Kriptografsko glasovanje u sustavu Helios

Kovačević, Antun

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:271345

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Image not found or type unknown

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

zir.nsk.hr

Image not found or type unknown

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANTUN KOVAČEVIĆ

KRIPTOGRAFSKO GLASOVANJE U SUSTAVU HELIOS

Diplomski rad

U Puli, 2018. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANTUN KOVAČEVIĆ

KRIPTOGRAFSKO GLASOVANJE U SUSTAVU HELIOS

Diplomski rad

JMBAG: 0303033052, redoviti student

Studijski smjer: Informatika

Predmet: Kriptografija

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Organizacija i informatika

Mentor: *izv.prof.dr.sc.* Valter Boljunčić
Sumentor: *dr.sc.* Siniša Miličić

U Puli, srpnja 2018. godine



IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Antun Kovačević, kandidat za magistra **INFORMATIKE** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, **Antun Kovačević** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom **"KRIPTOGRAFSKO GLASOVANJE U SUSTAVU HELIOS"** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____

Potpis

_____

Sadržaj

1. Uvod *1*

2. Kriptografija *3*

2.1. Alisa i Bob **3**

2.2. Simetrična kriptografija **4**

2.1.1 Data Encryption Standard (DES) 11
2.1.2 Triple Data Encryption Standard (Triple DES ili 3DES) 13
2.1.3 Advanced Encryption Standard (AES) 13
2.1.3 Usporedba algoritama 20

2.3. Asimetrična kriptografija **22**

2.2.1 RSA (Rivest–Shamir–Adleman) kriptosustav 24
2.2.2 Kriposustav ElGamal 26

2.4. Siguran hash algoritam (SHA) **29**

3. Arhitektura sustava Helios *31*

3.1. Tehnologije **31**

3.1.1 Programski jezici 31
3.1.2 Django 36
3.1.3 PostgreSQL 46
3.1.4 RabbitMQ 50
3.1.5 Celery 50
3.1.6 OAuth2 50

3.3 Dijagrami **52**

3.3.1 Dijagram slijeda obrazaca uporabe 53
3.3.2 Dijagram stanja 54
3.3.3 Kontekstualni dijagram 54

3.4 Relacijski model podataka **55**

4. Korisničko korištenje Heliosa *57*

4.1. Upravljanje glasačima **58**

4.2. Priprema pitanja/listića **58**

4.3. Zamrzavanje izbora **59**

4.4. URI-ovi i primjeri JSON podataka **62**

4.4.1 Revizija 62
4.4.2 Kriptomehanizam 63
4.4.3 Glasač 64
4.4.4 Zaštita privatnosti glasača 64
4.4.5 Glasovanje 65

4.4.6 Izbori 66
4.4.7 Otisak prsta izbora (eng. fingerprint) 68
4.4.8 Glas 69
4.4.9 Revizija izbora i dokazi bez znanja 69
4.4.10 Dokaz 70
4.4.11 Postupak revizije glasovanja 71
4.4.12 Rezultat 72
5. Garancije sustava Helios 73
5.1 Provjera jedinstvenog glasačkog listića 73
5.2 Revizija jednog glasačkog listića 76
5.3 Provjera kompletnog izbora 77
6. Instalacija sustava Helios na server (VPS) 80
6.1 Instalacija Helios sustava na Linux operacijski sustav 80
6.2 Konfiguracija Apache servera za Django web framework za produkciju 83
6.3 Helios autentifikacija 88
6.3.1 Podešavanje Google oauth2 za prijavu u sustav Helios 88
7. Potencijalni korisnici sustava 90
8. Usporedba Heliosa s Prêt-à-Voter (PaV) glasačkim sustavom 91
9. Zaključak 98
Literatura 99
Sažetak 111
Summary 111

1. Uvod

Predmet istraživanja ovog diplomskog rada je analiza principa rada kriptografskog sustava Helios. Udruge, zadruge, srednja i velika poduzeća gdje odluke ne donosi jedan čovjek, imaju potrebu koristiti se sustavom za glasovanje. Često na skupovima ne mogu prisustvovati svi članovi koji imaju pravo glasa, zbog čega im je potreban javni digitalni sustav pomoću kojeg mogu glasovati. Takav sustav mora biti siguran, otporan na napade i manipulaciju rezultatima glasovanja. Upravo zbog toga na primjeru Heliosa opisana je problematika glasovanja s kriptografskog aspekta.

Helios je E2E sustav što znači da je sustav neovisnog nadzora koji pohranjuje kriptirane glasove te glasaču *„potvrđuje kako je njegov glas preuzet i pohranjen, osigurava da su sve glasove dali vjerodostojni glasači i omogućuje naknadna provjera rezultata glasovanja.“* (CARNet CERT, 2007.)

Cilj ovog diplomskog rada je opisati princip rada sustava Helios, analizirari tehnologije u kojem je rađen te razumjeti korištene kriptografske algoritme. Na temelju toga, svrha ovog rada je primijeniti postojeća saznanja, uočiti nedostatke te daljnjim istraživanjima poboljšati sustav glasovanja.

U drugom poglavlju pod imenom “Kriptografija” ukratko je spomenuta povijest kriptografije nakon čega je objašnjena simetrična i asimetrična kriptografija. Od simetrične kriptografije objasnit će se DES i AES, a od asimetrične kriptografije RSA i ElGamal koji se koristi u Heliosu te i *hash* algoritam SHA. Treće poglavlje, “Arhitektura sustava Helios”, opisuje tehnologije u kojima je izrađen Helios, pa tako to poglavlje obuhvaća i opisuje programske jezike, bazu podataka i *web framework* Django. Uz tehnologije prikazani su dijagrami te relacijski model baze podataka sustava Helios. Četvrto poglavlje “Korisničko korištenje Heliosa” opisuje princip rada sustava Helios, od početnih koraka kako se glasuje pa do logike kako taj sustav funkcionira.

Peto poglavlje “Garancija sustava Helios” prikazat će princip provjere glasačkih listića i provjeru cjelokupnog izbornog procesa. Šesto poglavlje “Instalacija sustava Helios“ na server (VPS)” sustavno i temeljito opisuje način na koji se pravilno instalira sustav Helios na lokalni i na produkcijski poslužitelj. U sedmom poglavlju “Potencijalni korisnici sustava”

navedeno je za koje skupine ljudi je Helios namijenjen. Osmo poglavlje “Usporedba Heliosa s Prêt-à-Voter (PaV) glasačkim sustavom” prikazati će i opisati razlike između njih te razmotriti kome su namijenjeni i koji su njihovi nedostaci.

2. Kriptografija

U ovom suvremenom informacijskom vremenu u kojem se svake sekunde razmjene milijarde informacija i bankovnih transakcija putem interneta, nezamislivo bi bilo da te informacije nisu dobro zaštićene, odnosno šifrirane.

Šifriranje informacija započelo je u ranoj povijesti. Rimski vojskovođa Gaj Julije Cezar koristio je metodu šifriranja koja stoljećima nije bila probijena. Svojim generalima slao je poruke šifrirane pomakom u abecedi za 3 slova, A→D, B→E, Z→C. Ova vrsta jednostavnog šifriranja koristila se u vojne i poslovne svrhe.

Nakon Cezarove šifre, bilo je još načina šifriranja, jedan od njih je stroj Enigma koji se koristio u drugom svjetskom ratu, ali i on je probijen. Tehnologija za šifriranje teksta s Enigminim algoritmom može se izvesti ručno, mehanički, elektromehanički ili digitalno, a probijanje statistički, elektromehanički ili digitalno. U moderno doba pojavila s potreba za novim i jačim kriptosustavima, da zadovolje potrebe raznih oblika sigurne i tajne komunikacije. Aktualne vrste kriptosustava su simetrične i asimetrične šifre. Jedan od simetričnih blok šifri je AES kriptosustav koji je odobren 2001. godine od strane američkog nacionalnog instituta za standarde i tehnologiju (NIST). AES je simetrični kriptosustav koji koristi isti ključ za šifriranje i dešifriranje podataka što znači da se ključevi moraju pažljivo razmjenjivati jer isti ključ imaju primatelj i pošiljatelj. Siguran način razmjene ključeva zasnovan na diskretnom logaritmu ponudili su Whitfield Diffie i Martin Hellman 1976. godine. Kvalitetniji i pouzdaniji, ali i sporiji (zahtijeva više procesorske snage) su asimetrični kriptosustavi kod kojih su potrebna dva ključa, javni i privatni (Dujella, 2007).

2.1. Alisa i Bob

Alisa i Bob su izmišljeni likovi u kriptografskim primjerima i objašnjenjima koji međusobno razgovaraju koristeći razne kriptosustave. Uz Alisu i Boba također postoji, Eva koja ih pasivno prisluškuje te zlonamjerni napadač Mallory i drugi.

Alisa i Bob prvi puta su spomenuti u Rivestovom, Shamirovom i Adlemanovom primjeru komunikacije iz 1978. godine, u radu "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". Korišteni su sa svrhom objašnjenja funkcioniranja kriptografije javnog ključa, a od tada su postali široko korišteni u drugim znanostima i inženjerskim domenama (Doctorow, 2017).

2.2. Simetrična kriptografija

Simetrične šifre koriste iste ključeve za šifriranje i dešifriranje poruke. Dizajnirane su s ciljem da se pomoću njih efikasno šifriraju velike poruke u stvarnom vremenu. Efikasno i brzo šifriranje možemo postići korištenjem FPGA (Field Programmable Gate Arrays). FPGA je uređaj koji se sastoji od poluvodičkih elemenata temeljenih na matrici konfigurabilnih logičkih blokova povezanih preko programabilnih međusobnih veza. FPGA se može reprogramirati na željeni zahtjev, pa se tako može programirati i AES algoritam. Postiže veliku brzinu jer se koristi isključivo u svrhu za koju je programiran od strane programera (OpenCores: AES, 2015).

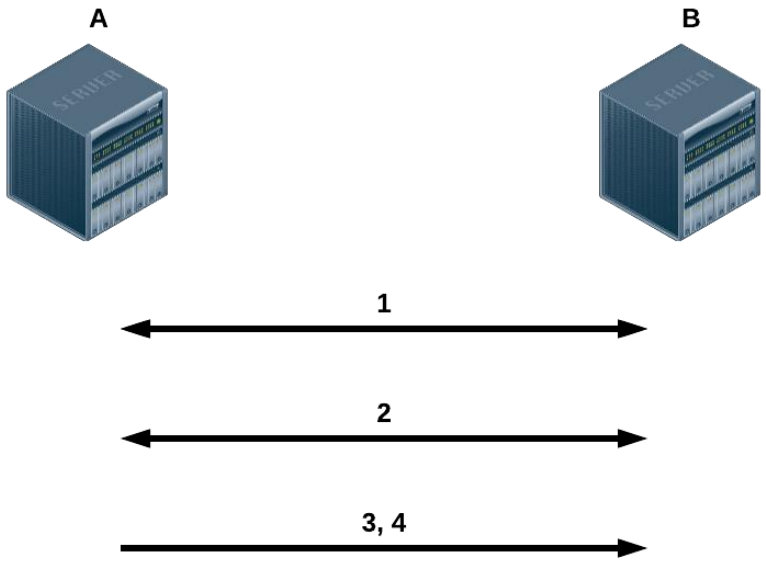
Simetrični kriptosustavi iznimno su pogodni za situacije u kojima nema potrebe za razmjenom ključeva, na primjer kod šifriranja vlastitih datoteka. Kod korištenja šifri za sigurnu komunikaciju dviju ili više strana nastaju problemi vezani uz upravljanje ključevima. Takvi se problemi mogu riješiti pomoću hibridnog pristupa koji uključuje upotrebu asimetričnih protokola. Simetrične šifre su osnova mnogih kriptografskih sustava i često se koriste s drugim kriptografskim mehanizmima pomoću kojih nadoknađuju svoje nedostatke.

Simetrično šifriranje može raditi u blok ili u protočnom načinu. Uvijek je moguće prebacivati protočni način u blok način i obrnuto. U blok načinu, simetrični algoritam dijeli ulaznu poruku u niz malih blokova fiksnih veličina, a zatim te blokove šifrira ili dešifrira jednog po jednog. U protočnom načinu, svaki znak ulazne poruke šifriran je zasebno.

