacijske sisteme. Podpora spletnega strežnika Apache protokolu HTTPS temelji na OpenSSL kriptografski knjižnici.

Poleg SSL pa OpenSSL podpira večje število kriptosistemov, med drugim tudi RSA in EC (če želimo vključiti podporo EC, moramo prevesti izvorno kodo OpenSSL s posebnimi stikali). Vsebuje tudi funkcionalnosti za generiranje digitalnih potrdil in upravljanje z njimi.

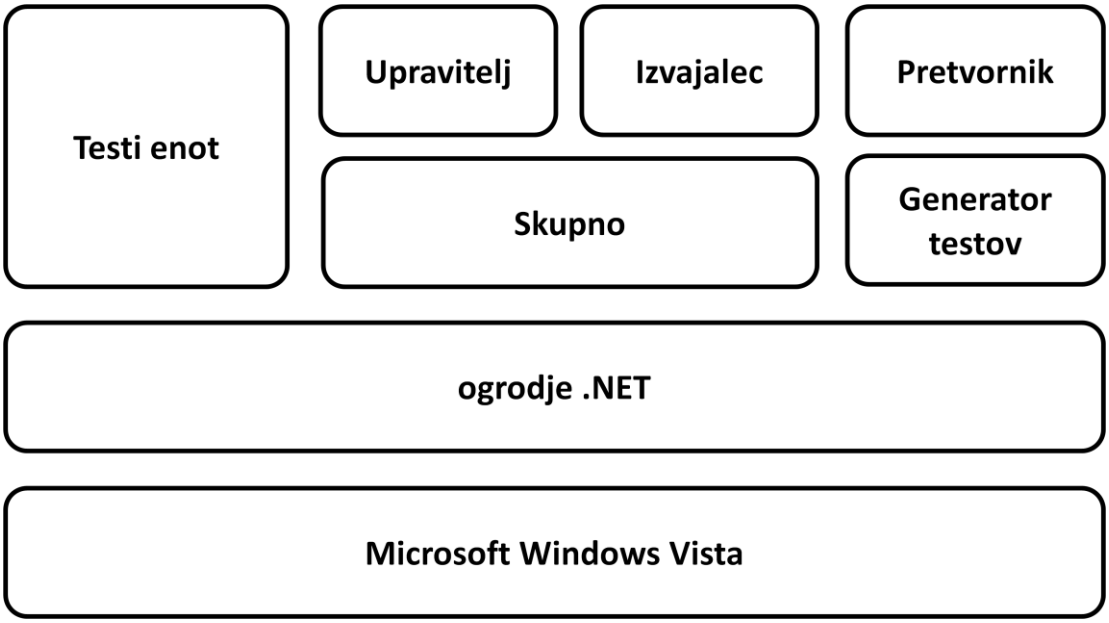
Poglavje 7

Aplikacija

Za potrebe te diplome je bila izdelana aplikacija, ki omogoča izvajanje obremenitvenih testov. Razvita je bila v ogrodju .NET 3.5 z uporabo programskega jezika C#. Izvaja se na operacijskem sistemu Microsoft Windows Vista. Kot načrtovalno orodje je bil uporabljen Star UML, kot razvojno orodje pa Microsoft Visual Studio 2008.

Struktura aplikacije

Aplikacija je razdeljena na šest sklopov: Upravitelj, Izvajalec, Skupno, Generator testov, Pretvornik in Testi enot.



Slika 6: Struktura aplikacije

Lastnosti aplikacije:

- Konzolna: ker izvajanje testa odzivnosti ne potrebuje uporabniške interakcije (z izjemo izdelave nastavitvene datoteke pred začetkom izvajanja), bi bilo v tem

primeru nesmiselno izdelati kakršen koli uporabniški vmesnik (npr. okenski ali spletni).

- Nastavljiva: ker želimo sami določati in spreminjati parametre testa, potrebujemo način, kako to tudi storiti. Tekstovna nastavitvena datoteka v XML formatu se je izkazala kot dobra rešitev, saj omogoča enostavno, pregledno in strukturirano zgradbo, poleg tega pa je XML datoteke enostavno uporabljati preko programskega vmesnika z uporabo serializacije.
- Porazdeljena: spletni strežnik Apache velja za visoko optimiziranega in če ga želimo obremeniti do te mere, da bomo lahko videli očitne razlike v zakasnitvah pri izvajanju zahtev, potrebujemo več računalnikov, ki bodo simultano izvajali obremenitveni test. Komunikacija poteka med računalnikom, kjer se izvaja upravitelj testa ter računalniki, ki so gostitelji izvajalca testa, z uporabo omrežnih vtičnikov (socketov). Vsa sporočila, ki potujejo po omrežju se ob vhodu v omrežje binarno serializirajo, na drugi strani pa binarno deserializirajo nazaj v prvotno obliko. Za komunikacijo se uporablja interni protokol, ki določa nabor ukazov in pripadajoče parametre ter rezultate operacij.
- Razširljiva: kadar želimo povečati stopnjo obremenitve spletnega strežnika, pa tega zaradi omejene procesorske moči računalnikov, na katerih se izvaja obremenitveni test, ni mogoče storiti, preprosto dodamo nove računalnike. Nanje namestimo Izvajalca testa ter dodamo podatke novih računalnikov v konfiguracijsko datoteko obremenitvenega testa pred začetkom naslednjega testa.
- Nadgradljiva: v primeru, da želimo dodati nove funkcionalnosti, imamo na voljo več programskih vmesnikov, ki jih lahko implementiramo in tako dosežemo želene spremembe.
- Platformno odvisna: kljub temu, da je ogrodje .NET izdelano kot platformno neodvisno, pa smo v aplikaciji vezani na Microsoft Windows Vista operacijski sistem, saj le v družini Microsoft Windows operacijskih sistemov podpira certifikate, narejene z EC kriptosistemom. Za operacijski sistem Linux obstaja implementacija ogrodja .NET imenovana Mono, torej bi bilo vsaj teoretično aplikacijo mogoče izvajati tudi na Linuxu, vendar je to le ideja, ki bi jo veljalo preizkusiti tudi v praksi.

Upravitelj

Glavna naloga upravitelja je, kot namiguje že samo ime, upravljanje testa. Upravitelj je večnitna konzolna aplikacija, ki z izvajalci komunicira preko omrežnih vtičnikov. Med svojim delovanjem vodi dnevnik izvajanja, kar olajša vpogled v izvajanje aplikacije ter odpravljanje morebitnih napak.

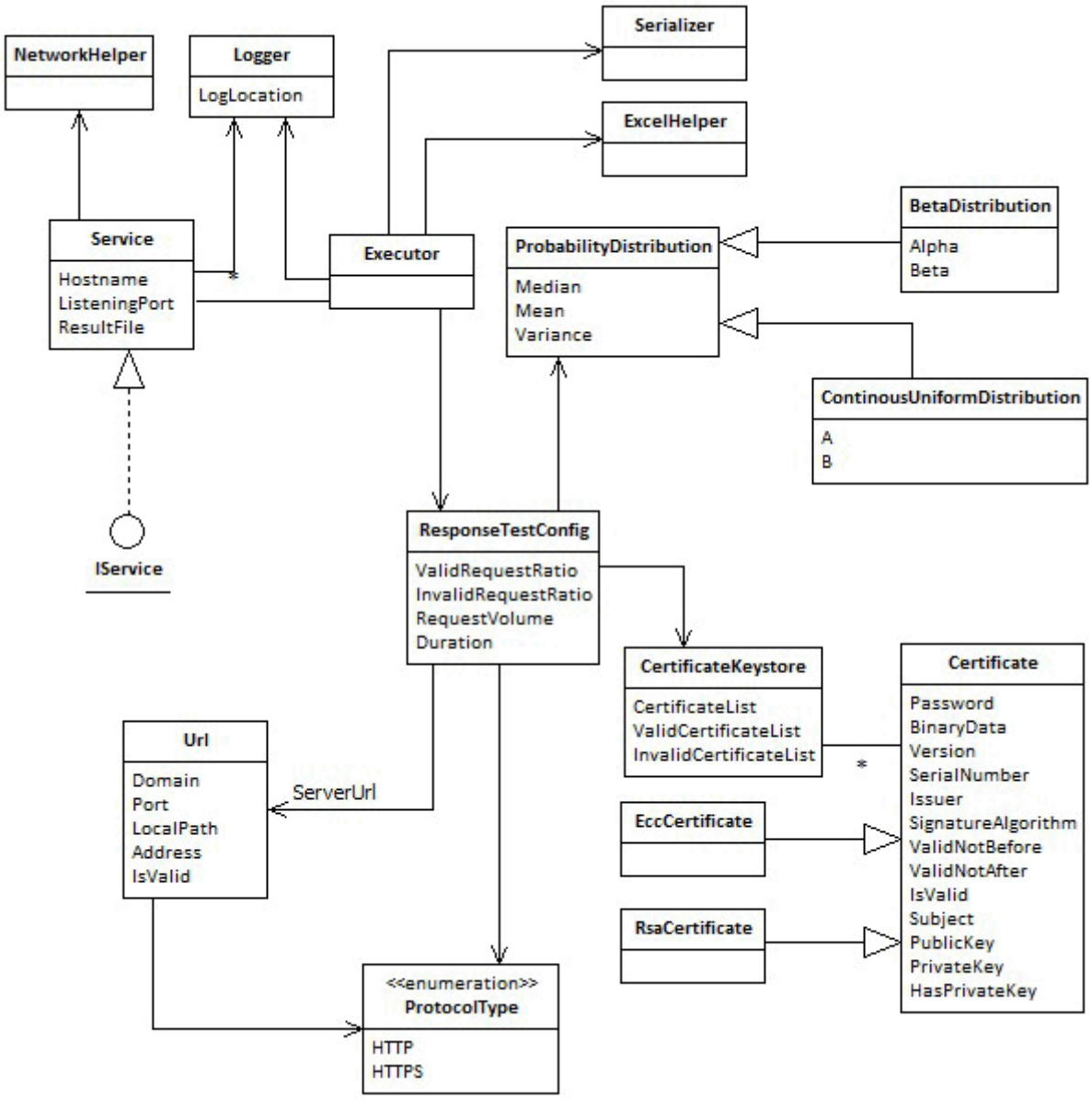
Upravitelj se izvaja na svojem računalniku, za svoje delovanje potrebuje Microsoft Windows Vista operacijski sistem ter nameščeno ogrodje .NET 3.5. Prav tako mora imeti prost omrežni dostop do vseh računalnikov, kjer se izvajajo izvajalci testa. Program poženemo v ukazni vrstici, kot parameter pa podamo pot do nastavitvene datoteke testa.

Nastavitvena datoteka je tekstovna datoteka v XML formatu, kjer so zapisani vsi parametri, potrebni za izvajanje testa. Vsebuje naslednje parametre: pot do dnevnika, parametre testa in podatke o računalnikih, kjer se izvajajo izvajalci testa. Parametri testa vsebujejo: trajanje testa (v sekundah), število zahtev, ki se morajo izvesti v tem času, delež veljavnih in neveljavnih zahtev, verjetnostno porazdelitev zaporedja izvajanja zahtev (podprti sta Beta ter zvezna geometrijska verjetnostna porazdelitev), protokol (HTTP ali HTTPS), naslov strežnika, vrata, na katerih posluša spletni strežnik ter pot do posameznih digitalnih potrdil in pripadajočih gesel.