Iako su blokovi potpuno šifrirani, poruka može biti ranjiva na neke trivijalne napade. Ako su dva identična bloka kriptirana na isti način, koristeći istu funkciju i ključ, odgovarajući šifrirani blokovi bili bi identični.

Načini rada unose dodatnu varijablu u funkciju koja određuje stanje izračuna. Stanje se mijenja tijekom postupka šifriranja ili dešifriranja i kombinira se sa sadržajem svakog bloka. Ovaj pristup ublažava probleme s identičnim blokovima. Vrijednost inicijalizacije dodatne varijable naziva se inicijalizacijskim vektorom (IV).

Na slici 1 prikazuje se postupak simetrične kriptografije. Čvor A i B najprije se slažu što se tiče tehnike šifriranja koja će se koristiti za šifriranje i dešifriranje priopćenih podataka. Tada se slažu oko tajnog ključa koji će oboje koristiti u vezi s tim. Nakon završetka dogovora o načina šifriranja, čvor A započinje slanje svojih podataka šifriranim zajedničkim ključem, a na drugoj strani čvor B koristi isti ključ za dešifriranje poruka.



1. A i B se slažu oko kriptosustava.
2. A i B se slažu oko ključa koji će se koristiti.
3. A šifrira poruke pomoću zajedničkog ključa
4. B dekriptira šifrirane poruke pomoću zajedničkog ključa

Slika 1. Postupak simetrične kriptografije (Al-Tamimi, 2006)

U daljnjem tekstu opisana su 4 AES načina šifriranja (ECB, CFB, OFB, CTR):

1. *Electronic Codebook* (ECB)

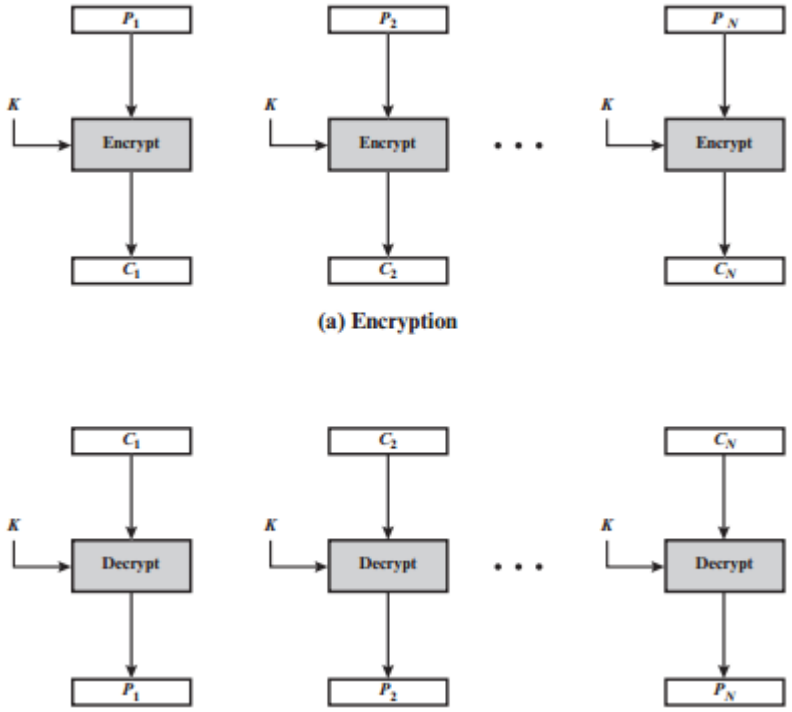
Poruka se dijeli na blokove, svaki blok beskonačnih bita kodira se neovisno pomoću istog ključa, odnosno svaki blok je zasebno šifriran. Koristi se za siguran prijenos pojedinačnih vrijednosti kao na primjer ključ za šifriranje. U nastavku su formule za šifriranje i dešifriranje u ECB modu, oznaka E je funkcija za šifriranje, D je funkcija za dešifriranje, C je šifrat, K je ključ i P je otvoreni tekst (Stallings, 2017).

Formula za šifriranje:

$$C_j = E(K, P_j) \quad j = 1, \ldots, N$$

Formula za dešifriranje:

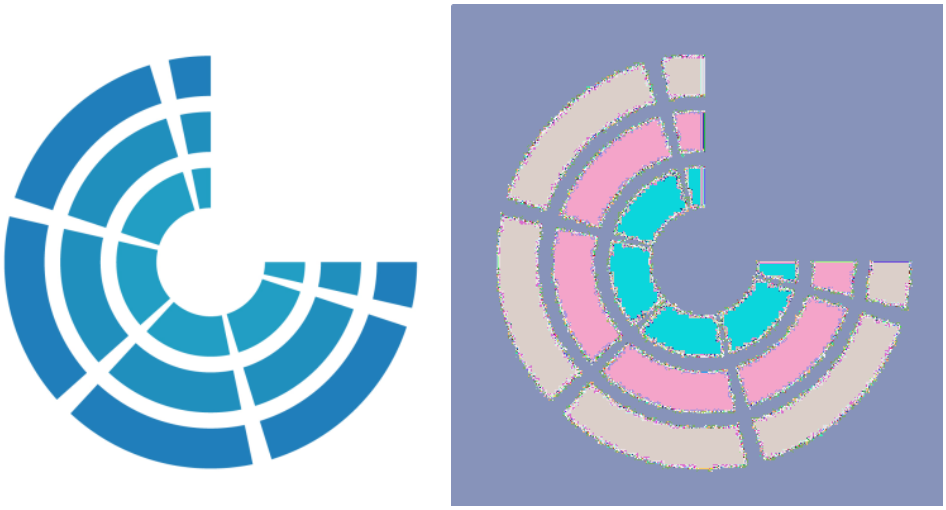
$$P_j = D(K, C_j) \quad j = 1, \ldots, N$$



Slika 2. Prikaz ECB načina rada (Stallings, 2017., str. 215.)

Na slici 2, u dijelu (a) prikazuje se kriptiranje Otvorenog teksta P_1 pomoću ključa K, rezultat je C_1, a u dijelu (b) prikazuje se obratna situacija dešifriranja. Oznaka C_1 je šifrirani tekst koji se dešifrira ključem K, a rezultat je otvoreni tekst P_1 (Stallings, 2017).

ECB načinom šifriranja nije preporučljivo šifrirati *bitmap* slike. Iako je svaki piksel kriptiran, slika se može prepoznati kao na slici 3.



Slika 3. Primjer ECB načina rada na slici loga FIPU

2. *Cipher Feedback* (CFB)

Ulazni bitovi obrađuju se odjednom. Prethodni je šifrirani tekst upotrijebljen kao ulaz u algoritam šifriranja za stvaranje pseudo slučajnog izlaza, koji je XOR-an s jasnim tekstom kako bi se proizvela sljedeća jedinica šifriranog teksta. CFB način šifriranja koristi se kod ovjera i prijenosa podataka opće namjene. Oznaka $\oplus$ označava „logičko ILI“ (eng. XOR), oznaka MSB (eng. *Most Significant Bit*) označava najznačajniji bit, a oznaka IV označava inicijalizacijski vektor (Stallings, 2017).

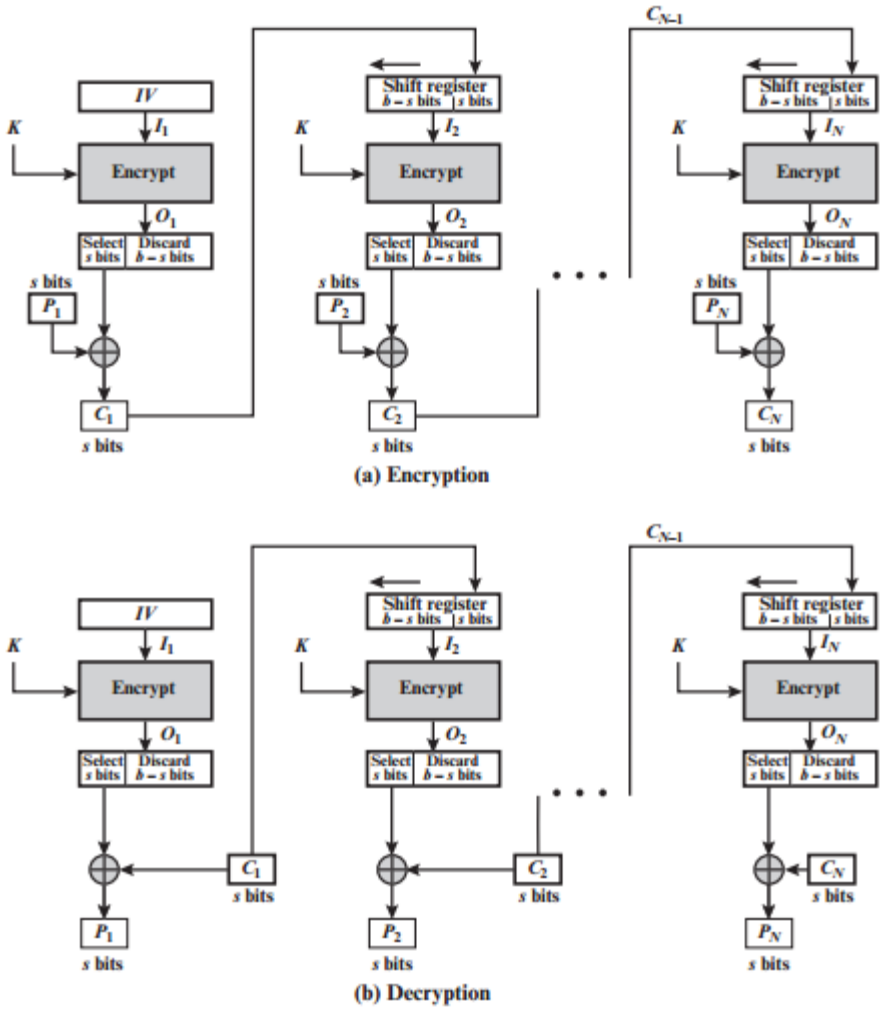
Formula za šifriranje:

$$C_1 = P_1 \oplus MSB_s[E(K, IV)]$$

Formula za dešifriranje:

$$P_1 = C_1 \oplus MSB_s[E(K, IV)]$$

Kod CFB šifriranja, svaki ulazni blok ovisi o kriptiranoj funkciji prethodnog bloka osim prvog. Višestruke operacije šifriranja prema naprijed ne mogu se izvršiti paralelno, dok se dešifriranje može. Slika 4. prikazuje shemu šifriranja i dešifriranja kod CFB moda (Stallings, 2017).



Slika 4. Prikaz CFB moda (Stallings, 2017., str. 219.)

3. *Output Feedback* (OFB)

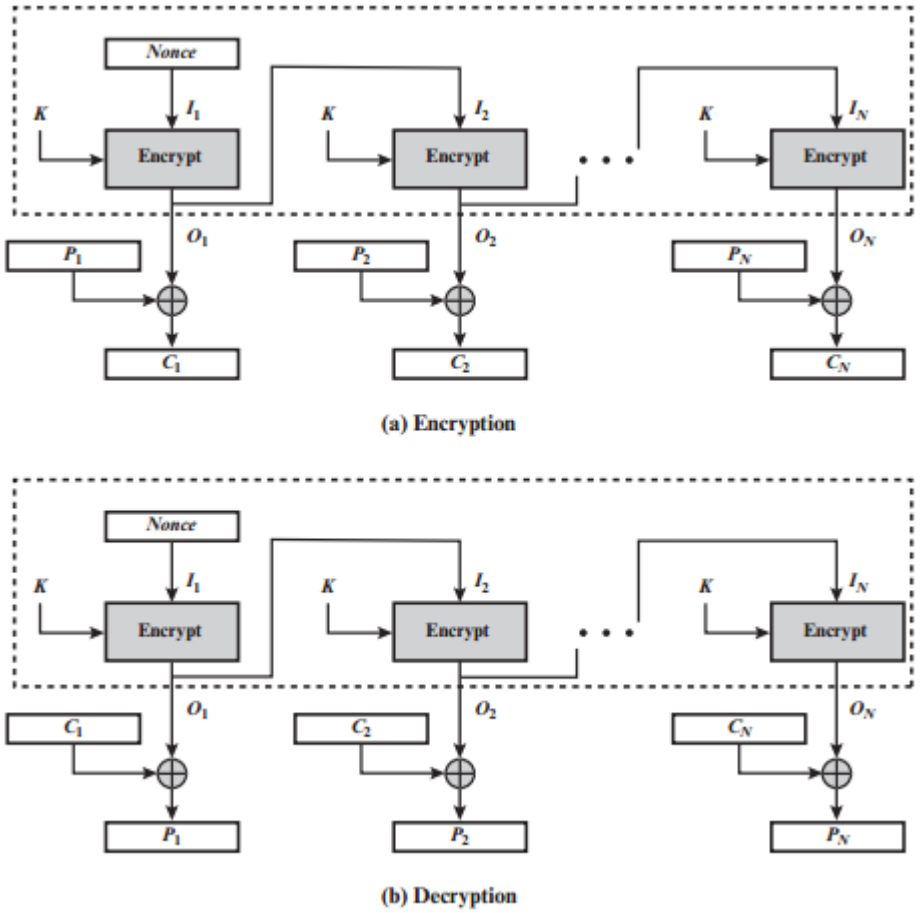
Način rada OFB sličan je CFB-u, razlika je što se izlazna funkcija šifriranja vraća natrag kako bi postala ulaz za šifriranje. Druga razlika je u tome što način rada OFBa funkcionira na punim blokovima čistog teksta i šifriranog teksta, dok CFB radi na s-bitnom podskupu. Primjer gdje se koristi OFB način šifriranja je satelitska komunikacija. Slika 5. prikazuje shemu šifriranja i dešifriranja kod OFB moda (Stallings, 2017). Šifriranje se izvodi korištenjem isključivo ILI operacijom ($\oplus$) između P_j i funkcije šifriranja E. Funkcija za šifriranje sadrži ključ K, prethodne blokove čistog i šifriranog teksta između na kojima je izvršena isključivo ILI operacija.