Primer nastavitvene datoteke:

```
<?xml version="1.0" encoding="utf-8"?>
<ResponseTestConfig
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Log>D:\Tests\executor.log</Log>
  <RequestParameters>
    <Duration>300</Duration>
    <RequestVolume>700</RequestVolume>
    <ValidRequestRatio>1</ValidRequestRatio>
    <InvalidRequestRatio>0</InvalidRequestRatio>
    <ProbabilityDistribution>
      <ContinousUniformDistribution>
        <A>0</A>
        <B>1</B>
      </ContinousUniformDistribution>
    </ProbabilityDistribution>
    <ProtocolType>SSL</ProtocolType>
    <ServerAddress>
      <Hostname>212.235.189.45</Hostname>
```

```
      <Port>443</Port>
      <LocalPath>025kb.html</LocalPath>
    </ServerAddress>
  </RequestParameters>
  <Certificates>
    <Certificate>
      <File>D:\Keys\RSA\1024b\rsa1024-8E.p12</File>
      <Password>zrzgjoiog</Password>
    </Certificate>
    <Certificate>
      <File>D:\Keys\RSA\1024b\rsa1024-8F.p12</File>
      <Password>oiuvyvpou</Password>
    </Certificate>
  </Certificates>
  <ServiceAddresses>
    <ServiceAddress>
      <Hostname>212.235.182.205</Hostname>
      <Port>9999</Port>
      <ResultFile>D:\Tests\212.235.182.205.txt</ResultFile>
    </ServiceAddress>
    <ServiceAddress>
      <Hostname>212.235.182.207</Hostname>
      <Port>9999</Port>
      <ResultFile>D:\Tests\212.235.182.207.txt</ResultFile>
    </ServiceAddress>
    <ServiceAddress>
      <Hostname>212.235.182.210</Hostname>
      <Port>9999</Port>
      <ResultFile>D:\Tests\212.235.182.210.txt</ResultFile>
    </ServiceAddress>
  </ServiceAddresses>
</ResponseTestConfig>
```



Slika 7: Razredni diagram Upravitelja prikazuje razrede Upravitelja in relacije med njimi

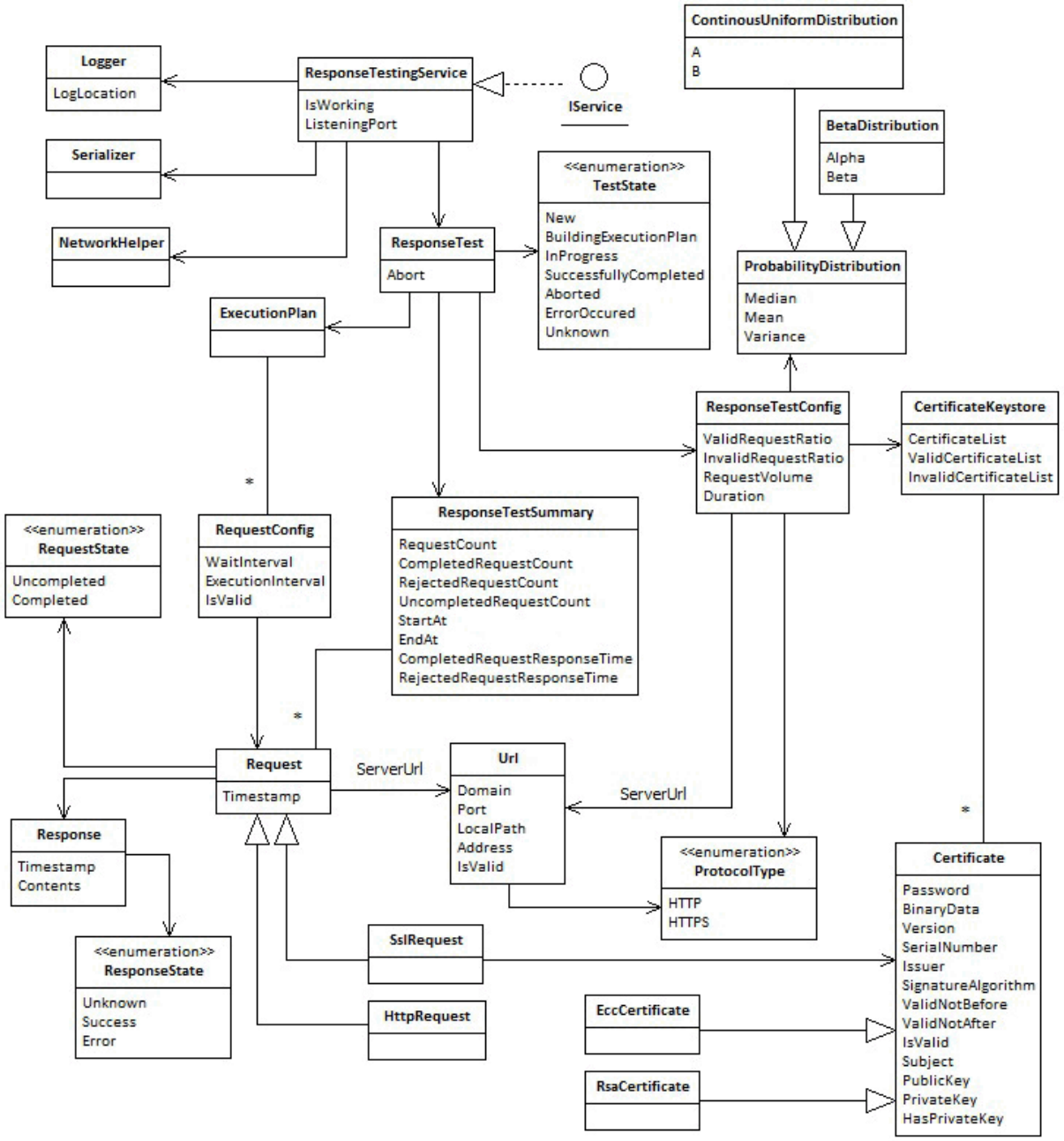
Izvajalec

Naloga izvajalca je izvajanje obremenitvenega testa. Izvajalec deluje pod nadzorom Upravitelja ter izvaja zahteve na spletni strežnik in meri njihove odzivne čase. Upravitelj Izvajalcu narekuje pričetek testa, parametre testa, ima pa tudi možnost predčasne prekinitve izvajanja testa. Za razliko od Upravitelja in spletnega strežnika, je lahko Izvajalcev tudi več, kar je pravzaprav namen arhitekture aplikacije, saj lahko več Izvajalcev bolj obremeni spletni strežnik in tako omogoči natančnejšo analizo odzivnosti strežnika. Izvajalec je tako z Upraviteljem, kot s strežnikom povezan preko računalniškega omrežja. Vsa komunikacija

med Upraviteljem in Izvajalcem poteka po nešifriranem internem protokolu, komunikacija s spletnim strežnikom pa poteka preko HTTP oziroma HTTPS protokola.

Izvajalec deluje kot konzolna aplikacija. Posluša na omrežnih vratih, ki jih določimo ob zagonu Izvajalca, prav tako lahko ob zagonu določimo pot do datoteke, v katero se bo zapisoval dnevnik izvajanja testa. Za uspešno komunikacijo s Strežnikom in Upraviteljem, mora požarni zid omogočati njen dostop do omrežja.

Izvajalec pasivno čaka na sporočilo oziroma ukaz s strani Upravitelja. Ko ta ukaz prejme, ga najprej deserializira (vsa sporočila se namreč serializirajo pred vstopom v omrežje) ter ga interpretira. Vkolikor ukazu sledijo tudi parametri, počaka nanje, nato pa izvede ukaz. Po izvedbi ukaza v odgovor upravitelju pošlje rezultat izvajanja ukaza. Vsako dejanje zapiše tudi v dnevnik, kot se zgodi tudi v primeru izjeme, kar omogoča lažje sledenje dogajanju med testom ter lažje odpravljanje napak.



Slika 8: Razredni diagram Izvajalca prikazuje razrede Izvajalca in relacije med njimi

Skupno

Skupno predstavlja programsko knjižnico, ki vsebuje vso funkcionalnost in programske vmesnike, ki so skupni tako Upravitelju kot Izvajalcu. Nesmiselno bi namreč bilo dvakrat programirati iste stvari, kar bi vodilo v nekonsistentnost, povečalo možnosti nastanka napak in otežilo njihovo odpravljanje.

Generator testov

Zaradi večjega števila testov, ki jih je bilo potrebno opraviti (več o tem v poglavju Analiza odzivnosti), se je pojavila potreba po programu, ki bi znal sam generirati vse datoteke, potrebne za izvajanje testov. Generator testov tako za vsakega izmed testov sestavi namestitveno datoteko testa, skripto za izvajanje testa, skripto za spremljanje pisanja v dnevnik izvajanja Upravitelja ter skripto za pretvarjanje prejetih rezultatov v Excelovo datoteko.

Pretvornik

Prejeti rezultati iz vsakega izmed testnih računalnikov se zapišejo v tekstovno datoteko v binarno serializirani obliki. Ker so kot taki neberljivi in zato neuporabni, se je pojavila potreba po programu, ki bi znal te podatke pretvoriti v primeren format za nadaljnjo obdelavo in ravno to je naloga Pretvornika. Ta rezultate zapiše v tabelarični obliki v Excelovo preglednico, ki jih je mogoče z uporabo formul in vrtilnih tabel tudi analizirati.

from [ms]	to [ms]	duration [ms]	protocol	source	destination	contents
390,63	406,25	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
609,38	625,00	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
1.000,00	1.015,63	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
1.390,63	1.406,25	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
2.156,25	2.171,88	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
2.515,63	2.531,25	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
2.765,63	2.781,25	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
3.328,13	3.343,75	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
3.875,00	3.890,63	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
4.359,38	4.468,75	109,38	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
4.562,50	4.578,13	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
5.078,13	5.093,75	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
5.546,88	5.562,50	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
6.046,88	6.062,50	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...
6.500,00	6.515,63	15,63	SSL	212.235.182.205	https://212.235.189.45:9999/025kb.html	PaZSdEVgPaBSmNP...

Tabela 3: Primer delnega izpisa rezultatov enega izmed opravljenih testov

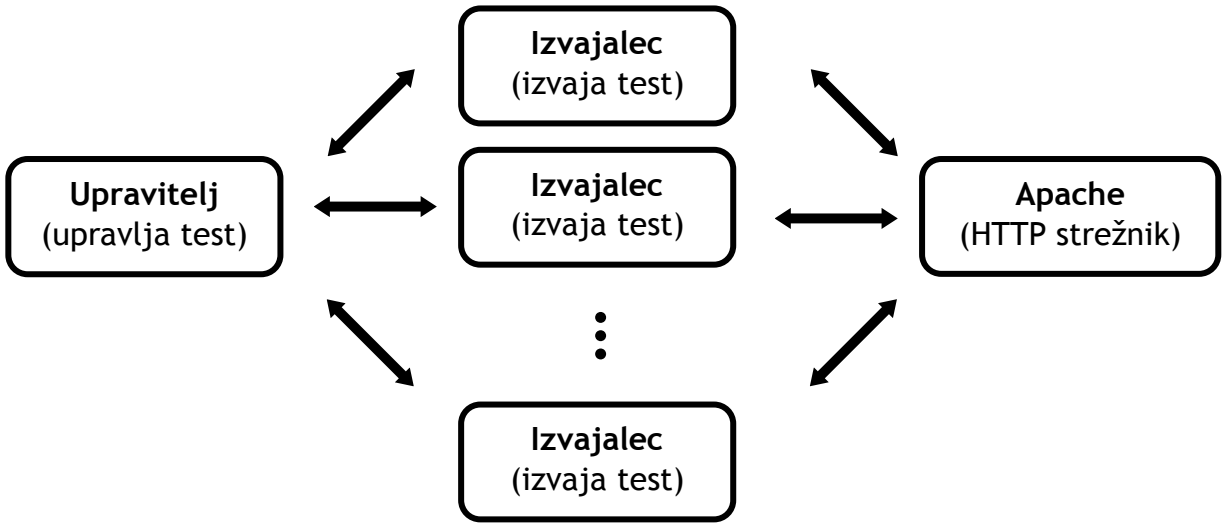
NUnit testi

NUnit testi združujejo množico unit testov, namenjenih testiranju delovanja posameznih razredov aplikacije. Ideja, ki stoji za unit testi je ta, da je potrebno temeljito preveriti vse metode razredov in stanja, v katerih se razred lahko nahaja.

Postavitev aplikacije

Aplikacija je porazdeljena, torej se istočasno izvaja na več računalnikih. Računalnike delimo v tri skupine: računalnik, na katerem se izvaja Upravitelj, računalnik, na katerem se izvaja spletni strežnik ter računalnike, na katerih teče Izvajalec. Upravitelj in strežnik sta vedno po en, medtem ko je izvajalcev lahko več (glej sliko 9).

Računalniki so med seboj povezani preko TCP/IP omrežja. Komunikacija med Upraviteljem in Izvajalcem poteka preko internega protokola, komunikacija med strežnikom in izvajalcem pa preko HTTPS protokola.



Slika 9: Postavitev aplikacije

Ker sta protokola HTTP in HTTPS platformno neodvisna, ima lahko strežnik nameščen poljuben operacijski sistem ter poljuben spletni strežnik, ki podpira protokol HTTP oziroma HTTPS.

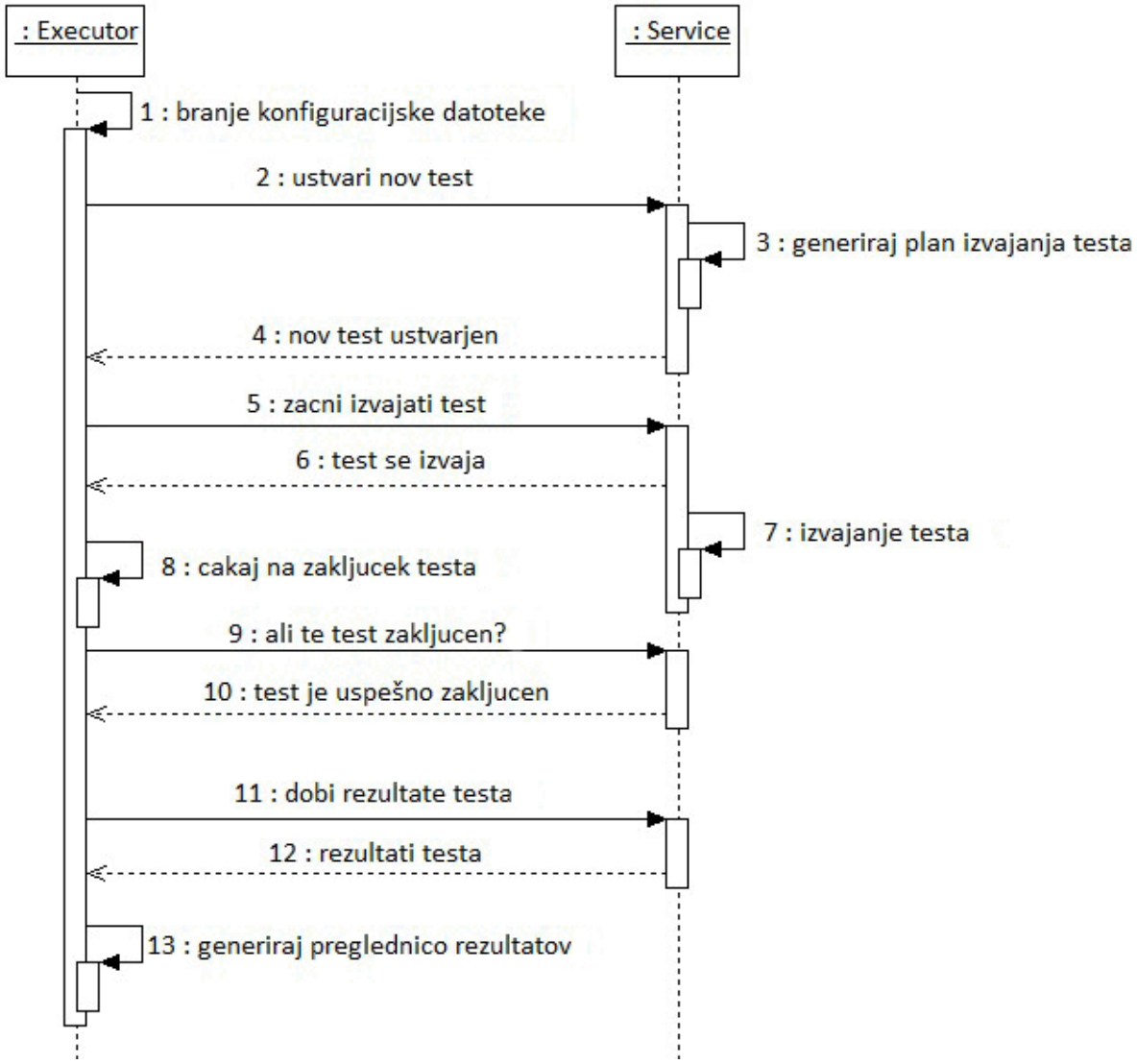
Testni računalniki

Testni računalniki se delijo v dve skupini: računalnik, na katerem se izvaja Upravitelj ter računalniki, na katerih se izvajajo Izvajalci. Prvi je lahko samo eden, slednjih pa je lahko več.

Vsi testni računalniki morajo imeti nameščen operacijski sistem Microsoft Windows Vista, saj le ta podpira EC kriptosistem, prav tako morajo imeti nameščen .NET Framework 3.5.

Izvajanje testa

Med izvajanjem aplikacije je vedno prisoten en Upravitelj in eden ali več Izvajalcev. V najenostavnejši obliki je prisoten en sam Izvajalec, oba, tako Upravitelj kot Izvajalec pa se izvajata na istem računalniku. Vkolikor se pojavi potreba po večji obremenitvi spletnega strežnika, dodamo v sistem več omrežno povezanih računalnikov, na vsakem izmed njih pa se pod nadzorom istega Upravitelja izvaja drug Izvajalec.



Slika 10: Diagram zaporedij izvajanja testa

Potek izvajanja testa:

1. Upravitelj prebere namestitveno datoteko, ki jo prejme kot parameter ob zagonu ter uvozi podatke o testnih računalnikih, nastavitve testa in digitalna potrdila.
2. Vsem testnim računalnikom pošlje ukaz, naj ustvarijo nov test, kot parameter pa jim pošlje konfiguracijo testa in digitalna potrdila, ki jih bodo potrebovali za izvajanje HTTPS zahtev.
3. Izvajalec posluša na omrežnih vratih določenih ob zagonu in pasivno čaka na ukaz. Ko prejme ukaz za nov test ter parametre izvajanja, zgradi plan testiranja, ki vsebuje po časovnih intervalih določeno zaporedje izvajanja zahtev. Kljub temu, da bi se lahko plan testiranja gradil sproti med izvajanjem, pa predčasna zgraditev plana zmanjša zakasnitve zaradi potratnih numeričnih operacij med samim testom in tako omogoča večjo gostoto izvajanja zahtev.
4. Izvajalec upravitelju odgovori, da je pripravljen na začetek izvajanja testa.
5. Upravitelj izda ukaz za začetek izvajanja obremenitvenega testa.
6. Izvajalec pošlje potrditev o začetku izvajanja testa.
7. Izvajalec začne z izvajanjem zahtev na spletni strežnik. Med izvajanjem meri odzivni čas vsakega odgovora na zahtevo, podatke o vsaki zahtevi pa zapisuje v časovno urejen seznam.
8. Upravitelj čaka na konec izvajanja obremenitvenega testa. Čaka toliko časa, dokler se test ne zaključi.
9. Upravitelj pošlje izvajalcu poizvedbo o stanju izvajanja testa. Vkolikor se test ne zaključi v predvidenem času, upravitelj v enakih časovnih intervalih poizveduje pri izvajalcu, ali je že zaključil z izvajanjem testa.
10. Ko izvajalec uspešno zaključi z izvajanjem testa, o tem obvesti upravitelja.
11. Upravitelj pošlje zahtevo po rezultatih izvajanja obremenitvenega testa.

12. Izvajalec v odgovor upravitelju pošlje časovno urejen seznam izvajanja zahtev in odgovorov nanje. Prav tako Upravitelju pošlje dnevnik izvajanja testa.

13. Upravitelj prejete rezultate v binarno serializirani obliki shrani v datoteko, ki jo je mogoče s Pretvornikom pretvoriti v Excelovo preglednico, ki uporabniku omogoča nadaljnjo statistično obdelavo rezultatov.

V primeru več izvajalcev poteka izmenjava sporočil simultano in neodvisno med Upraviteljem in vsakim posameznim Izvajalcem. Tak način je dosežen z uporabo večnitnosti, kjer se vsaka nit obnaša kot samostojen proces znotraj aplikacije. Niti so med seboj neodvisne, vendar pa imajo mehanizme, ki jim omogočajo medsebojno komunikacijo. Za vsakega izvajalca Upravitelj zažene svojo nit, ki se ob koncu izvajanja testa tudi zaključi.

Primer Upraviteljevega dnevnika izvajanja testa:

```
15.07.08 20:04:39.221 [INFO] Request parameters successfully imported from configuration file.
15.07.08 20:04:39.236 [INFO] Server parameters successfully imported from configuration file.
15.07.08 20:04:39.236 [INFO] Certificates successfully imported from configuration file.
15.07.08 20:04:39.674 [INFO] Probability distribution successfully imported from configuration file.
15.07.08 20:04:39.674 [INFO] Configuration successfully imported from config file.
15.07.08 20:04:39.689 [INFO] Creating new test on 212.235.182.205:9999
15.07.08 20:04:40.252 [INFO] New test created on 212.235.182.205:9999
15.07.08 20:04:40.252 [INFO] Creating new test on 212.235.182.207:9999
15.07.08 20:04:40.877 [INFO] New test created on 212.235.182.207:9999
15.07.08 20:04:44.330 [INFO] Test state on 212.235.182.207:9999 is Ready
15.07.08 20:04:44.330 [INFO] Starting test on 212.235.182.207:9999
15.07.08 20:04:44.330 [INFO] 212.235.182.207:9999 is waiting for other services to become ready
15.07.08 20:04:44.330 [INFO] Test state on 212.235.182.205:9999 is Ready
15.07.08 20:04:44.330 [INFO] Starting test on 212.235.182.205:9999
15.07.08 20:04:44.330 [INFO] 212.235.182.205:9999 is waiting for other services to become ready
15.07.08 20:04:47.330 [INFO] 212.235.182.205:9999: all services are ready
15.07.08 20:04:47.330 [INFO] 212.235.182.207:9999: all services are ready
15.07.08 20:04:48.127 [INFO] Test state on 212.235.182.205:9999 is InProgress
15.07.08 20:04:48.127 [INFO] Test successfully started on 212.235.182.205:9999
15.07.08 20:04:48.127 [INFO] Waiting for test to finish on 212.235.182.205:9999
15.07.08 20:04:48.142 [INFO] Test state on 212.235.182.207:9999 is InProgress
15.07.08 20:04:48.142 [INFO] Test successfully started on 212.235.182.207:9999
15.07.08 20:04:48.142 [INFO] Waiting for test to finish on 212.235.182.207:9999
15.07.08 20:05:18.127 [INFO] Test should have finished by now on 212.235.182.205:9999
15.07.08 20:05:18.142 [INFO] Test should have finished by now on 212.235.182.207:9999
15.07.08 20:05:19.533 [INFO] Test state on 212.235.182.207:9999 is SuccessfullyCompleted
15.07.08 20:05:19.533 [INFO] Test successfully completed on 212.235.182.207:9999
15.07.08 20:05:19.533 [INFO] Exporting test results from 212.235.182.207:9999 to C:\Test\212.235.182.207.txt
15.07.08 20:05:19.627 [INFO] Test state on 212.235.182.205:9999 is SuccessfullyCompleted
15.07.08 20:05:19.627 [INFO] Test successfully completed on 212.235.182.205:9999
15.07.08 20:05:19.627 [INFO] Exporting test results from 212.235.182.205:9999 to C:\Test\212.235.182.205.txt
```