Formula za šifriranje:

$$C_j = P_j \oplus E(K, [C_{j-1} \oplus P_{j-1}])$$

Formula za dešifriranje:

$$P_j = C_j \oplus E(K, [C_{j-1} \oplus P_{j-1}])$$



Slika 5. Prikaz OFB moda (Stallings, 2017., str. 221.)

Šifrirani ili otvoreni tekst koristi se samo za konačni XOR, operacije blok šifriranja mogu se izvršiti unaprijed čime se konačni korak može izvršiti paralelno nakon što je dostupan otvoreni ili šifrirani tekst (Stallings, 2017).

4. *Counter* (CTR)

Na svaki blok čistog teksta izvršava se operacija isključivo ILI sa šifriranim brojačem. Brojač T se povećava za svaki sljedeći blok što pokazuje slika 6. Šifriranje ili dešifriranje u CTR modu može se izvršiti paralelno na više blokova otvorenog teksta P_N ili šifrata C_N. Za načine povezivanja, algoritam mora dovršiti izračun za jedan blok prije početka

sljedećeg bloka. To ograničava maksimalnu propusnost algoritma u uzajamnom vremenu za jedno izvršenje šifriranja ili dešifriranja bloka. U CTR modu propusnost je ograničena prema količini paralelizma koji se postiže. CTR je koristan za zahtjeve velike brzine. Oznaka T označava brojač, oznaka $\oplus$ označava „logičko ILI“ (eng. XOR), oznaka MSB (eng. *Most Significant Bit*) označava najznačajniji bit. (Stallings, 2017).

Formula za šifriranje:

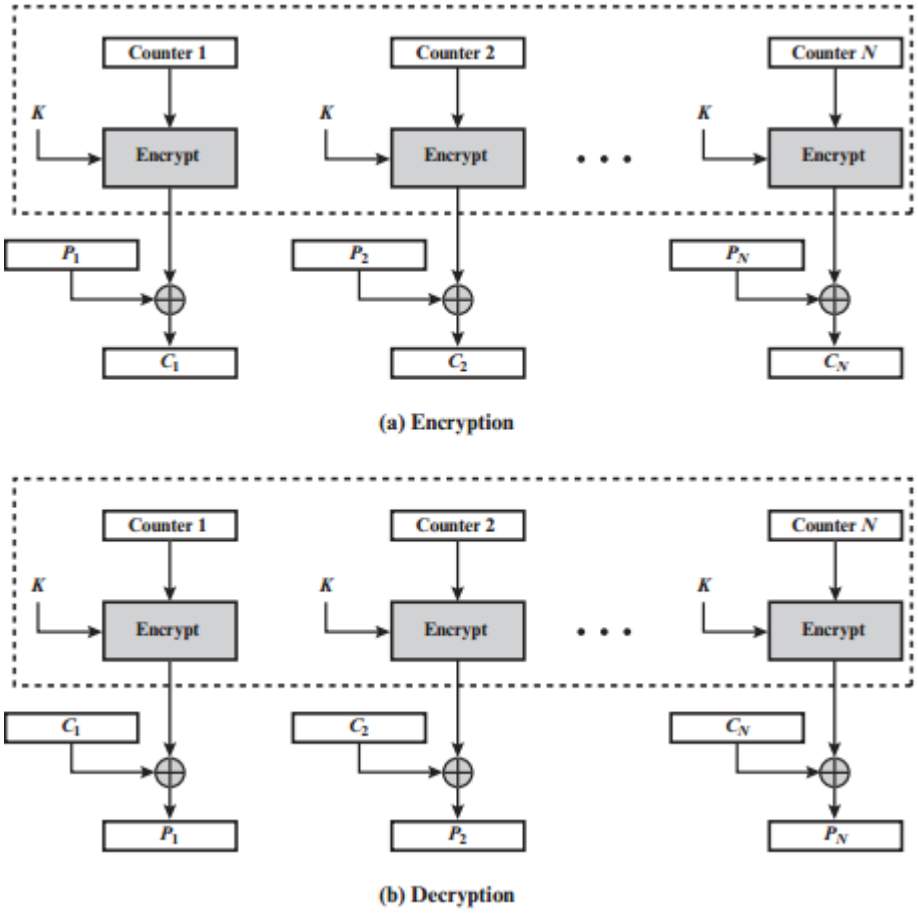
$$C_j = P_j \oplus E(K, T_j) \quad j = 1, \ldots, N-1$$

$$C_N^* = P_N^* \oplus MSB_u[E(K, T_N)]$$

Formula za dešifriranje:

$$P_j = C_j \oplus E(K, T_j) \quad j = 1, \ldots, N-1$$

$$P_N^* = C_N^* \oplus MSB_u[E(K, T_N)]$$



Slika 6. Prikaz CTR načina rada (Stallings W., 2017., str. 223.)

CTR se u velikoj mjeri koristi za šifriranje datoteka jer nudi slučajni pristup podacima.

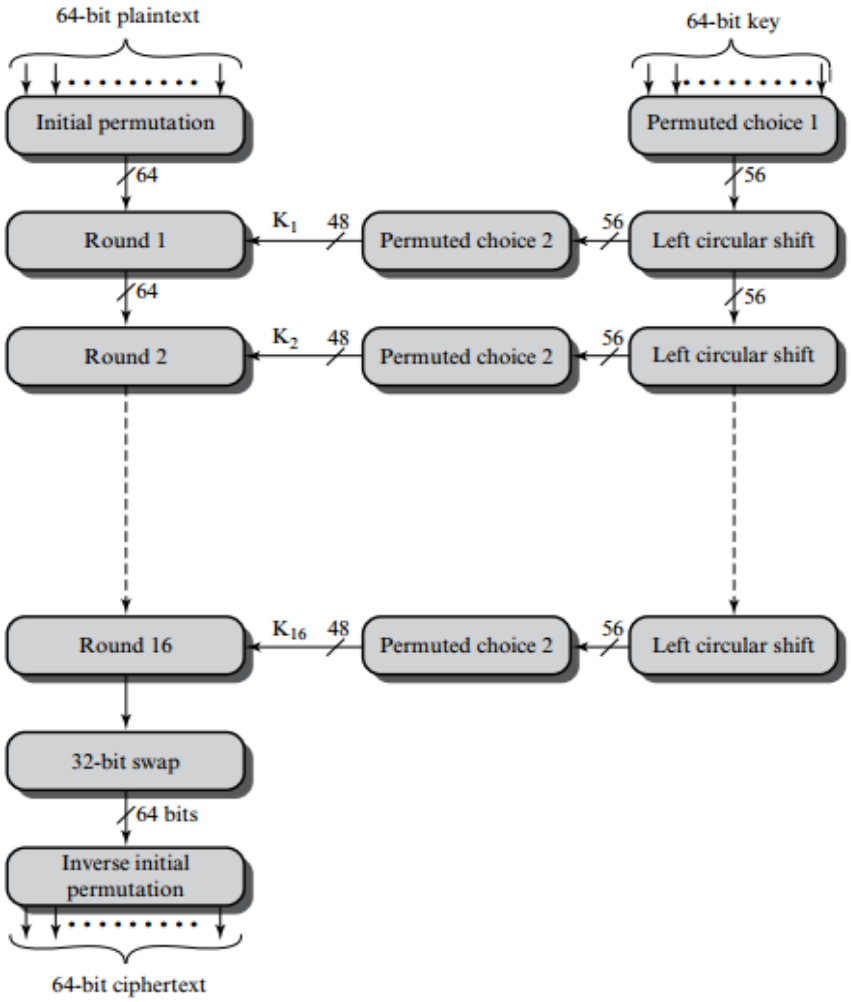
2.1.1 Data Encryption Standard (DES)

Standard za šifriranje podataka DES bio je najčešće korišten sustav šifriranja do 2001. godine, a izdan je 1977. godine od strane američkog nacionalnog instituta za standarde i tehnologiju (NIST). Algoritam DEA kojeg koristi DES, šifrira podatke u 64-bitnim blokovima pomoću 56-bitnog ključa. Algoritam pretvara 64-bitni otvoreni tekst kroz niz koraka u 64bitni šifrat.

> *Tijekom godina, DES je postao dominantan algoritam simetričnog šifriranja, posebno u financijskim aplikacijama. Nakon nekog vremena DES nije bio preporučljiv za kriptiranje tajnih državnih podataka od strane NIST-a, pa je 1994. godine izdao novu verziju svog standarda koji je ukazivao na to da se DES treba koristiti samo za naslijeđene sustave, a novi 3DES za nove sustave.* (Stallings, 2017., str. 129.)[1]

Ukupna shema za DES šifriranje ilustrirana je na slici 7. Kao i kod svih shema šifriranja, postoje dva ulaza u funkciju šifriranja, ključ i otvoreni tekst koji treba biti šifriran. U ovom slučaju, običan tekst je duljine 64 bita, a ključ je duljine 56 bita. Na lijevoj polovici slike 7 prikazana je obrada otvorenog teksta koja počinje u tri faze. Najprije, 64-bitni tekst prolazi kroz početnu permutaciju (eng. initial permutation) koja preoblikuje bitove kako bi se stvorio zatvoreni ulaz.

[1] Vlastiti prijevod



Slika 7. Općenito funkcioniranje simetričnog kriptosustava u komunikacij (Stallings, 2017., str. 130.)

Nakon toga slijedi faza koja se sastoji od šesnaest ponavljanja iste funkcije, ona uključuje permutiranje i supstitucijske operacije na ulaznom vektoru bitova. Izlaz posljednjeg (šesnaestog) ponavljanja sastoji se od 64 bita koji su funkcija unosa teksta i ključa. Lijeva i desna polovica izlaza se obrću i prolaze kroz permutaciju [IP^1] koja je inverzna od početne permutacijske funkcije kako bi se stvorio 64-bitni šifrirani tekst. Uz iznimku početnih i konačnih permutacija, DES ima strukturu Feistelove mreže. Desni dio slike 7 pokazuje način na koji se koristi 56-bitni ključ. U početku ključ prolazi kroz funkciju permutacije, a zatim za svaki od šesnaest krugova, potključ (K_i) proizvodi kombinaciju lijevog kružnog pomaka i permutacije. Ista permutacijska funkcija daje u raznim iteracijama različit potključ.

2.1.2 Triple Data Encryption Standard *(Triple DES ili 3DES)*

Triple Data Encryption Standard je vrsta šifriranja gdje se DES primjenjuje tri puta na svaki blok teksta. Veličina ključa u 3DES-u povećana je kako bi se osigurala dodatna sigurnost. *Triple* DES privremeno je zamijenio nedostatke DES-a, ali nije bio dovoljno brz i pogodan za šifriranje pojedinih vrsta podataka poput digitalnih videa te je postojala potreba za novim, sigurnijim standardom za simetričnu kriptografiju.