15.07.08 20:05:19.861 [INFO] Results successfully exported from 212.235.182.205:9999 to C:\Test\212.235.182.205.txt
15.07.08 20:05:19.955 [INFO] Results successfully exported from 212.235.182.207:9999 to C:\Test\212.235.182.207.txt
15.07.08 20:05:20.064 [INFO] Service log obtained from 212.235.182.205:9999 and saved to: C:\Test\212.235.182.205.log
15.07.08 20:05:20.174 [INFO] Service log obtained from 212.235.182.207:9999 and saved to: C:\Test\212.235.182.207.log

Primer Izvajalčevega dnevnika izvajanja testa:

15.07.08 20:04:34.900 [INFO] Incomming connection established with: 212.235.182.208:53885
15.07.08 20:04:35.072 [INFO] Message recievced: NewTest()
15.07.08 20:04:35.150 [INFO] Creating new test
15.07.08 20:04:35.197 [INFO] Building execution plan
15.07.08 20:04:35.197 [INFO] Building request sequence
15.07.08 20:04:35.197 [INFO] New test created.
15.07.08 20:04:35.213 [INFO] Sending response
15.07.08 20:04:35.260 [INFO] Setting invalid requests
15.07.08 20:04:35.260 [INFO] Execution plan completed
15.07.08 20:04:35.572 [INFO] Connection closed
15.07.08 20:04:39.072 [INFO] Incomming connection established with: 212.235.182.208:53888
15.07.08 20:04:39.275 [INFO] Message recievced: GetTestState()
15.07.08 20:04:39.275 [INFO] Test state is: Ready
15.07.08 20:04:39.275 [INFO] Sending response
15.07.08 20:04:39.635 [INFO] Connection closed
15.07.08 20:04:42.479 [INFO] Incomming connection established with: 212.235.182.208:53889
15.07.08 20:04:42.682 [INFO] Message recievced: StartTest()
15.07.08 20:04:42.682 [INFO] Test started.
15.07.08 20:04:42.697 [INFO] Sending response
15.07.08 20:04:43.057 [INFO] Connection closed
15.07.08 20:04:43.057 [INFO] Incomming connection established with: 212.235.182.208:53891
15.07.08 20:04:43.088 [INFO] Message recievced: GetTestState()
15.07.08 20:04:43.088 [INFO] Test state is: InProgress
15.07.08 20:04:43.088 [INFO] Sending response
15.07.08 20:04:43.447 [INFO] Connection closed
15.07.08 20:05:14.322 [INFO] Incomming connection established with: 212.235.182.208:53893
15.07.08 20:05:14.525 [INFO] Message recievced: GetTestState()
15.07.08 20:05:14.525 [INFO] Test state is: SuccessfullyCompleted
15.07.08 20:05:14.525 [INFO] Sending response
15.07.08 20:05:14.900 [INFO] Connection closed
15.07.08 20:05:14.900 [INFO] Incomming connection established with: 212.235.182.208:53896
15.07.08 20:05:15.025 [INFO] Message recievced: GetTestResults()
15.07.08 20:05:15.025 [INFO] Returning test results
15.07.08 20:05:15.025 [INFO] Sending response
15.07.08 20:05:15.150 [INFO] Connection closed
15.07.08 20:05:15.150 [INFO] Incomming connection established with: 212.235.182.208:53897
15.07.08 20:05:15.260 [INFO] Message recievced: GetLog()
15.07.08 20:05:15.260 [INFO] Returning log contents

Prednosti in pomanjkljivosti aplikacije

Prednosti aplikacije, kot že omenjeno so nastavljivost, razširljivost, porazdeljenost in nadgradljivost.

Največja pomanjkljivost pa je platformna odvisnost, saj je aplikacija zaradi podpore eliptičnim krivuljam vezana na operacijski sistem Microsoft Windows Vista. Vsekakor bi bila platformna neodvisnost prednost, vendar pa pri izvajanju testa to ni predstavljalo večjih ovir glede na to, da so računalniki v laboratoriju imeli možnost uporabe navideznih strojev, ki takšne težave premoščajo.

Uporabljene tehnologije

- Strežnik: kot spletni strežnik je bil uporabljen Apache HTTP Server 2.2.8 (http://httpd.apache.org/), z OpenSSL 0.9.8g (http://www.openssl.org/).
- Načrtovanje: kot sistem za načrtovanje je bil uporabljen jezik za modeliranje UML (http://www.uml.org/) in orodje StarUML 5 (http://www.staruml.com/).
- Razvoj: kot razvojno orodje je bil uporabljen Microsoft Visual Studio 2008 (http://msdn2.microsoft.com/en-us/vstudio/default.aspx) in .NET ogrodje 3.5 (http://www.microsoft.com/downloads/details.aspx?FamilyID=333325fd-ae52-4e35-b531-508d977d32a6&displaylang=en). Za potrebe dela z verjetnostjo je bila uporabljena odprtokodna matematična knjižnica MathNet.Iridium (http://mathnet.opensourcedotnet.info/), za delo s nastavitvenimi datotekami pa jezik XML (eXtended Markup Language, http://www.w3.org/XML/)
- Testiranje: za potrebe testiranja je bilo uporabljeno orodje NUnit (http://www.nunit.org/), ki za testiranje uporablja unit teste.

Poglavje 8

Analiza odzivnosti

Namen analize je bil izdelati primerjavo izmerjenih povprečnih odzivnih časov glede na različne dolžine ključev. Primerjala naj bi se odzivnost 1024, 2048 in 3072 bitnih dolžin ključev z uporabo RSA kriptosistema ter odzivnost 160, 224 in 256 bitnih dolžin ključev z uporabo EC kriptosistema. Te dolžine ključev so paroma (glej Tabelo 1) primerljive glede na stopnjo varnosti, je primerjava njihove odzivnosti smiselna. Vendar pa se je izkazalo, da tako Microsoft Windows Vista, kot Apache spletni strežnik podpirata le EC z dolžinami ključev 256, 384 in 521 bitov ter RSA z dolžinami ključev 1024, 2048, 3072 in 4096 bitov (glej poglavje 5). Tako je za primerjavo ostala le kombinacija 3072 bitov dolgih RSA ključev in 256 bitov dolgih EC ključev. Analiza tako vsebuje vse podprte dolžine ključev; 1024, 2048, 3072 in 4096 RSA kriptosistema ter 256, 384 in 521 EC kriptosistema.

Vsaka dolžina ključa je bila testirana v več različicah:

- Pri zahtevah datotek dolžine 0kB, 25kB, 50kB in 100kB.
- Z avtentikacijo strežnika in odjemalca (v Tabeli 6 kratica **asio** – avtentikacija strežnika in odjemalca).
- Samo z avtentikacijo strežnika (v Tabeli 6 kratica **as** – avtentikacija strežnika).

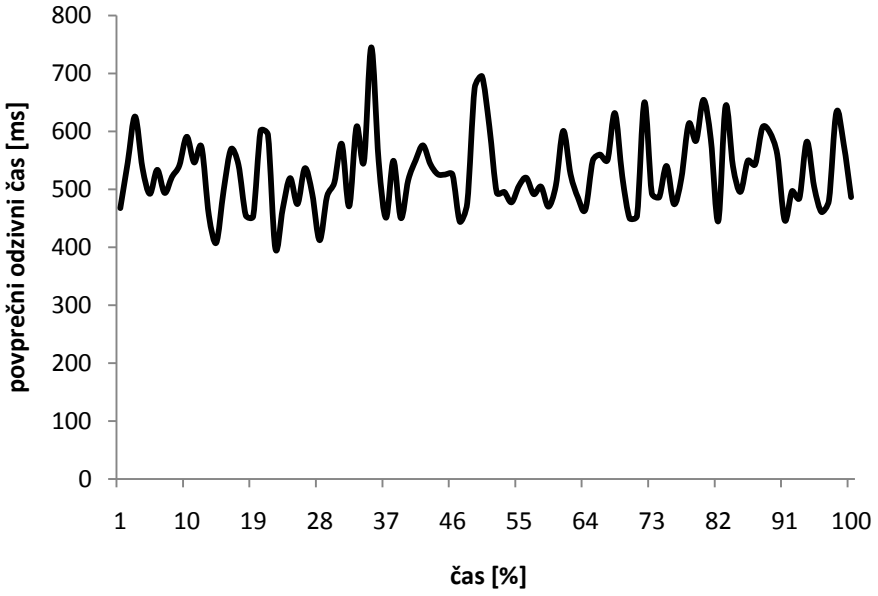
Opraviti bi bilo potrebno 84 testov (glej tabelo 6), vendar pa testi 5-20 ter 77-84 zaradi omejitev uporabljene programske opreme niso bili izvedljivi.

Odjemalec			Strežnik	Datoteke			
				0kB	25kB	50kB	100kB
HTTP protokol				1	2	3	4
SSL protokol	**ECC kriptosistem**	**160 bitni ključ**	asio	5	6	7	8
			as	9	10	11	12
		224 bitni ključ	asio	13	14	15	16
			as	17	18	19	20
		256 bitni ključ	asio	21	22	23	24
			as	25	26	27	28
		384 bitni ključ	asio	29	30	31	32
			as	33	34	35	36
		521 bitni ključ	asio	37	38	39	40
			as	41	42	43	44
	RSA kriptosistem	**1024 bitni ključ**	asio	45	46	47	48
			as	49	50	51	52
		2048 bitni ključ	asio	53	54	55	56
			as	57	58	59	60
		3072 bitni ključ	asio	61	62	63	64
			as	65	66	67	68
		4096 bitni ključ	asio	69	70	71	72
			as	73	74	75	76
		7680 bitni ključ	asio	77	78	79	80
			as	81	82	83	84

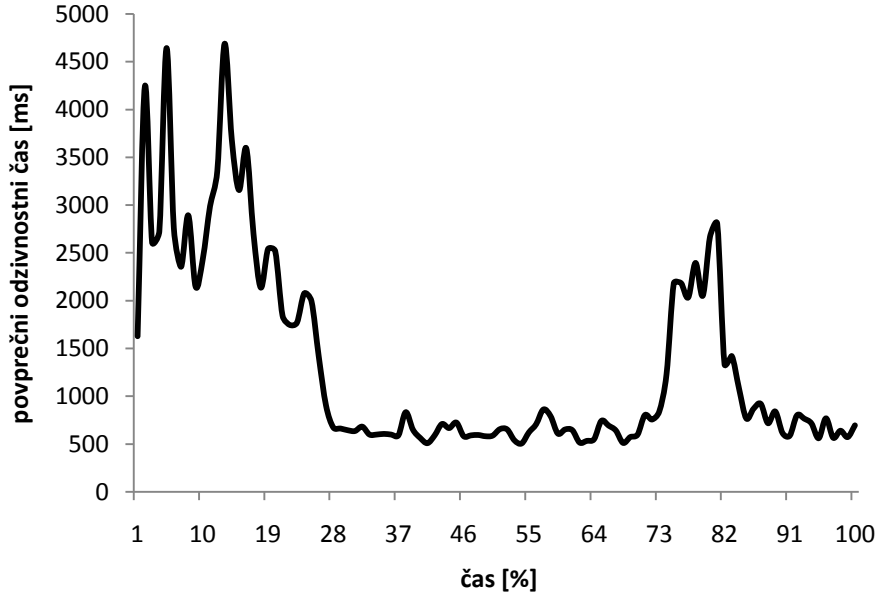
Tabela 4: Seznam testov

Parametri

Vsak test je trajal pet minut, v katerem je devet računalnikov izvedlo po sedemsto zahtev, torej skupaj 6300 zahtev oziroma 21 zahtev na sekundo po enakomerni časovni porazdelitvi. Pri večjem številu zahtev (36 zahtev na sekundo) so postali odzivni časi bolj neenakomerno porazdeljeni in s tem dobljeni rezultati vprašljivi, v ekstremnem primeru pa je postal strežnik neodziven, zahteve pa so se pogosto prekinile zaradi časovne prekinitve (time out).