2.1.3 Advanced Encryption Standard (AES)

Advanced Encrytion Standard je simetričan standardizirani sustav šifriranja, koji je namijenjen tome da zamijeni DES zbog sigurnosnih propusta i sporosti 3DES-a. NIST je objavio natječaj za novi i brži simetrični blokovski algoritam koji treba raditi sa 128-bitnim blokovima i s tri duljine ključa: 128, 192 i 256 bita.

Natječaj je završio 15.06.1998. godine, a broj prijavljenih iznosio je 21, dok ih je 15 odgovaralo NIST-ovim uvjetima. U kolovozu 1999. godine NIST je objavio pet finalista: MARS, RC6, RIJNDAEL, SERPENT i TWOFISH. Nakon svih pregleda i testiranja, 02.08.2000. godine objavljen je pobjednik natječaja AES RIJNDAEL (Stallings, 2017).

> *„RIJNDAEL su razvili belgijski kriptografi Joan Daemen i Vincent Rijmen s Katoličkog sveučilišta Leuven, po kojima je i dobio ime. RIJNDAEL se razlikuje od ostalih finalista po tome što se u konstrukciji S-kutija koriste aritmetičke operacije u konačnom polju $GF(2^8)$. U AES-u se sve operacije izvode na 8-bitnim bajtovima.“* (Dujella, 2007)

Algoritam za šifriranje AES definira brojne transformacije koje treba izvesti nad podacima. Transformacije se ponavljaju tijekom više krugova. Broj krugova određen je duljinom ključa:

- 10 rundi za 128bitni ključ (16 bajta)
- 12 rundi za 192bitni ključ (24 bajta)
- 14 rundi za 256bitni ključ (32 bajta).

Ulaz za šifriranje i dešifriranje algoritma je jedan 128-bitni blok. Taj blok prikazan je kao 4×4 kvadratna matrica bajtova. Blok se mijenja u svakoj fazi šifriranja ili dešifriranja. Slično tome, ključ je prikazan kao kvadratna matrica bajtova. Svaki znak je četiri bajta, a ukupni raspored ključa je 44 znaka za 128-bitni ključ. Na primjer, prva četiri bajta 128-bitnog unosa u šifrantu zauzimaju prvi stupac matrice, druga četiri bajta zauzimaju drugi stupac i tako dalje. Slično tome, prva četiri bajta proširenog ključa, koje čine riječ, zauzimaju prvi stupac matrice.

Šifra se sastoji od „N krugova“, pri čemu broj krugova ovisi o duljini ključa. AES šifriranje izvršava u svakom krugu (osim posljednjeg kruga) isti redoslijed jednostavnih operacija na matricama. Svaka runda se sastoji od četiri različite transformacijske operacije:

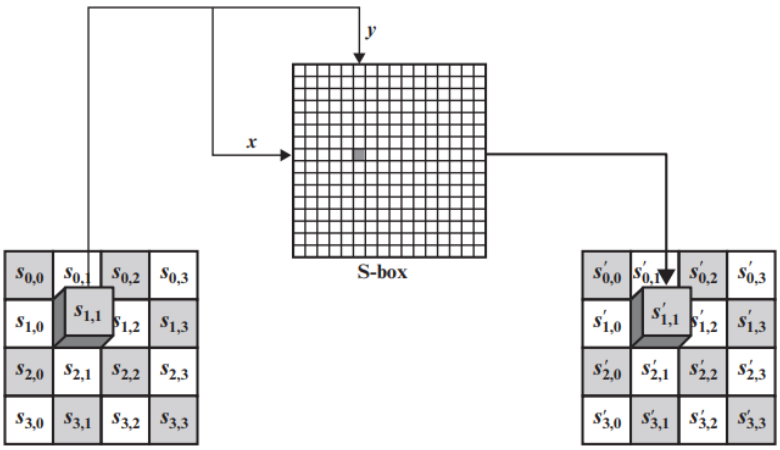
1. *SubstituteBytes* – je nelinearna operacija temeljena na posebno dizajniranoj S-kutiji. Svrha ove operacije jest oduprijeti se diferencijalnoj i linearnoj kriptoanalizi te drugim matematičkim napadima.
2. *ShiftRows* - je linearna operacija čija je svrha stvaranje difuzije.
3. *MixColumns* - svrha je ista kao i *ShiftRows*.
4. *AddRoundKey* - je skup isključivo ILI operacija na matricama. To je linearna operacija čija je svrha stvoriti zbunjenost.

Operacija *SubstituteBytes* koristi S-kutiju (*eng. S-Box*) da zamjeni bajtove. *S-kutija* je 16×16 matrica bajtova (tablica 1), koja je definirana na osnovi operacije razmnožavanja GF(2^8). AES *S-Box* je permutacija svih 256 bajta.

Tablica 1. S-kutija (eng. S-Box)

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***A***	***B***	***C***	***D***	***E***	***F***
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Svaki bajt (element bloka matrice 4×4) ima 8 bita od kojih su prva 4, x os, a zadnja 4 y os u matrici (tablici 1.). Primjerice ako je vrijednost elementa „5B“, gledamo gdje se sijeku red „5“ i stupac „B“, prema tome zamijenjena vrijednost iznosila bi „39“. Slika 8 prikazuje logiku operacije zamjene bajtova.



Slika 8. Operacija "Zamjena bajtova" (Stallings, 2017., str. 180.)

Primjer korištenja transformacije, odnosno zamjene bajtova 4×4 matrice prikazana je na slici 9.

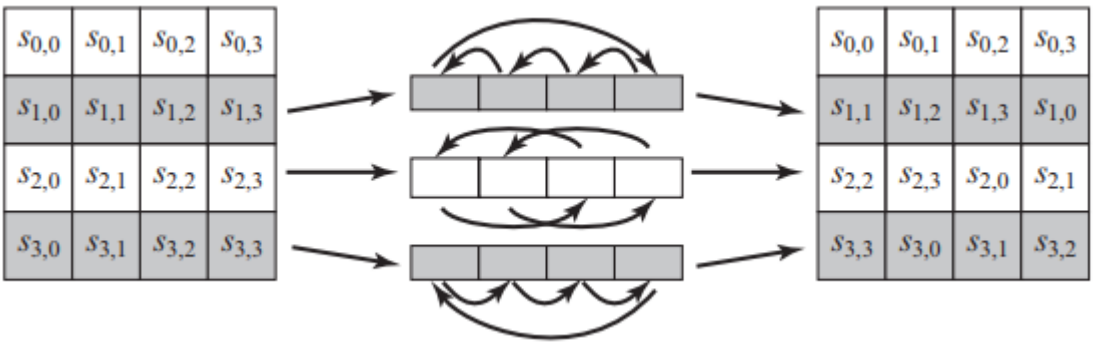
EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

Slika 9. Rezultat zamjene bajtova (Stallings, 2017. str. 182.)

Iduća operacija *ShiftRows* pravi izmjene u 2., 3. i 4. redu matrice, dok prvi red ostaje nepromijenjen. Na slici 10 prikazuje se logika izmjene redova.



Slika 10. Zamjena redova (Stallings, 2017., str. 186.)

Primjer promijenjenih redova u 4×4 matrici prikazana je na slici 11.

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Slika 11. Rezultat promjenjenih redova (Stallings, 2017., str. 185.)

Nakon operacije miješanja redova slijedi operacija *MixColumns*, odnosno miješanje stupaca. Svaki bajt stupca preslikava se u novu vrijednost koja je funkcija svih četiri bajta u tom stupcu. Slika 12 prikazuje primjer transformacije stupaca matrice.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Slika 12. Transformacija stupaca (Stallings, 2017. str. 186.)

Miješanje se izvodi tako da se element iz prve matrice (na slici 13), iz prvog reda množi s prvim elementom prvog stupca druge matrice, poslije toga ide drugi element reda prve matrice s drugim elementom prvog stupca druge matrice. Takav se proces odvija sve dok ne dođe do kraja prvog stupca što je prikazano formulama sa slike 13. Ista radnja se odvija i s ostalim stupcima.

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

Slika 13. Način računanja jednog stupca matrice (Stallings, 2017., str. 187.)

Rezultat miješanja stupaca na matrici izgleda kao na slici 14, a primjer računanja jednog stupca matrice je na slici 15.

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Slika 14. Rezultat promjenjenih stupaca matrice (Stallings, 2017., str. 187.)

$$\begin{aligned} (\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\ \{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} &= \{37\} \\ \{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) &= \{94\} \\ (\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) &= \{ED\} \end{aligned}$$

Slika 15. Primjer računanja jednog supca matrice (Stallings, 2017., str. 187.)

Dobivene vrijednosti množenja pretvaraju se u binarne brojeve te se izvodi operacija isključivo ILI. Slika 16 prikazuje rezultat dobiven operacijom isključivo ILI za dobivanje prvog elementa prvog stupca matrice.

$$
\begin{aligned}
\{02\} \cdot \{87\} &= 0001\ 0101 \\
\{03\} \cdot \{6E\} &= 1011\ 0010 \\
\{46\} &= 0100\ 0110 \\
\{A6\} &= \underline{1010\ 0110} \\
&\quad 0100\ 0111 = \{47\}
\end{aligned}
$$

Slika 16. Način dobivanja rezultata korištenjem operacije "isključivo ILI" (Stallings, 2017., str. 187.)

Četvrta funkcija pod nazivom *AddRoundKey* vrši isključivo ILI operaciju između glavne 128 bitne matrice i matrice pod nazivom *RoundKey,* također 128 bitne. Rad se prvo odvija između prvog stupca glavne matrice i jedne riječi *RoundKey* matrice. Slika 17 prikazuje primjer operacije *AddRoundKey. „AddRoundKey je XOR aes-bloka s međuključem koji odgovara trenutnoj rundi.“* (Dujella A., 2007)

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$\oplus$

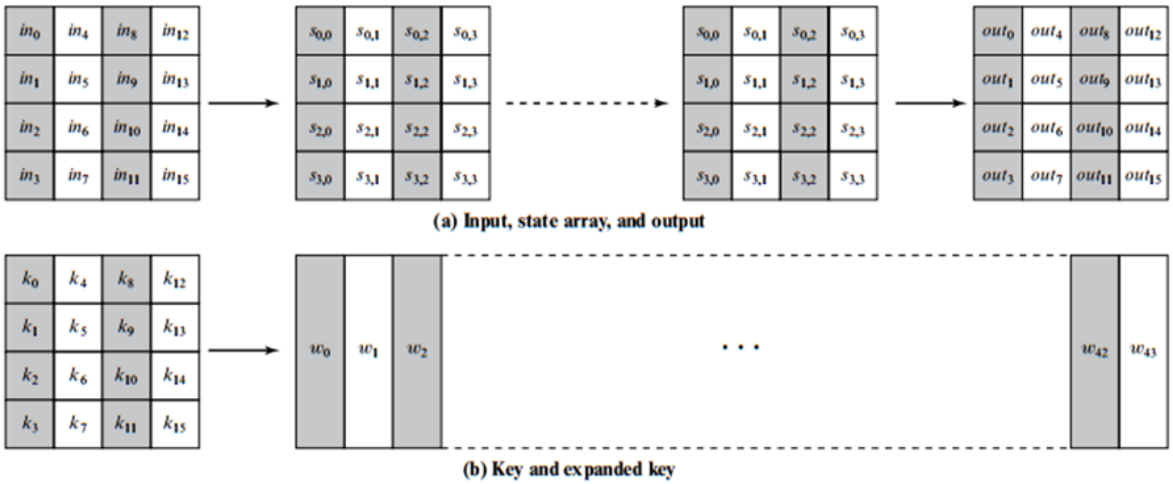
AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

$=$

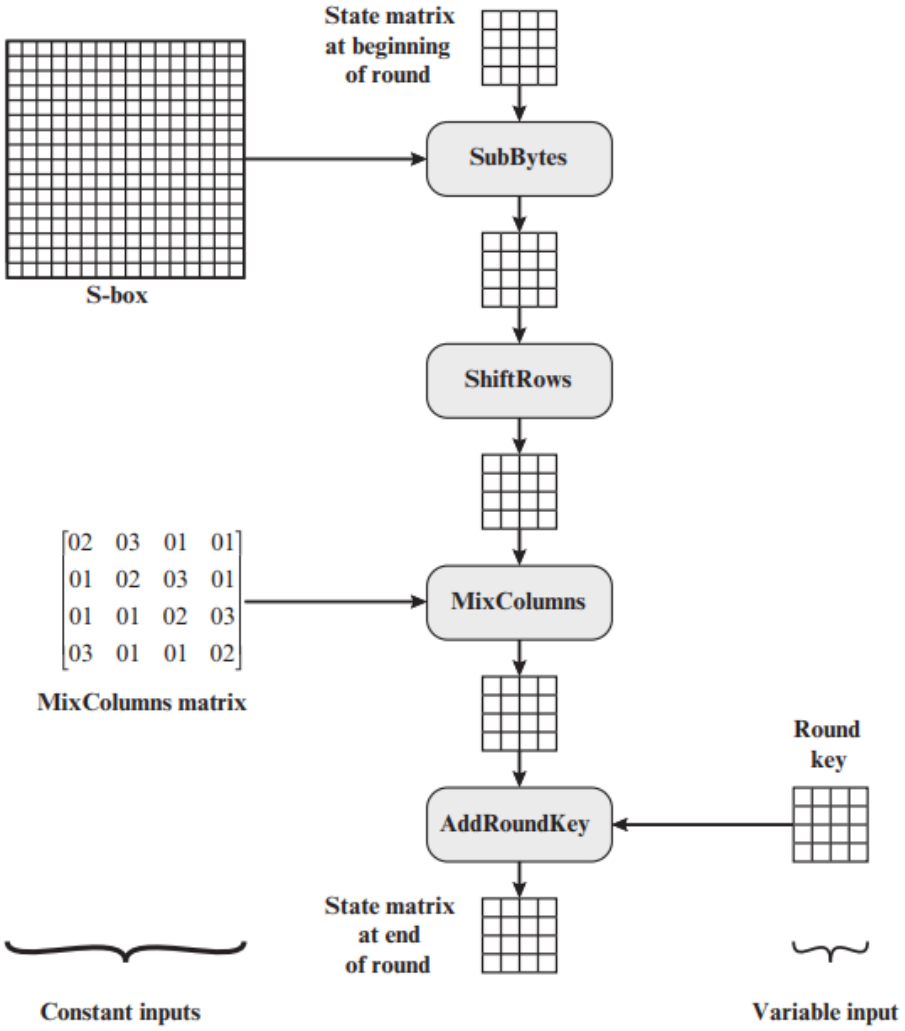
EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

Slika 17. Rezultat operacije "AddRoundKey" (Stallings, 2017., str. 189.)

Prije prve runde dodaje se ključ *AddRoundKey*, a u zadnjoj rundi izostavlja se funkcija *MixColumns* (Dujella, 2007). Slika 18 pokazuje da je izlaz svakog kruga matrica 4×4, s izlazom konačnog kruga kriptiranog teksta. Također, ključna funkcija ekspanzije generira $N + 1$ kružnog ključa, od kojih je svaka odvojena matrica od 4×4. Svaki ključ služi kao jedan od ulaza u transformaciju AddRoundKey u svakom krugu. Na slici 19 prikazan je slijed jedne runde šifriranja u AES kriptosustavu.



Slika 18. AES struktura podatala (Stallings, 2017., str. 176.)



Slika 19. Prikaz jedne runde AES šifriranja (Stallings, 2017., str. 190.)

2.1.3 Usporedba algoritama

U ovom potpoglavlju prikazane su neke od glavnih razlika važnih simetričnih kriptosustava i opće usporedbe.

DES: (Data Encryption Standard) bio je prvi standard za šifriranje kojeg preporučuje NIST (National Institute of Standards and Technology). Temelji se na predloženom IBM algoritmu nazvanom Lucifer. DES je postao standard od 1974. godine, a značajno je da su od tog vremena zabilježeni mnogi napadi i razne metode koje iskorištavaju njegove slabosti, što ga čini nesigurnom blok-šifrom.

3DES je poboljšanja verzija DES-a, odnosno razlika je u tome da se algoritam DES primjenjuje 3 puta kako bi se povećala razina sigurnosti. Usprkos tome, poznata je činjenica da je 3DES sporiji od ostalih metoda šifriranja blokova.

AES: (Advanced Encryption Standard) standard za šifriranje preporučen NIST-a kao zamjena za DES. Algoritam Rijndael odabran je 1997. godine nakon natječaja za odabir najboljeg standarda šifriranja.

U tablici 2. prikazani su rezultati testiranja pojedinih simetričnih algoritama pomoću Crypto++[2] biblioteke. Pokazalo se da su Blowfish i AES otporniji na napade nego ostali, odnosno da imaju bolje šifriranje.

[2] Crypto++ je besplatna C++ biblioteka kriptografskih shema.

Tablica 2. Usporedba rezultata pomoću Crypto++ (Al-Tamimi, 2006)

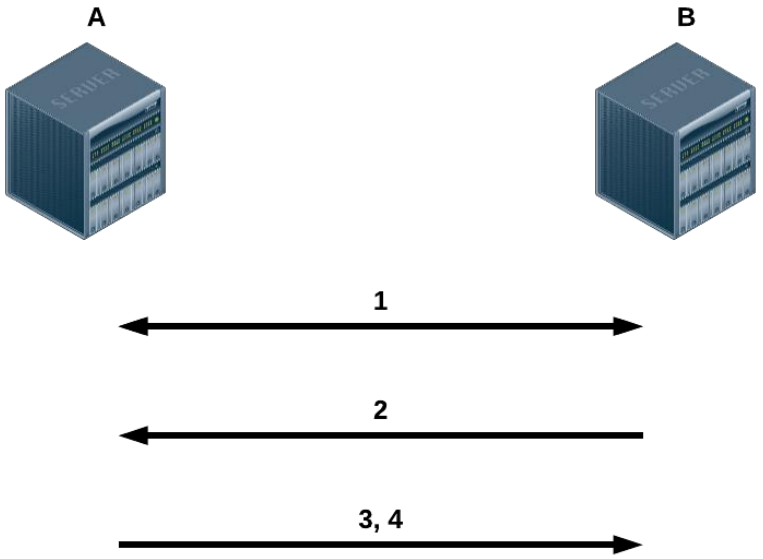
Algorithm	Megabytes(2^{20} bytes) Processed	Time Taken	MB/Second
Blowfish	256	3.976	64.386
Rijndael (128-bit key)	256	4.196	61.010
Rijndael (192-bit key)	256	4.817	53.145
Rijndael (256-bit key)	256	5.308	48.229
Rijndael (128) CTR	256	4.436	57.710
Rijndael (128) OFB	256	4.837	52.925
Rijndael (128) CFB	256	5.378	47.601
Rijndael (128) CBC	256	4.617	55.447
DES	128	5.998	21.340
(3DES)DES-XEX3	128	6.159	20.783
(3DES)DES-EDE3	64	6.499	9.848

Jedna od AES-ovih specifičnosti je korištenje konačnog polja $GF(2^8)$ (alternativna oznaka je F_{2^8}). Preciznije, elementi od $GF(2^8)$ su polinomi oblika $a_7x^7 + a_6x^6 + \ldots + a_1x + a_0, a_1 \in \{0,1\}$, a operacije su zbrajanje i množenje polinoma iz $Z_2[X]$ modulo fiksni ireducibilni polinom $g(x) = x_8 + x_4 + x_3 + x + 1$.

Dakle, uzimamo da je $GF(2^8) = Z_2[X]/(g(x))$. Elemente od $GF(2^8)$ možemo prikazati i kao bajtove (nizove od 8 bitova). Npr. polinomu $x_6 + x_4 + x_2 + x + 1$ odgovara bajt 01010111 ili 57 u heksadecimalnom zapisu (Dujella, 2007).

2.3. Asimetrična kriptografija

Asimetrično kriptiranje je poznato kao „kriptografija javnog ključa“ (PKC[3]), kod kojeg se koriste dva ključa: javni ključ koji je poznat javnosti i privatni ključ koji je poznat samo korisniku. Slika 20 prikazuje upotrebu dvaju ključeva između čvora A i B. Nakon prihvaćanja vrste šifriranja koja će se koristiti u vezi, čvor B šalje svoj javni ključ u čvor A. Čvor A koristi primljeni javni ključ za šifriranje njegove poruke. Nakon što šifrirane poruke stignu, čvor B koristi svoj privatni ključ da bi ih dešifrirao (Shostack, 2014).



Slika 20. Postupak simetrične kriptografije (Al-Tamimi, 2006)

„Asimetrične tehnike šifriranja gotovo su 1000 puta sporije od simetričnih tehnika, jer zahtijevaju veću računalnu procesorsku snagu.“ (Al-Tamimi, 2006)

Kako bi se ostvarile prednosti simetričnih i asimetričnih metoda šifriranja, obično se koristi hibridna tehnika. U hibridnoj tehnici, asimetrična kriptografija koristi se za razmjenu

[3] PKC (*Public Key Cryptography*) – Kriptografija javnog ključa

tajnog ključa za simetrično šifriranje, a zatim se simetrično šifriranje koristi za prijenos podataka između pošiljatelja i primatelja.

Prema Dujelli, u modernoj se kriptografiji, onoj koja se na primjer pojavljuje u situacijama kupovine preko interneta, mogu pojaviti navedeni problemi:

1. *POVJERLJIVOST (confidentiality): Poruku koju osoba A šalje osobi B ne može pročitati nitko drugi.*
2. *VJERODOSTOJNOST (authenticity): Osoba B zna da je samo osoba A mogla poslati poruku koju je ona upravo primila.*
3. *NETAKNUTOST (integrity): Osoba B zna da poruka koju je poslala osoba A nije promijenjena prilikom slanja.*
4. *NEPOBITNOST (non-repudiation): Osoba A ne može kasnije zanijekati da je poslala poruku.*

„Kriptosustavi javnog ključa su mnogo sporiji od simetričnih. Stoga je njihova uporaba u 1. problemu ograničena na razmjenu ključeva za simetrične šifre. S druge strane, "digitalni potpisi", koji se koriste u rješavanju 2., 3. i 4. problema, zahtijevaju uporabu kriptografije javnog ključa.“ (Dujella, 2007)

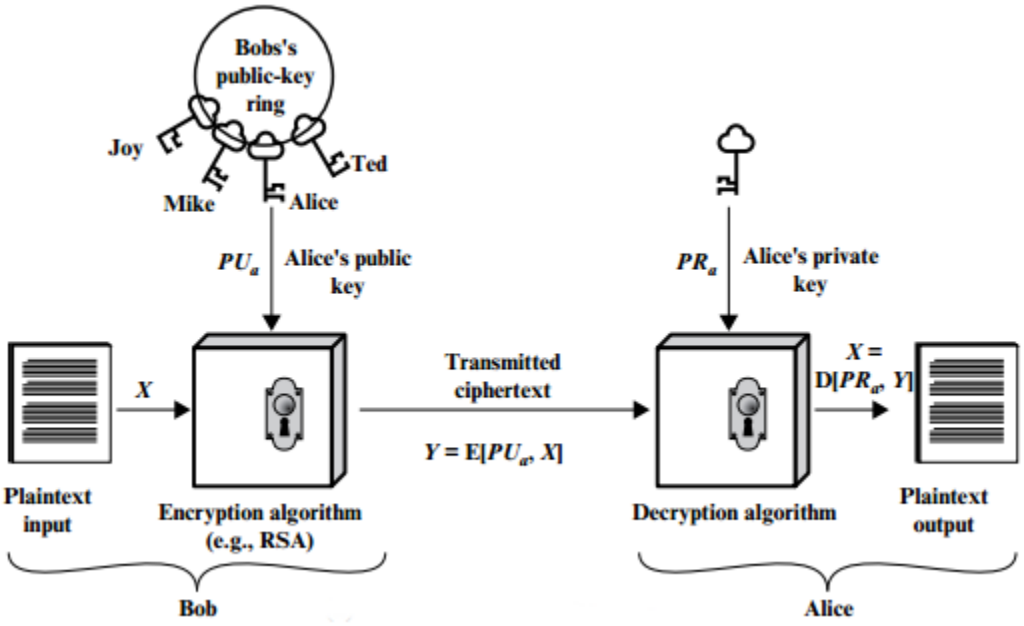
ElGamavol kriptosustav koristi jednosmjernu funkciju koju je lako izračunati, dok je njezin inverz teško izračunati bez tajne vrijednost. Takva se funkcija zove osobna jednosmjerna funkcija koja je u ElGamalovom kriptosustavu konstruirana na problemu diskretnog logaritma.