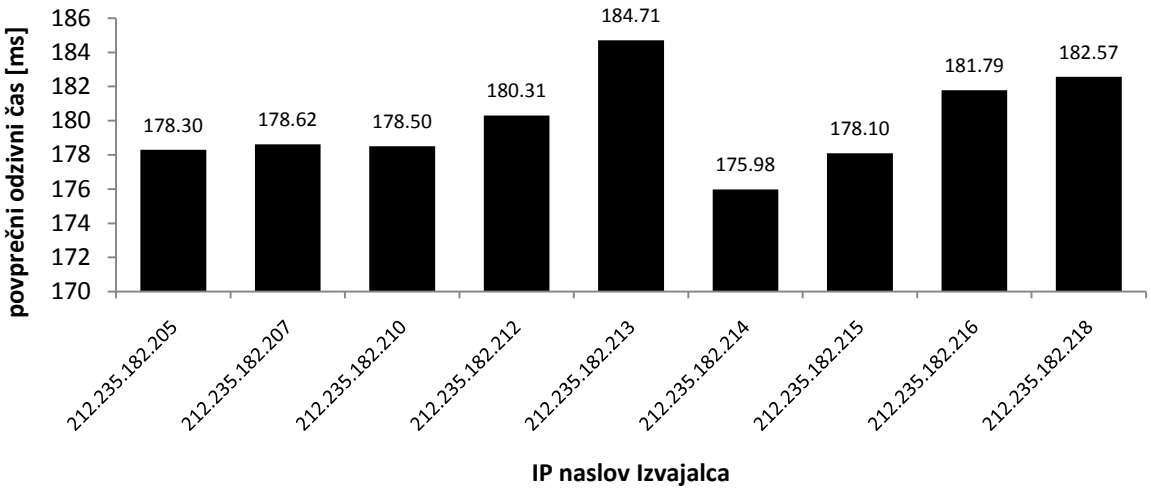
Slika 11: Povprečni odzivni čas v milisekundah testa 44 pri obremenitvi strežnika 21 zahtev na sekundo



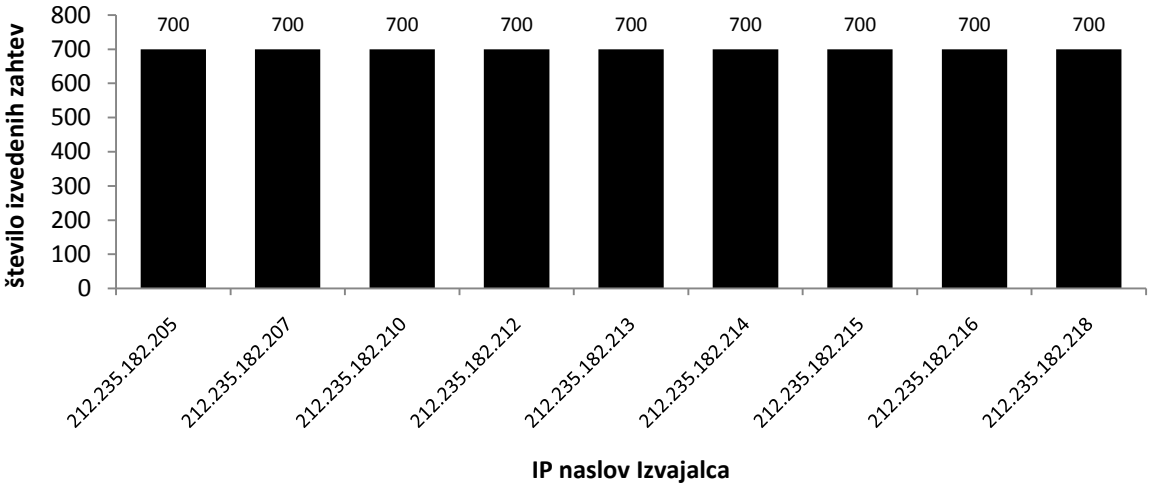
Slika 12: Povprečni odzivni čas testa 44 pri obremenitvi strežnika 36 zahtev na sekundo

Pri izvajanju zahtev so bila uporabljena izključno veljavna digitalna potrdila. Testiral se je tako HTTP kot HTTPS protokol (z uporabo RSA in EC digitalnih potrdil). Datoteke, ki so se prenašale, so bile velikosti 0kB, 25kB, 50kB in 100kB in so vsebovale naključno generirane nize. Prazna datoteka z dolžino 0kB je bila uporabljena zato, ker se v rezultatih odraža povprečni čas izvajanja samega rokovanja, saj se v tem primeru druga faza SSL protokola (simetrično šifrirana izmenjava podatkov) sploh ne izvede. Ostale datoteke so vključene v

testiranje zato, da ponazorijo razmerje med simetrično šifrirano komunikacijo in samim rokovanjem. Za vsako dolžino ključev je bilo izdelano sto veljavnih digitalnih potrdil za vsakega izmed kriptosistemov, ki so jih Izvajalci uporabljali pri testiranju HTTPS protokola.



Slika 13: Primer povprečnega odzivnega časa posameznega Izvajalca testa 44 pri obremenitvi strežnika 21 zahtev na sekundo



Slika 14: Število izvedenih zahtev posameznega Izvajalca pri testu 44

Rezultati

Rezultat posameznega testa predstavlja povprečni odzivni čas, merjen v milisekundah.

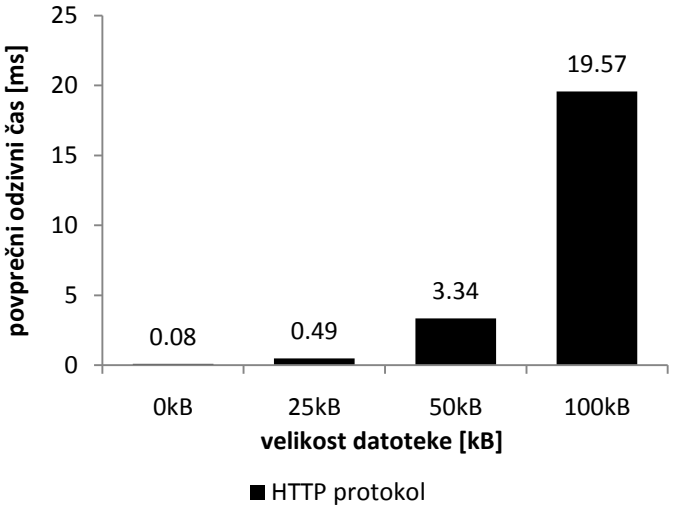
Odjemalec			Strežnik	Datoteke			
				0kB	25kB	50kB	100kB
HTTP protokol				0,08	0,49	3,34	19,57
SSL protokol	**ECC kriptosistem**	**160 bitni ključ**	asio				
			as				
		224 bitni ključ	asio				
			as				
		256 bitni ključ	asio	181,50	199,11	210,84	232,31
			as	51,73	54,43	63,14	75,97
		384 bitni ključ	asio	220,33	233,46	241,50	264,33
			as	50,89	57,47	68,27	79,52
		521 bitni ključ	asio	445,33	458,70	480,36	529,18
			as	132,60	147,56	157,82	179,88
	RSA kriptosistem	**1024 bitni ključ**	asio	17,29	30,53	32,92	47,38
			as	2,24	18,48	19,15	35,80
		2048 bitni ključ	asio	59,97	65,17	71,73	85,11
			as	17,26	33,09	34,12	47,14
		3072 bitni ključ	asio	142,92	158,46	165,36	184,96
			as	51,64	57,16	67,26	81,77
		4096 bitni ključ	asio	354,79	368,32	384,79	419,03
			as	133,70	148,60	158,67	190,78
		7680 bitni ključ	asio				
			as				

Tabela 5: Povprečni odzivni čas obremenitvenih testov (v milisekundah) (za primerljive dolžine ključev glej poglavje 5 za)

Analiza rezultatov

HTTP protokol

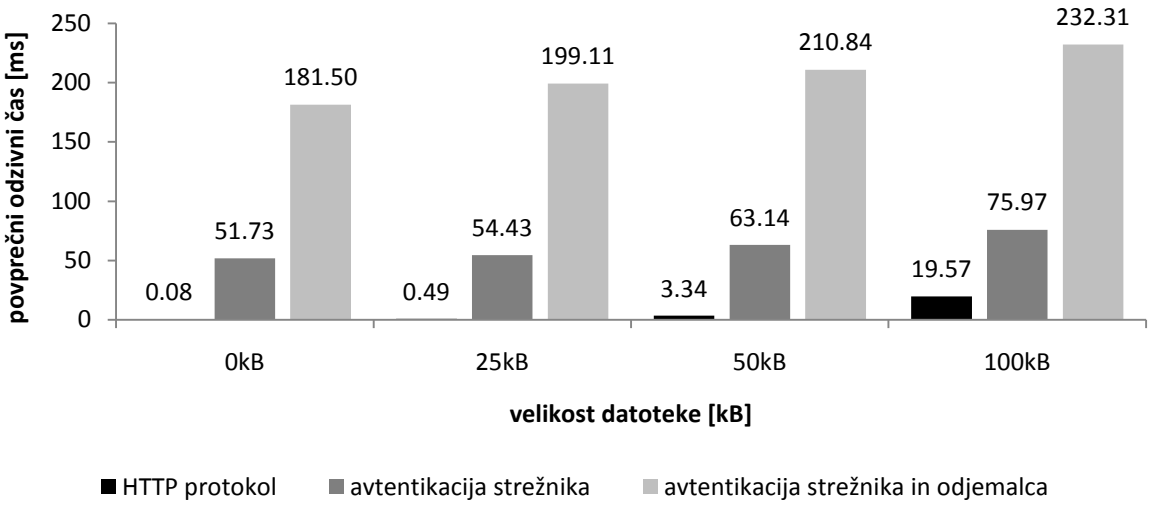
HTTP protokol je bil uporabljen za primerjavo s protokolom HTTPS. Kot pričakovano, se je pri vseh velikostih datotek izkazal za najhitrejšega, saj odgovor na posamezno zahtevo ne vsebuje ne rokovanja, ne simetričnega šifriranja podatkov, temveč le vzpostavitev povezave in prenos podatkov.