Osobne jednosmjerne funkcije kod drugih kriptosustava zasnovane su na matematičkim problemima poput:

- problem ruksaka
- faktorizacija velikih brojeva
- dekodiranje linearnih kodova
- problem diskretnog logaritma

2.2.1 RSA (Rivest–Shamir–Adleman) kriptosustav

Asimetrični kriptosustav RSA razvili su Ron Rivest, Adi Shamir i Len Adleman 1977. godine, a koji je prvi put je objavljen na MIT-u (Massachusetts Institute of Technology) 1978. godine. Shema *Rivest-Shamir-Adleman (RSA)* od tada je vladala kao najrašireniji i najprimjenjiviji opći pristup za šifriranje javnih ključeva. Slika 21 prikazuje šifriranje s javnim ključem pomoću RSA algoritma.



Slika 21. Šifriranje s javnim ključem pomoću RSA algoritma (Stallings, 2017., str. 287.)

RSA je kriptosustav u kojem su šifrirani i nešifrirani tekstovi cijeli brojevi između 0 i $n-1$ za neki n. Tipična veličina za n je 1024 bita, odnosno 309 decimalnih znamenki, gdje je n manji od 2^{1024}. Otvoreni tekst šifriran je u blokovima, pri čemu svaki blok ima binarnu vrijednost manju od nekog broja n. Veličina bloka mora biti manja ili jednaka $\log_2(n)+1$. U praksi veličina blokova je i veličina bitova, $2^i < n \leq 2^{(i+1)}$. Šifriranje i dešifriranje su sljedećeg oblika, za neki blok otvorenog teksta M i blok šifriranog teksta C.

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Prema Stallingsu, pošiljatelj i primatelj moraju znati vrijednost n. Pošiljatelj zna vrijednost e, a samo primatelj zna vrijednost d. Dakle, ovo je algoritam šifriranja s javnim ključem $PU = \{e, n\}$ i privatnim ključem $PR = \{d, n\}$. Da bi ovaj algoritam bio zadovoljavajući za šifriranje javnih ključeva, moraju biti zadovoljeni sljedeći uvjeti:

- moguće je pronaći vrijednosti e, d i n tako da $M^{ed} \bmod n = M$ za sve $M < n$,
- relativno je lako izračunati $M^e \bmod n$ i $C^d \bmod n$ za sve vrijednosti $M < n$,
- nije praktično moguće utvrditi d dobiven od e i n.

$$M^{ed} \bmod n = M$$

Prethodni odnos podrazumijeva da ako su e i d multiplikativni inverz modulo $\varphi(n)$, gdje je $\varphi(n)$ Eulerova funkcija (*Euler totient*[4]), tada je odnos između e i d moguće izraziti kao

$$ed \bmod \varphi(n) = 1$$

e i d su multiplikativni inverteri mod $\varphi(n)$. Prema pravilima modularne aritmetičke vrijednosti, to vrijedi samo ako je d (stoga i e) relativno mala vrijednost za $\varphi(n)$. Na temelju toga funkcija glasi:

$$\gcd(\varphi(n), d) = 1.$$

Gdje su varijable:

- p, q dva primarna broja (privatni, izabrano)
- $n = pq$ (javno, izračunato)
- $e, s, gcd\ (\varphi(n), e) = 1;\ 1 < e < \varphi(n)$ (javno, izabrano)
- $\mathrm{d\ K}\ e^{-1}\ (\mathrm{mod}\ \varphi(\mathrm{n}))$ (privatno, izračunato)

Alisa generira javni i privatni par ključeva; Bob šifrira pomoću Alisinog javnog ključa; i Alisa dešifrira koristeći svoj privatni ključ. Privatni ključ sastoji se od $\{d, n\}$ i javni ključ se sastoji od $\{e, n\}$. Pretpostavimo da je Alisa objavila svoj javni ključ te da joj Bob želi poslati poruku M. Tada Bob izračunava i šalje $C = M^e \bmod n$. Po primitku tog šifrata, Alisa dešifrira poruku računanjem $M = C^d \bmod n$.

[4] Euler totient – "Eulerova funkcija φ(n) definirana je kao broj pozitivnih cjelina koje su manje ili jednake n i relativno su premale za n. Na primjer, φ(9) = 6, jer svaki od: 1, 2, 4, 5, 7, 8 su relativno najbolji za 9, dok 3 i 6 nisu. Eulerov teorem navodi da ako je gcd(a, n) = 1, onda inverzni modulo n sigurno postojati. Štoviše, ima jednostavniju formu pomoću Eulerove funkcije totienta φ." (Wang I Kissel, 2015., str. 97.)

2.2.2 Kriposustav ElGamal

ElGamal je asimetrični kriptosustav temeljen na problemu diskretnog logaritma. Koristi se za šifriranje, dešifriranje te za digitalni potpis. Autor kriptosustava ElGamal je Taher ElGamal koji je predstavio svoj asimetrični algoritam 1985. godine, a matematički je zasnovan na istoj ideji kao i Diffie-Hellman protokol. ElGamal-u je glavna alternativa RSA kriptosustav jer imaju slične razine sigurnosti. Sigurnost ElGamala temelji se na težini problema diskretnih algoritama kao i na Diffie-Helman problemu. Nedostatak ElGamal kriptosustava je potreba za slučajnim brojevima što je jedan od razloga koji ga čini sporijim u odnosu na druge kriptosustave, ali to nije veliki problem ako se koristi samo za razmjenu privatnih ključeva. Ideja kriptiranja s javnim ključem je otežati dešifriranje poruke s javnim ključem u nekom razumnom vremenu. Ta ideja osigurava da se razmjene privatni ključevi za simetričnu kriptografiju koja radi brže.

Kriptosustav funkcionira tako da ima javne i tajne vrijednosti. To su informacije odnosno podaci koji se nalaze u javnom i privatnom ključu. Javni i privatni ključ služe da, primjerice, Alisa šifrira poruku javnim ključem Boba i pritom može biti sigurna da će samo Bob moći dešifrirati poruku svojim privatnim ključem. Na temelju toga se može zaključiti da su javni i privatni ključ povezani.

Najprije je potrebno odrediti javni i privatni ključ. Oni se određuju tako da se odredi veliki prosti broj p, a zatim dva slučajno generirana broja G i x koji su u rasponu $\{0, \ldots, p-1\}$. Na osnovu p, G i x računa se Y po formuli:

$$Y = G^x \bmod p$$

Vrijednosti $p, G\ i\ Y$ svrstavaju se u javni ključ, dok se u privatni ključ svrstavaju p i tajna vrijednost x. Ta situacija dokazuje da je javni ključ ovisan o privatnom ključu. Vrijednost Y iz javnog ključa je dobivena od vrijednosti G iz javnog ključa koji je potenciran s vrijednošću x iz privatnog ključa s modulo p iz oba ključa.

Kriptiranje se izvodi tako da se blokovi poruke pretvaraju u brojevni zapis koji je manji od p. Za svaki pojedini blok posebno se generira slučajni broj k za kojeg isto vrijedi da je manji od p. Poželjno je da nema tragova korištenih k brojeva jer se pomoću njih može dešifrirati poruka. Blok poruke M kriptira se po formulama:

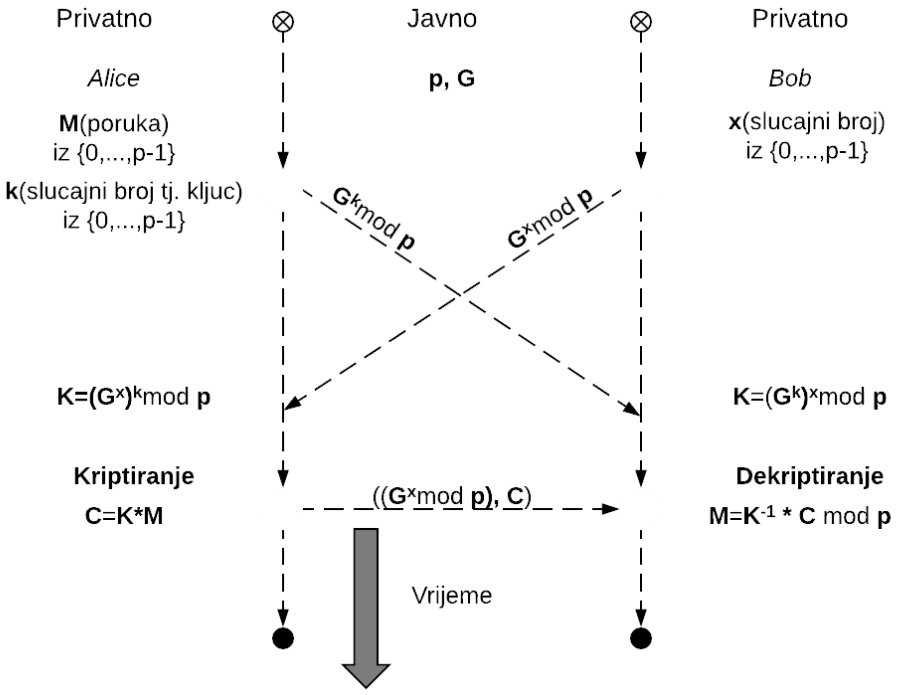
$$a = G^k \bmod p$$

$$b = Y^k M \bmod p$$

Šifrirana poruka je (a, b) i ima dvije vrijednosti za razliku od čiste poruke koju čini jedna vrijednost. Šifrirana poruka šalje se u obliku dvije vrijednosti (a, b) i može je dešifrirati samo ona osoba koja ima točno određeni privatni ključ od kojeg je izveden javni ključ. Tako je poruka M "zamaskirana" množeći se sa Y^k. Samo onaj netko koji poznaje tajni broj x može iz G^k izračunati Y^k i ukloniti "masku". Poruka se dešifrira po formuli:

$$M = \frac{b}{a^x} \bmod p$$

Operacija potenciranja modulo p, odnosno "$a^x \ mod \ p$" izvodi se na temelju metode uzastopnog kvadriranja gdje se eksponent prikaže po bazi 2. Inverz modulo p se računa primjenom Euklidovog algoritma. Slika 22 pokazuje tok šifriranja i dešifriranja poruke.



Slika 22. Tok šifriranja i dešifriranja poruke u ElGamalovom kriptosustavu (Khai, 2003)

Navedena teorija može se potkrijepiti primjerom gdje su vrijednosti: $\mathrm{P} = 2134$, $\mathrm{G} = 3$ i $\mathrm{x} = 567$. Na osnovu formule $Y = G^x \bmod p$, vrijednost Y iznosi 691. Vrijednosti $p, G \ i \ Y$ uvrštavaju se u javni ključ, a vrijednosti p i x u privatni ključ. Brojčana vrijednost poruke M je 1234, a vrijednost k se slučajno generira tijekom šifriranja. Šifrat se sastoji od dva dijela a i b. Prema formuli za $a = G^k \ mod \ p$ i $b = Y^k M \ mod \ p$ dobivene su njihove vrijednosti, s obzirom na to da je u trenutku računanja k iznosio 285, tako su dobivene

vrijednosti $a = 309\, i\, b = 244$. Dakle, šifrat izgleda $(309{,}244)$ i moguće ga je dešifrirati samo pomoću odgovarajućeg privatnog ključa. Šifrat dešifriramo po formuli $M = b/a^x \bmod p$, odnosno $M = b \times (a^x)^{-1} \bmod p$ uz uvjet da su poznate vrijednosti iz privatnog ključa. Inverz $(a^x)^{-1} \bmod p$ rješava se pomoću Euklidovog algoritma. Python implementaciju za Euklidov algoritam prikazuje programski kod 1.

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

Kod 1. Python implementacija za Euklidov algoritam (Extended Euclidean algorithm, 2017)

Algoritam odnosno Python implementacija iz koda 1. prikazuje kako se izvodi inverz po određenom modulu koristeći Euklidov algoritam. Funkcija `modinv(a,m)` zaprima vrijednost a koju treba invertirati i vrijednost m koja predstavlja modul na osnovu kojeg se vrši inverz. Funkcija `modinv(a,m)` prosljeđuje iste parametre rekurzivnoj funkciji `egcd(a,b)` koja također radi po Euklidovom algoritmu. Euklidov algoritam je postupak kojim se određuje najveća zajednička mjera dvaju cijelih brojeva. Obje funkcije uzimaju pozitivne brojeve a, b kao ulaz i vraćaju trostruku vrijednost (g, x, y), tako da je $ax + by = g$ jednaka funkciji `egcd(a,b)`. Tako je g najveća zajednička mjera, a dobivena je sumom umnoška ax i by, a funkcija `modinv(a,m)` računa $x\ mod\ m$ i vraća rezultat.

Primjer kriptiranja broja i teksta koji je izradio autor u programskom jeziku Python postavljen je na *github* na poveznici *https://github.com/akovace/elgamal_test.*

2.4. Siguran hash algoritam (SHA)

Raspršujuća *hash* funkcija ili *hash* algoritam koristi se za sažimanje i identificiranje podataka. Sažetak funkcije naziva se *hash*, a proces izračunavanja *hasha* naziva se *hashiranje*. *Hash* funkcija „sjecka i miješa“ ulazni tekst nekoliko puta s odabranim matematičkim operacijama, a rezultat je nerazumljivi tekst. Na Primjeru Python implementacije koristeći hashlib biblioteku prikazano je *hashiranje* riječi „Kriptografija“ i „kriptografija“ (Kod 2.).

```
>>> import hashlib
>>> hashlib.sha224(b"Kriptografija").hexdigest()
'5026402f1b5c9d2d17dffeeccbeb12e174fb380b91588d413b1ad776'
>>> hashlib.sha224(b"Kriptografija").hexdigest()
'5026402f1b5c9d2d17dffeeccbeb12e174fb380b91588d413b1ad776'
>>>
>>> hashlib.sha224(b"kriptografija").hexdigest()
'fbbfdba09a56c4c7c1fe383550480460d699ad8b841a5e580ba3c7b8'
>>>
```

Kod 2. Primjer pretvaranja riječi u hash SHA224

Iz navedenog primjera može se utvrditi da je *hash* uvijek isti ako je i ulaz isti. Ako je na ulaznom tekstu samo jedno slovo drugačije, što uključuje velika i mala slova, različit je i izlaz, odnosno *hash*.

Posljednjih godina, najčešće korištena *hash* funkcija je *Secure Hash algoritam*. Razvio ga je Nacionalni institut za standarde i tehnologiju (NIST) i objavio ga kao *FIPS 180* 1993. godine. Kada su otkrivene slabosti u SHA, sada poznatoj kao SHA-0, revidirana verzija izdana je kao *FIPS 180-1* 1995. godine i naziva se SHA-1. SHA se temelji na *hash* funkciji MD4, koja u velikoj mjeri modelira njegov dizajn.

SHA-1 proizvodi *hash* vrijednost od 160 bita. Godine 2002., NIST je proizveo revidiranu verziju standarda *FIPS 180-2*, koja je definirala tri nove verzije SHA-2, s duljinom *hash* vrijednosti od 256, 384 i 512 bitova, poznate kao SHA-256, SHA-384 i SHA-512. Kolektivno, ovi *hash* algoritmi poznati su kao SHA-2. Te nove verzije imaju istu temeljnu strukturu i koriste iste vrste modularnih aritmetičkih i logičkih binarnih operacija kao SHA-1. Revidirani dokument izdan je kao *FIP PUB 180-3* u 2008. godini, koji je dodao 224-bitnu verziju.

Tablica 3. Uspoređivanje SHA parametara (Stallings, 2017., str. 356.)

Algoritam	Veličina poruke	Veličina bloka	Veličina riječi	Veličina izvoda poruke
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Napomena: Vrijednosti su izražene u bitovima

NIST je 2015. godine izdao *FIPS 180-4*, koji je dodao dva dodatna algoritma: SHA-512/224 i SHA-512/256. SHA-1 i SHA-2 također su navedeni u *RFC 6234*, koji duplicira materijal iz *FIPS 180-3*, ali i dodaje implementaciju C koda. U 2005. godini NIST je najavio namjeru ukidanja odobrenja SHA-1 i oslanjanje na SHA-2 do 2010. godine. Ubrzo nakon toga, istraživački tim opisao je napad u kojem se mogu pronaći dvije zasebne poruke koje isporučuju iste SHA-1 *hash* pomoću 269 operacija, mnogo manje od 280 koje su prije bile potrebne za pronalaženje kolizije sa SHA-1 *hashom*. Ovaj rezultat trebao bi potaknuti prijelaz na SHA-2 (Stallings, 2017) .

NIST je 2015. godine objavio još jednog člana *hash* algoritma pod nazivom SHA-3 koji je razvijen od strane NSA dizajnera. SHA-3 nije planiran da zamjeni SHA-2 kao što su ga prethodne verzije namjeravale zamijeniti ranije, već ako je potrebno, SHA-3 može zamijeniti SHA-2. SHA-3 je razvijen kao druga alternativa MD5, SHA-0 i SHA-1.

3. Arhitektura sustava Helios

Helios je spoj nekoliko vrsta tehnologija, tako su u daljnjem tekstu nabrojane i opisane tehnologije koje su korištene za izradu sustava Helios. Opisana je i prikazana arhitektura baze podataka kroz relacijski model podataka te je opisan princip glasovanja kroz nekoliko dijagrama.

3.1. Tehnologije

Korištene tehnologije za izradu sustava Helios su Django *web framework*[5] koji je zasnovan na: programskom jeziku Python, bazi podataka PostgreSQL, raznim JavaScript kriptografskim paketima i RabbitMQ sustavu za slanje poruka.

3.1.1 Programski jezici

U sustavu Helios korištena su dva programska jezika, Python i JavaScript. Python je korišten za Django *web framework* i aplikacije samog projekta. JavaScript je korišten za logiku koja se odvija na pregledniku, primjerice za kriptiranje na pregledniku i slanje na server. Tako se informacije šalju kriptirane. Bez obzira na to, poželjno je imati i TSL certifikat.

3.1.1.1 Python

Python je programski jezik opće namjene, objavljen 1991. godine, a čiji je autor Guido van Rossum. Python je interpretirani[6] jezik i ima filozofiju dizajna koja naglašava čitljivost koda, naročito korištenjem razmaka da bi se odredili blokovi koda umjesto vitičastih zagrada ili ključnih riječi. Također naglašava i strukturu koja programerima omogućuje da izraze pojmove u manje redaka koda od onih koji se koristite na jezicima kao što su C++ ili Java. Značajke Pythona su dinamički sustav za automatsko upravljanje memorijom i to što podržava više paradigmi programiranja, uključujući objektno orijentirane, imperativne i funkcionalne programe. Python je multi-paradigmatski programski jezik jer podržava objektno orijentirano programiranje i strukturirano programiranje. Python sam po sebi nije naročito brz što se tiče izvođenja, ali nastoji biti što čitljiviji jezik.

---

[5] *Web* framework - je softverski "okvir" koji je dizajniran za podršku pri razvoju *web* aplikacija, uključujući *web* usluge, *web* resurse i *web* API. *Web* okviri pružaju standardne načine izrade i implementacije *web* aplikacija.

[6] Interpretirani jezici su izvedeni odmah nakon čitanja izvornog koda, bez međustadija prevođenja.

Python je poznat po tome što je vizualno lako čitljiv zato što se kod njega se često koriste (engleske) ključne riječi, dok se kod drugih jezika koriste interpunkcijski znakovi. To znači da Python ne odjeljuje izjave točka-zarezima i blokove vitičastim zagradama kao što je to slučaj kod mnogih drugih programskih jezika. Nadalje, Python ima manje sintaktičkih izuzetaka i posebnih slučajeva od C ili Pascala. Python obično koristi prazninu od 4 znaka za granicu kako bi odredio blokove umjesto vitičastih zagrada ili ključnih riječi. Uvlačenje retka dolazi nakon određenih izjava, dok vraćanje uvlačenja u prethodnu razinu označava kraj trenutnog bloka kao na primjeru programskog koda 3.

```
def fizzbuzz(n):
    if n % 3 == 0 and n % 5 == 0:
        return 'FizzBuzz'
    elif n % 3 == 0:
        return 'Fizz'
    elif n % 5 == 0:
        return 'Buzz'
    else:
        return str(n)
print "\n".join(fizzbuzz(n) for n in xrange(1, 21))
```

Kod 3. Primjer Python koda na primjeru FizzBuzz igre (FizzBuzz Python Solution, 2017)

Zbog svoje filozofije *Batteries Included*[7], Python se primjenjuje za razne vrste aplikacija i ima pristup za: *Web* i razvoj Interneta, pristup bazama podataka, desktop aplikacije, za znanost i numeriku, obrazovanje, mrežno programiranje, softver i razvoj igara. Uz sve to Python je otvorenog koda te je tako i proširiv.

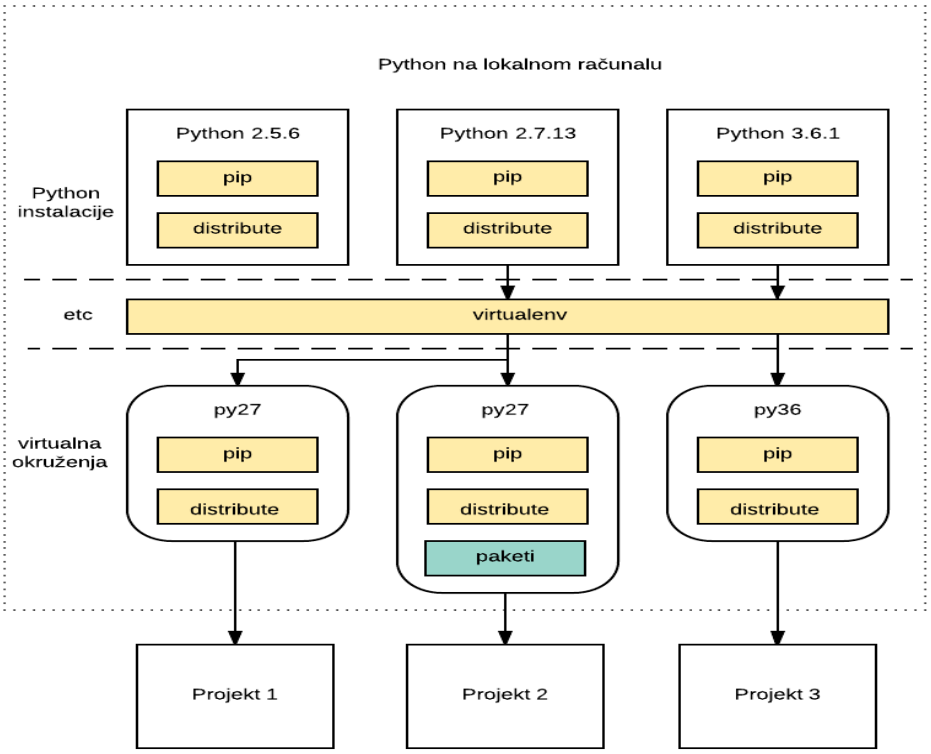
Postoji više verzija Pythona, od kojih su trenutno (30.06.2018.) stabilne verzije 2.7.15 i 3.7.0. Python2 i Python3 se razlikuju po tome što je Python2 ostavština što znači da se ono što je do sada napravljeno neće više razvijati, dok je Python3 budućnost jezika jer dolazi optimiziran i s više funkcionalnosti.

Prilikom iniciranja novog projekta potrebno je kreirati posebno virtualno okruženje za svaki novi projekt. To je potrebno iz razloga što svaki projekt može koristiti različite verzije programskog jezika Python, različite pakete i sl. Prilikom instalacije Pythona na osobno

[7] Filozofija *Batteries Included* - Python ima bogatu i svestranu standardnu knjižnicu koja je odmah dostupna, a da korisnik ne preuzima zasebne pakete. To omogućuje programeru lakše, brže I jednostavnije programiranje. (Python)

računalo on se instalira globalno, ako se instaliraju Python paketi oni se nadovezuju na globalnu instalaciju. To je dopustivo jedino ako je na računalu samo jedan projekt ili pak, ako postoji više njih koji koriste zajedničku verziju Pythona i zajedničke pakete. Ako nemaju, potrebno je instalirati virtualno okruženje (virtualenv).

Na slici 23 su prikazane instalacije tri verzije Pythona: Python 2.5.6, Python 2.7.13 i Python 3.6.1. Također su prikazana i tri projekta od kojih dva imaju virtualna okruženja od Pythona 2.7.13, a jedan ima okruženje od Pythona 3.6.1.