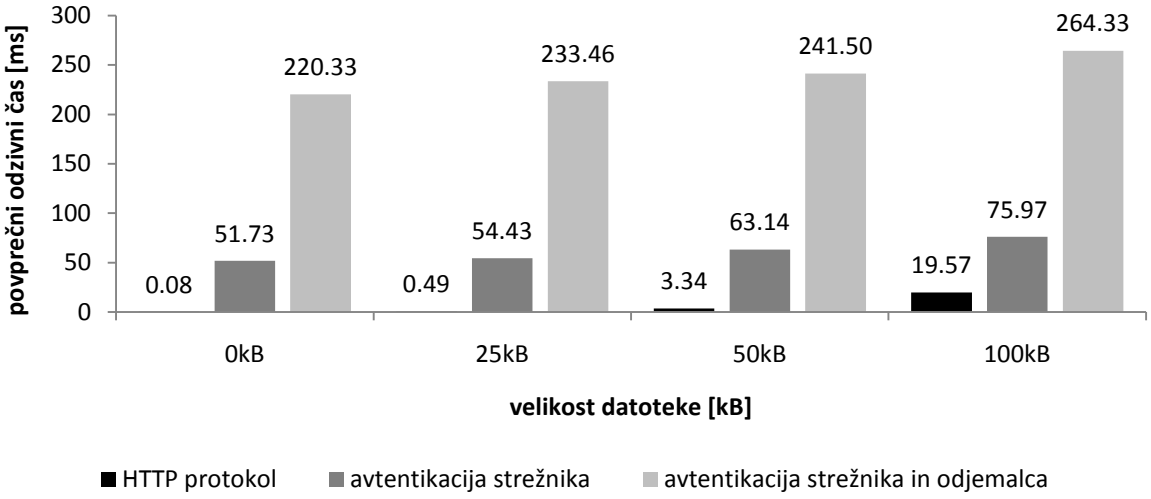
Slika 15: Poprečni odzivni čas HTTP protokola

HTTPS protokol z uporabo EC digitalnih potrdil

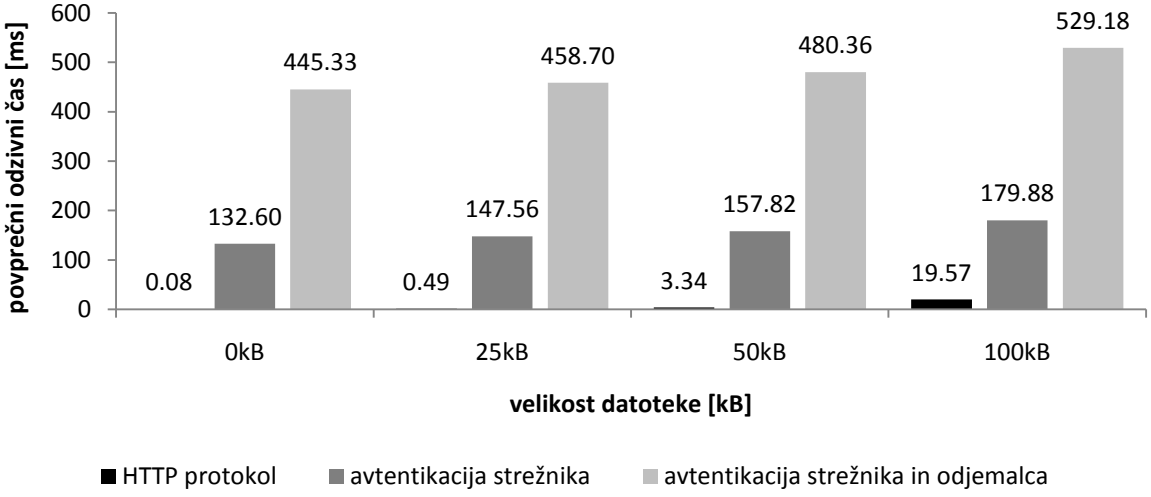
Protokol HTTPS z uporabo EC digitalnih potrdil je vseboval teste z dolžinami ključev 256, 384 in 521 bitov. Testirana sta bila dva načina delovanja: avtentikacija strežnika ter avtentikacija strežnika in odjemalca na štirih datotekah različnih dolžin: 0kB, 25kB, 50kB in 100kB.



Slika 16: Povprečni odzivni čas HTTPS protokola z uporabo 256-bitne dolžine ključev EC kriptosistema

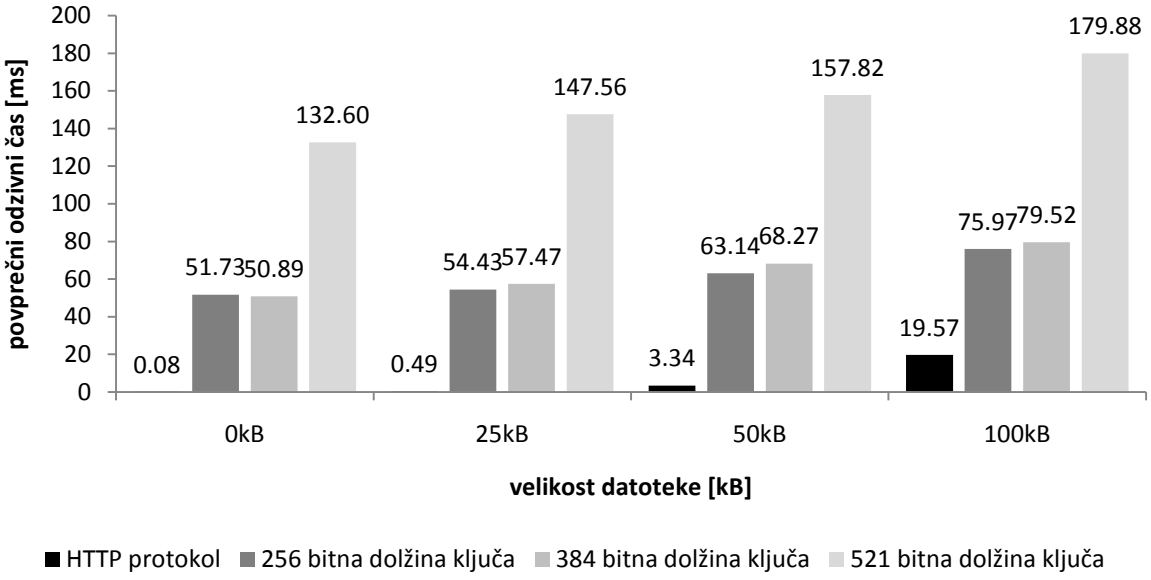


Slika 17: Poprečni odzivni čas HTTPS protokola z uporabo 384-bitne dolžine ključev EC kriptosistema

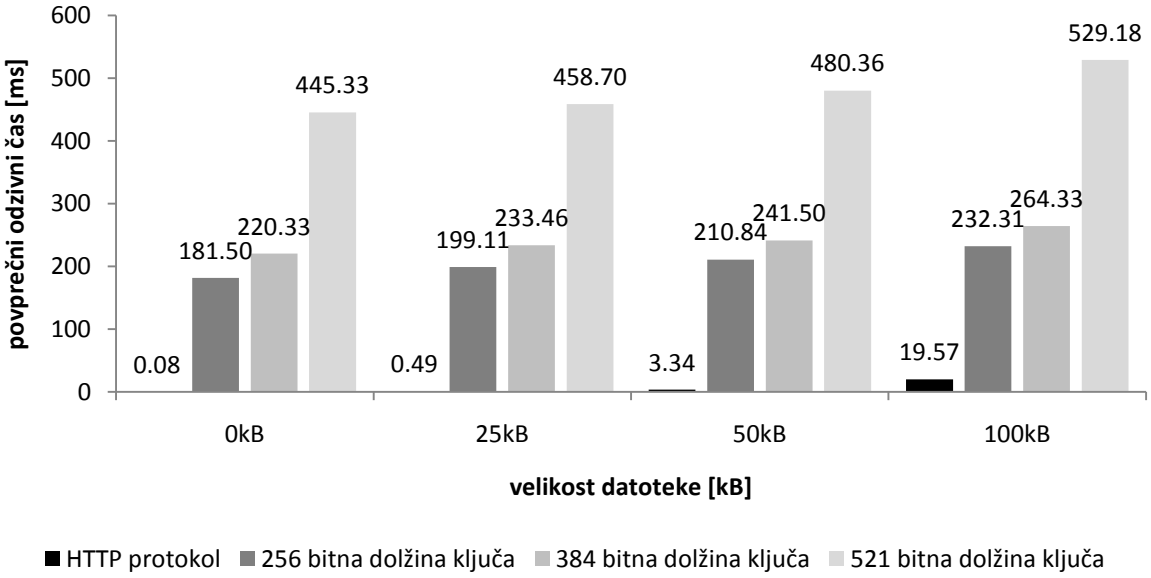


Slika 18: Poprečni odzivni čas HTTPS protokola z uporabo 521-bitne dolžine ključev EC kriptosistema

Kot je mogoče videti iz primerjalnih grafov, je dolžina rokovanja bistveno daljša od prenosa datoteke pri HTTP protokolu (datoteka velikosti 0kB).

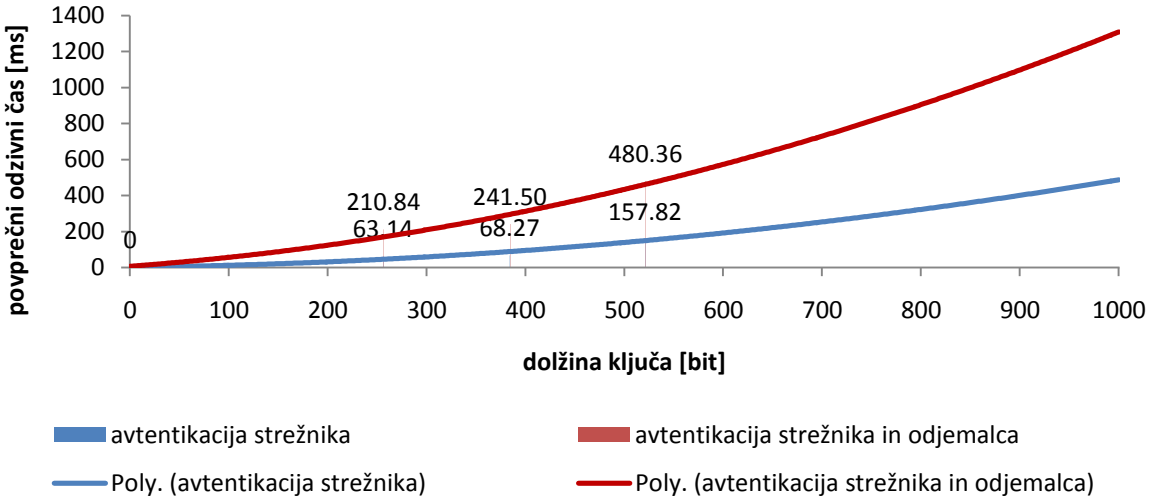


Slika 19: Primerjava povprečnih odzivnih časov HTTPS protokola z uporabo EC kriptosistema in avtentikacijo strežnika (as)



Slika 20: Primerjava povprečnih odzivnih časov HTTPS protokola z uporabo EC kriptosistema in avtentikacijo strežnika in odjemalca (asio)

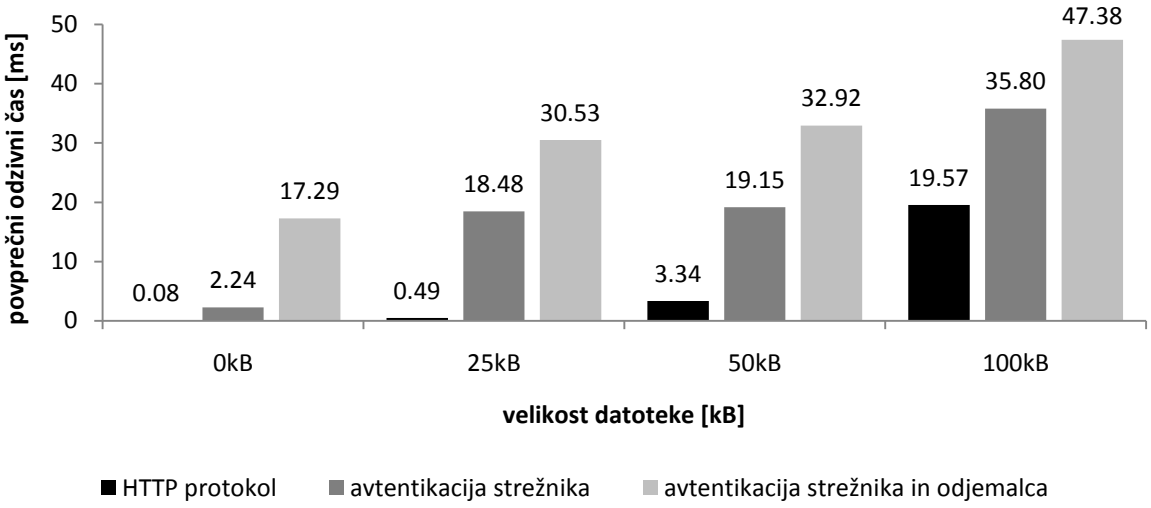
Z uporabo interpolacije lahko napovemo, kako bi se odzivni čas povečeval pri večjih dolžinah ključev. Uporabljeni so bili podatki za 50kB velike datoteke. Izkaže se, da obe krivulji naraščata počasi, kar pomeni, da je EC kriptosistem odlična izbira, ko se pojavi potreba po večji stopnji varnosti.



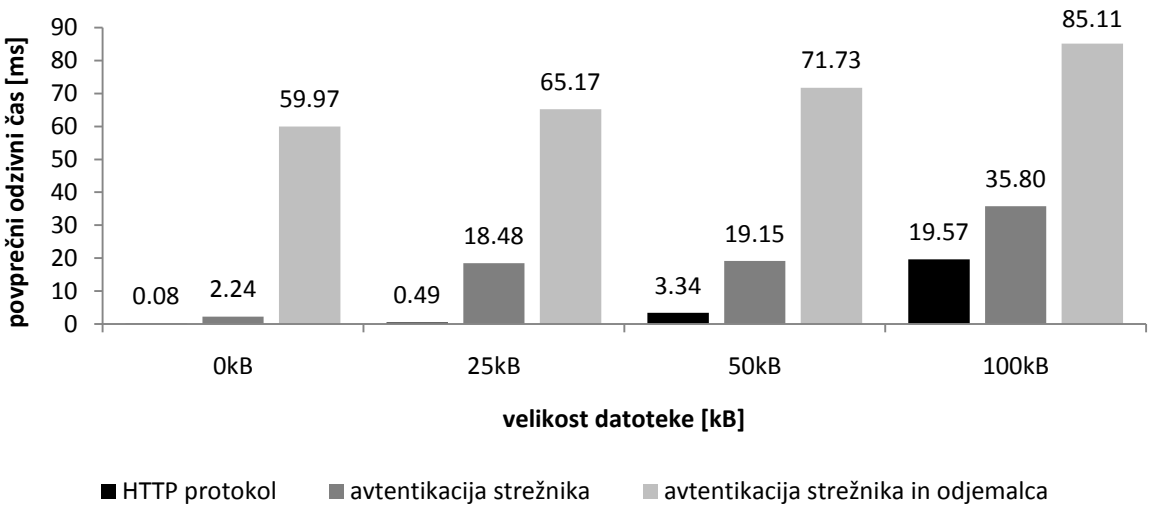
Slika 21: Pričakovani povprečni odzivni čas pri daljših dolžinah ključev EC kriptosistema

HTTPS protokol z uporabo RSA digitalnih potrdil

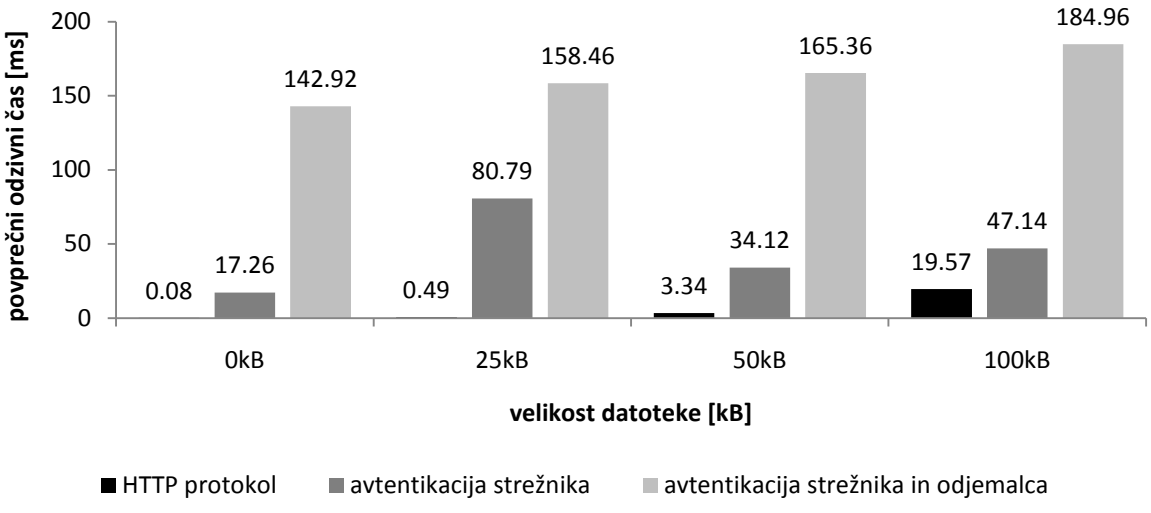
HTTPS protokol z uporabo RSA digitalnih potrdil je vseboval teste z dolžinami ključev 1024, 2048, 3072 in 4096 bitov. Testirana sta bila dva načina delovanja: avtentikacija strežnika ter avtentikacija strežnika in odjemalca ter štirimi datotekami različnih dolžin: 0kB, 25kB, 50kB in 100kB.



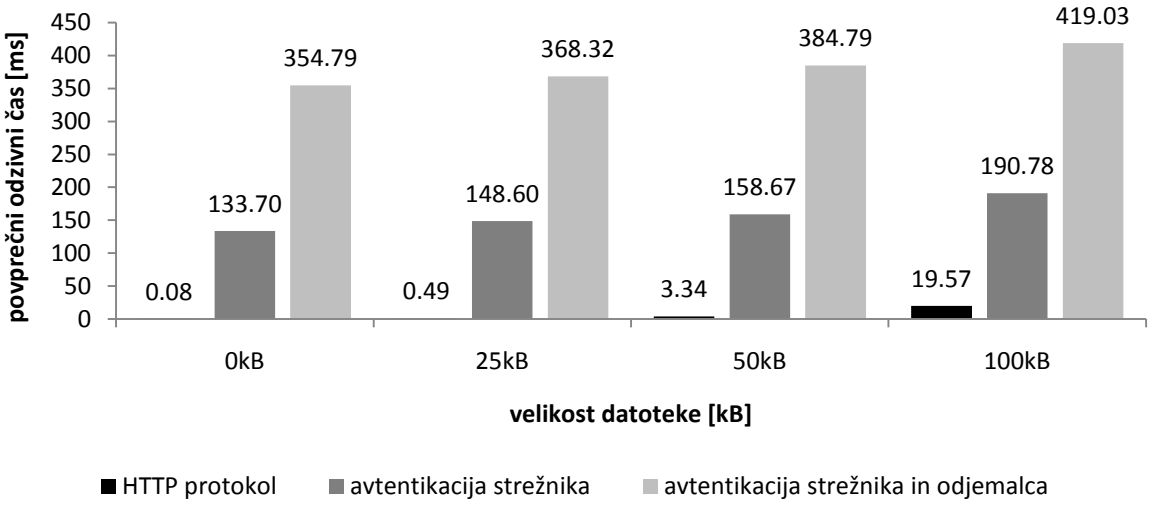
Slika 22: Poprečni odzivni čas HTTPS protokola z uporabo 1024-bitne dolžine ključev RSA kriptosistema



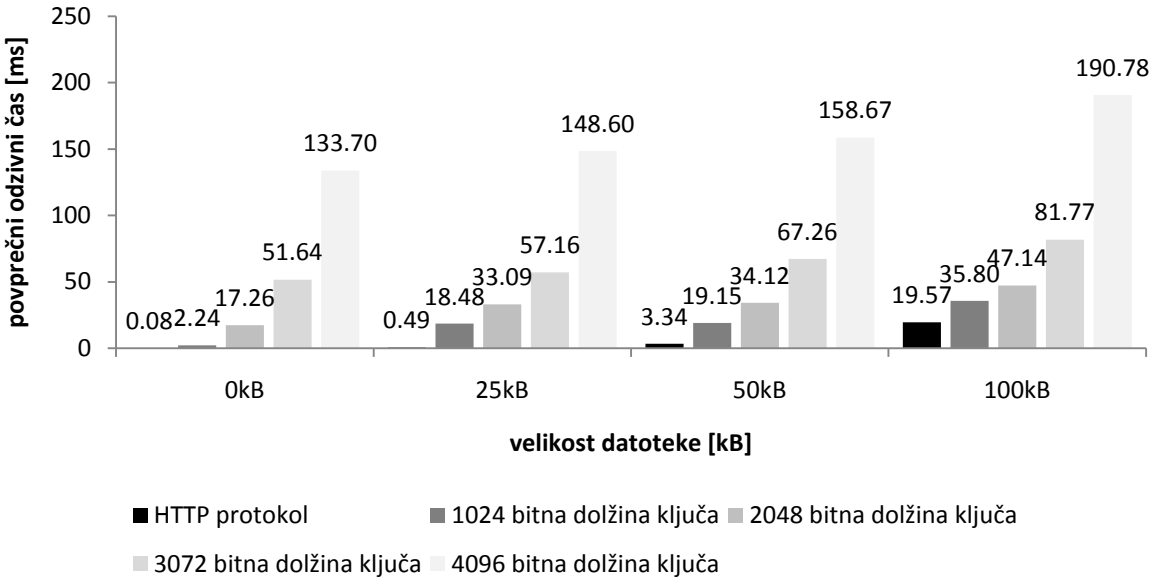
Slika 23: Poprečni odzivni čas HTTPS protokola z uporabo 2048-bitne dolžine ključev RSA kriptosistema



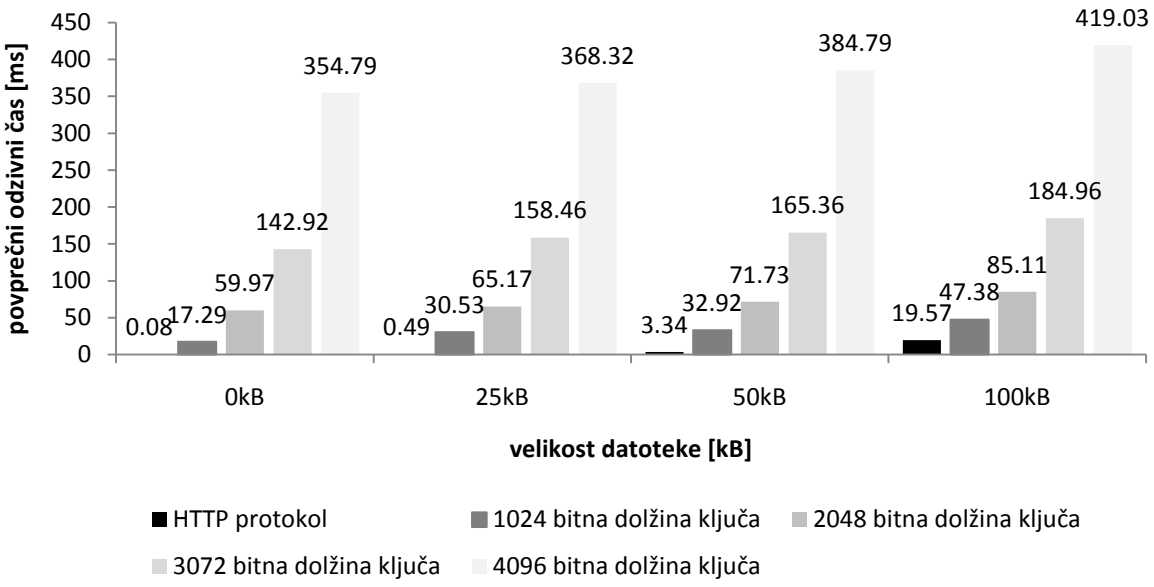
Slika 24: Poprečni odzivni čas HTTPS protokola z uporabo 3072-bitne dolžine ključev RSA kriptosistema



Slika 25: Poprečni odzivni čas HTTPS protokola z uporabo 4096-bitne dolžine ključev RSA kriptosistema

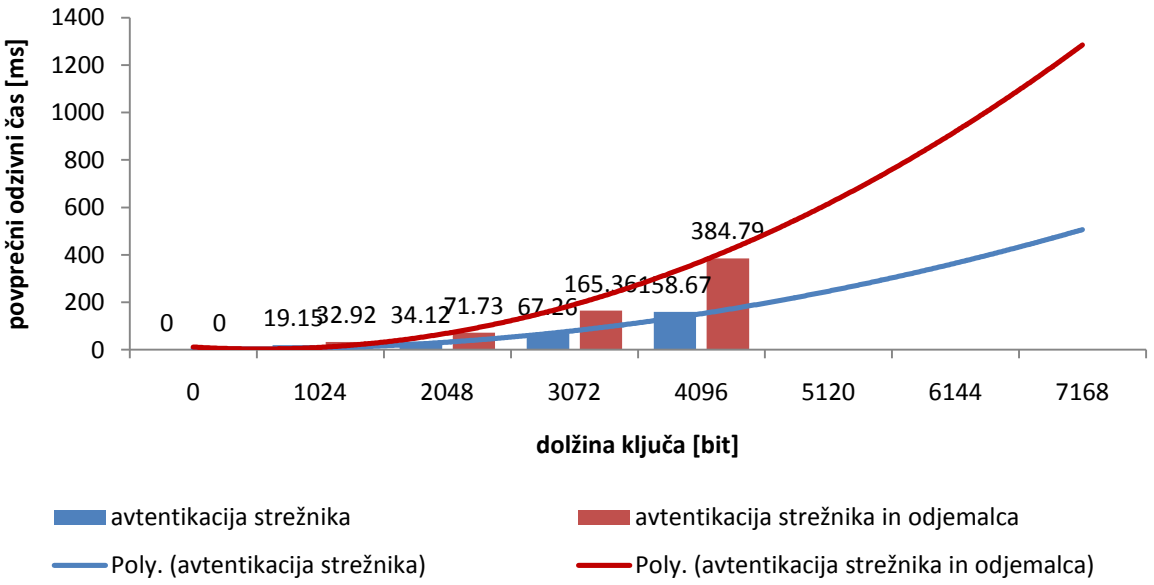


Slika 26: Primerjava povprečnih odzivnih časov HTTPS protokola z uporabo RSA kriptosistema in avtentikacijo strežnika (as)



Slika 27: Primerjava povprečnih odzivnih časov HTTPS protokola z uporabo RSA kriptosistema in avtentikacijo strežnika in odjemalca (asio)

Z uporabo interpolacije lahko napovemo, kako bi se odzivni čas povečeval pri večjih dolžinah ključev. Uporabljeni so bili podatki 50kB velike datoteke. Izkaže se, da obe krivulji naraščata bistveno bolj strmo kot pri EC kriptosistemu, vendar pa je RSA pri krajših dolžinah ključa hitrejši kot EC. Torej, kjer je potreba po nižji stopnji varnost, je RSA bolj primeren kot EC kriptosistem, kar bi lahko bila tudi posledica slabše implementacije EC.



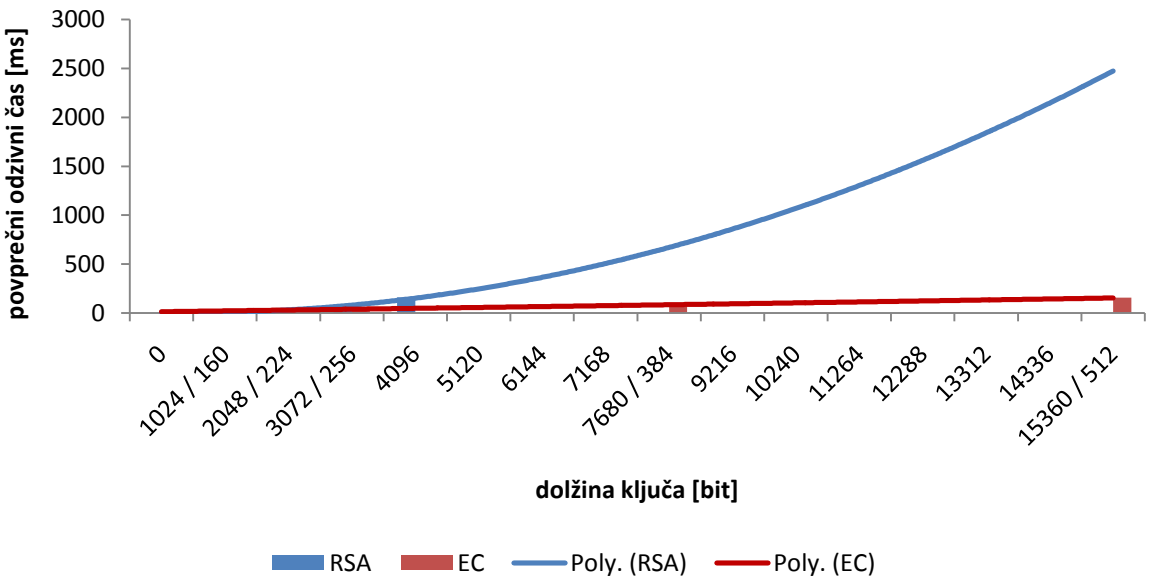
Slika 28: Pričakovani povprečni odzivni čas pri daljših dolžinah ključev RSA kriptosistema

Primerjava HTTPS protokola z uporabo EC in RSA digitalnih potrdil

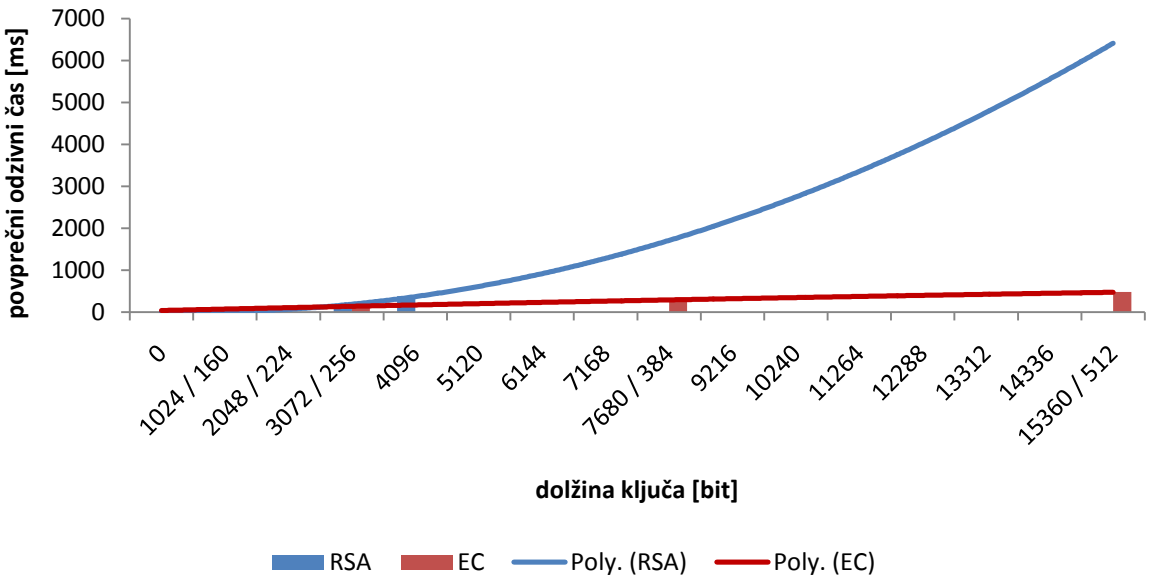
Ker ni bilo mogoče primerjati povprečnega odzivnega časa pri isti stopnji varnosti obeh kriptosistemov (z izjemo 3072 bitne dolžine ključa pri RSA kriptosistemu in 256 bitne dolžine ključa pri EC kriptosistemu), je bilo za primerjavo rezultatov potrebno uporabiti interpolacijo.

Za interpolacijsko krivuljo je bil izbran polinom druge stopnje, ki se izmed vseh preizkušenih krivulj (premica, polinomi višjih stopenj, eksponentna krivulja, …) najbolje prilega izmerjenim rezultatom.

Če primerjamo odzivni čas obeh kriptosistemov glede na dolžino ključev (z ozirom na enako stopnjo varnosti) obeh kriptosistemov se izkaže, da z naraščanjem dolžine ključa povprečni odzivni čas pri uporabi RSA digitalnih potrdil narašča hitreje kot pri uporabi EC digitalnih potrdil.



Slika 29: Primerjava EC in RSA kriptosistema pri avtentikaciji strežnika



Slika 30: Primerjava ECC in RSA kriptosistema pri avtentikaciji strežnika in odjemalca

Žal pa ni bilo mogoče preizkusiti hitrosti delovanja krajših dolžin ključev EC kriptosistema. Glede na prejete rezultate je mogoče sklepati, da se pri krajših dolžinah ključev EC kriptosistem izkaže za počasnejšega kot RSA, kar je sicer v nasprotju z pričakovanji, vendar pa je ta predpostavka smiselna, glede na to da so se proizvajalci programske opreme (Microsoft, Mozilla) odločili, da bodo podprli digitalna potrdila EC kriptosistema z dolžinami ključev 256, 384 in 521, ne pa tudi 160, 192 in 224 bitov.

RSA kriptosistem se izkaže za hitrejšega vse tja nekje do dolžine ključa nekaj več kot 3072 bitov, medtem ko je ECC kriptosistem hitrejši pri dolžini ključev nekaj več kot 256 bitov

Poglavje 9

Zaključek

Kot je bilo pričakovati, je SSL protokol z uporabo EC digitalnih potrdil resnično hitrejši od uporabe RSA digitalnih potrdil, vendar šele pri daljših dolžinah ključev. To gre sklepati tudi po programski opremi, katere proizvajalci so pred kratkim začeli vključevati podporo EC kriptosistemu le pri dolžinah od 256 bitov naprej. Pri krajših dolžinah ključev pa se za hitrejšega izkaže RSA kriptosistem. Pričakovati je torej, da bo EC kriptosistem izpodrinil RSA kriptosistem, vendar še ne tako kmalu, saj so trenutno najbolj razširjena digitalna potrdila z 1024 bitno dolžino ključev.

Kadar se pojavi potreba po močnejši zaščiti podatkov, je priporočljiva uporaba EC kriptosistem, medtem ko je pri potrebi po hitri komunikaciji primernejša uporaba RSA kriptosistema.

Vsekakor pa so potrebne nadaljne raziskave, ki bi dale bolj natančne rezultate za primerjavo obeh kriptosistemov.

Literatura

[1] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: Applied cryptography, CRC Press, 1997

[2] About Apache HTTP server project, http://httpd.apache.org

[3] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang, Nils Gura, Hans Eberle: Speeding up Secure Web Transactions using Elliptic Curve Cryptography (ECC), Sun Microsystems, Inc

[4] Scott Vanstone: Deployments of Elliptic Curve Criptography, University of Waterloo, 2005

[5] G.V.S. Raju in Rehan Akbani, Elliptic Curve Cryptosystem and its Applications, The University of Texas at San Antonio

[6] NSA: Fact Sheet NSA Suite B Cryptography, http://www.nsa.gov/ia/industry/crypto_suite_b.cfm?MenuID=10.2.7, 2008

[7] Elliptic Curve Criptography, An Implementation Tutorial, Tata Elxsi

[8] Ivan Vidav, Eliptične krivulje in eliptične funkcije, Društvo matematikov, fizikov in astronomov Slovenije, 1991

[9] Zhaohui Cheng, Simple Tutorial on Elliptic Curve Cryptography, 2004

[10] RSA Laboratories: What is Diffie-Hellman?, http://www.rsa.com/rsalabs/node.asp?id=2248

[11] Scott Vanstone, Deployments of elliptic curve criptography, University of Waterloo

[12] Francisco Rodríguez-Henríquez, Carlos E. López-Peza, Miguel Angel León-Chávez, Comparative Performance Analysis of Public-Key Cryptographic Operations in the WTLS Handshake Protocol, Benemérita Universidad Autónoma de Puebla

Izjava

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Aleksandra Jurišića ter soglašam, da je diplomsko delo v elektronski obliki dostopno v zbirki 'Dela FRI'.

Ljubljana, september 2008 Aljaž Simonič