Slika 23. Virtualno okruženje za projekte napisane u Python programskom jeziku

3.1.1.2 JavaScript

JavaScript (skraćeno JS) je interpretirani, objektno orijentirani jezik gdje su funkcije prvoklasni objekti (*first-class*[8]), najpoznatiji je kao skriptni jezik za *web* stranice, no upotrebljava se i u mnogim okruženjima bez preglednika.

JavaScript je prototipski, multi-paradigmatski skriptni jezik koji je dinamičan i koji podržava: objektno orijentirano programiranje, imperativne i funkcionalne stilove

[8] First-class - Jezik podržava funkcije prolaza kao argumente drugim funkcijama, vraća ih kao vrijednosti iz drugih funkcija i dodjeljuje ih varijablama ili ih pohranjuje u strukture podataka. Prema Wikipedija

programiranja. JavaScript se može koristiti za dizajniranje i programiranje te utječe na to kako se *web* stranice ponašaju prilikom pojave raznih događaja.

Sličan programski jezik JavaScriptu je ECMAScript, koji se smatra standardiziranim JavaScriptom s ponekim razlikama i pravilima. Standard za ECMAScript pod nazivom *ECMA-262* predstavlja specifikaciju tog skriptnog jezika. Osnovna sintaksa je slična Javi i C++, zbog toga je jednostavnije njeno savladavanje jer se smanjuje broj novih koncepata potrebnih za učenje jezika. Objekti se kreiraju programski u JavaScriptu, povezujući metode i svojstva na inače prazne objekte za vrijeme izvođenja, za razliku od definicija sintaktičke klase koja je uobičajena u sastavljenim jezicima poput C++ i Jave. Jednom kada se objekt konstruira može se koristiti kao nacrt (ili prototip) za stvaranje sličnih objekata (About JavaScript).

JavaScript koristi format JSON za prijenos podataka. JSON se često koristi kada se podaci šalju s poslužitelja na *web* stranicu i obratno.

> *„JavaScript Object Notation (JSON) je jednostavan format teksta za prijenos podataka neovisan o programskom jeziku. Proizlazi iz ECMAScript standarda za programski jezik. JSON definira mali skup pravila oblikovanja za prijenosni prikaz strukturiranih podataka.“* (Bray, 2014)

Dakle, JSON sintaksa je izvedena od sintakse JavaScript objekta, ali JSON format je samo tekst oblikovan po određenim pravilima. Kod za čitanje i stvaranje JSON podataka može se napisati na bilo kojem programskom jeziku. JSON format je sintaktički identičan kodu za izradu JavaScript objekata. Zbog te sličnosti JavaScript program može lako pretvoriti JSON podatke u izvorne JavaScript objekte.

- Pravila JSON sintakse:
    - podaci su u parovima imena/vrijednosti,
    - podaci su odvojeni zarezima,
    - vitičaste zagrade drže predmete,
    - uglate zagrade zadržavaju polja.

Primjer JSON formata nalazi se na primjeru:

```
{
    "zaposlenici": [
      {
        "ime": "Ivan",
        "prezime": "Mari\u0107"
      },
      {
        "ime": "Marko",
        "prezime": "Horvat"
      },
      {
        "ime": "Josipa",
        "prezime": "Cukon"
      }
    ]
  }
```

Kod 4. JSON format

Kod pretvaranja JSON-a u objekt, na Kodu 5., JSON tekst spremljen je u varijablu `text` te je nakon toga parsiran naredbom `JSON.parse(text)` i tako pretvoren u objekte.

```
<!DOCTYPE html>
<html>
<body>
<h2>Create Object from JSON String</h2>
<p id="demo"/>
<script>
var text = '{"employees":[' +
   '{"firstName":"John","lastName":"Doe" },' +
   '{"firstName":"Anna","lastName":"Smith" },' +
   '{"firstName":"Peter","lastName":"Jones" }
]}';
obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
</body>
</html>
```

Kod 5. Pretvaranje JSON-a u object (W3C: JavaScript JSON)

3.1.2 Django

Primarni cilj Djanga je olakšati stvaranje složenih *web* stranica temeljenih na bazi podataka. Django ističe ponovnu iskoristivost komponente, odnosno instaliranog paketa i omogućuje brz razvoj. Sam *framework* ima vlastiti ORM[9] za komunikaciju s bazom podataka. Održava ga Django Software Foundation (DSF), nezavisna neprofitna organizacija. Python se koristi u cijelom frameworku, za postavke datoteka i modela podataka. Django dolazi s već postojećim administrativnim sučeljem koje u startu olakšava baratanje podacima. Admin sučelje je moguće nadograđivati. Django u svojem osnovnom proširenju (Django/contrib) paketu uključuje: proširiv sustav provjere, dinamičko administrativno sučelje, alat za generiranje *RSS* i *Atom syndication feedova*, *web*-lokaciju koja omogućuje instalaciji Django pokretanje više *web* stranica svaku sa svojim sadržajem i aplikacijama, alate za izradu *Google Sitemapsa* te zaštitu od SQL *injection* i drugih *web* napada. Većina njih je uključena po zadanim postavkama uz što se također nudi i okvir za izradu GIS aplikacija. Djangov konfiguracijski sustav dopušta uključivanje koda treće strane u redovni projekt, pod uvjetom da slijedi konvencije ponovne upotrebe aplikacije. Dostupne su u više od 2500 paketa, pružaju rješenja za izdanja koja nisu izvorni alati poput: prijave, pretraživanja, pružanja API-ja, CMS-a itd. S poslužiteljske strane, Django se može izvoditi na Apache, NGINX http serveru koristeći WSGI, Gunicorn ili Cherokee koristeći flup (Python modul). Django također uključuje mogućnost pokretanja FastCGI poslužitelja, omogućujući korištenje iza bilo kojeg *web* poslužitelja koji podržava FastCGI, kao što su Lighttpd ili Hiawatha. Također je moguće koristiti i neke druge WSGI-kompatibilne *web* poslužitelje.

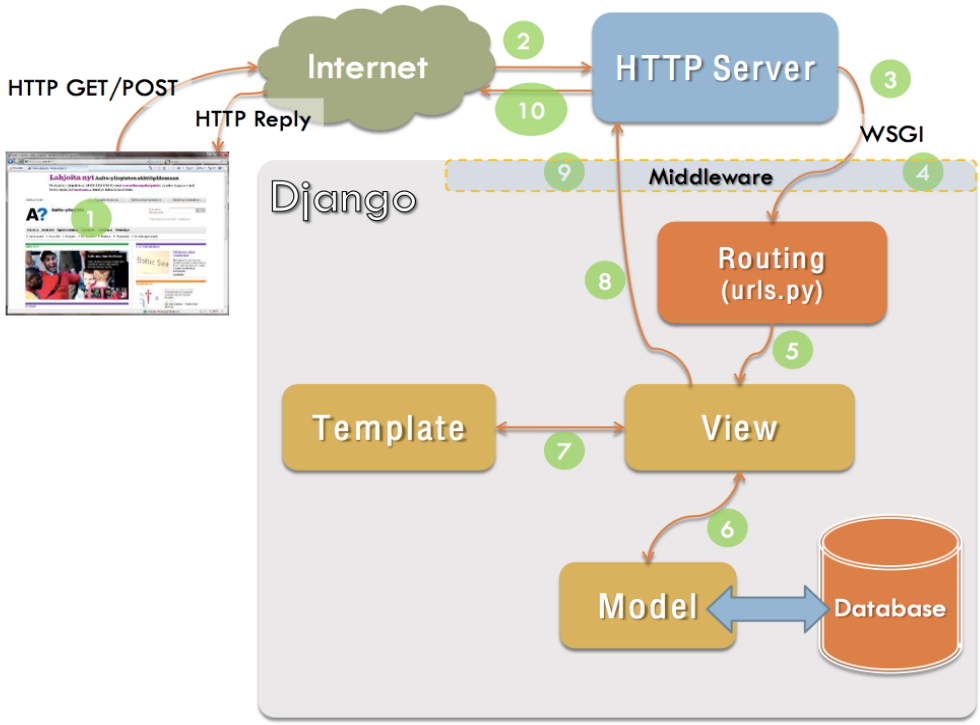
[9] ORM (Object-relational mapping) - programska tehnika za pretvaranje podataka između nekompatibilnih tipova sustava u oop jezicima. To zapravo stvara "bazu podataka virtualnog objekta" koja se može koristiti iz programskog jezika.

3.1.2.1 Model View Controller/Model View Template

Django je brz i stabilan *web* framework otvorenog koda koji je zasnovan na MVT arhitekturi po uzoru na MVC arhitekturu, a kodiran je u programskom jeziku Python. MVT se razlikuje od MVC-a već po samom značenju njihova naziva. MVT dolazi od imena *Model-View-Template*, a MVC dolazi od imena *Model-View-Controller*, gdje riječ *Model* označava klase unutar *frameworka* po kojima se kreiraju i tablice u bazi podataka. Riječ *View* se također nalazi kod oba naziva, ali ima različito značenje. Ona unutar MVT-a podrazumijeva značenje kao što ga kod MVC-a podrazumijeva riječ *Controller*. Dakle, *View* unutar MVT-a označuje logiku aplikacije koja zaprima podatke, obrađuje ih i distribuira u bazu podataka ili na *Template* koji prikazuje obrađene podatke iz *View-a*.

Slika 24 pokazuje korake i princip rada MVT-a:

1. Zahtjev korisnika za otvaranje *web* stranice zasnovane na MVT arhitekturi.
2. Internet preusmjerava upit na HTTP server tražene stranice
3. HTTP server preko WSGI sučelja za pokretanje Python *web* aplikacija ulazi u *web* aplikaciju i pretražuje put na osnovu url-a u mapi urls.py.
4. Django *web* aplikacija
5. Ruta s url-a usmjerava zahtjev, odnosno upit (1. točka) korisnika na View, odnosno logiku same tražene stranice.
6. Na traženi upit View procesira dobiveni zahtjev i šalje upit na bazu koja je definirana Model-om.
7. Nakon što je View dobio traženi odgovor s baze podataka, ponovo procesira podatke te ih “upakira” u određenu formu na osnovu Template-a. Primjer template-a je izgled same *web* stranice.
8. Obrađeni podaci, odnosno odgovor Django aplikacije vraća se na *HTTP* server.
9. Django *web* aplikacija
10. HTTP server preko interneta šalje odgovor korisniku, odnosno traženu *web* stranicu



Slika 24. Django arhitektura (Django, 2017)

3.1.2.2. Primjer Django aplikacije

Web framework po zadanim postavkama radi sa SQLite bazom. SQLite je pogodna baza kada je u pitanju manji projekt i kada nema puno upita na bazu podataka u nekom određenom vremenu. Ako se rade zahtjevniji projekti i još k tome puno upita na bazu, poželjno je odabrati neku od drugih baza podataka kao što su: MySQL, PostgreSQL ili Oracle, a koje podržava Django *web framework*. U nastavku je dan primjer kako se radi Django blog aplikacija na verziji Django 2.0 koja radi na Linux operacijskom sustavu:

Najprije je potrebno provjeriti instalaciju Pythona u terminalu pomoću naredbe `python3 -v`. Za verziju Django 2.0 potrebna je minimalna verzija Pythona 3.4. (Django Instalacija). Sljedeći korak je kreiranje direktorija u kojem će se izraditi projekt. Preko terminala je potrebno upisati `cd /home` te nakon toga kreirati folder django-apps naredbom `mkdir django-apps`. Za ulaz u kreirani direktorij potrebno je upisati naredbu `cd django-apps`. Unutar direktorija se kreira virtualno okruženje pod nazivom *myblog-env* naredbom `virtualenv myblog-env` te se aktivira naredbom `source myblog-env/bin/activate`. Nakon aktivacije virtualnog okruženja, sam njegov naziv mora biti vidljiv u zagradama u

sljedećem retku terminala, a to izgleda: `(myblog-env) antun@ubuntu:~/django-apps$`. Sljedeća naredba je instalacija Django-a naredbom `pip install django` (Morris, 2018).

Naredbom `django-admin.py startproject myproject` kreira se projekt pod nazivom *myproject*, nakon te naredbe kreira se folder s istim imenom te se u njega ulazi naredbom `cd /myproject`, a aplikaciju unutar projekta kreiramo naredbom `python manage.py startapp myblog`. Trenutna struktura projekta izgleda kao:

```
myproject/
    manage.py
    myproject/
        __init__.py
        settings.py
        urls.py
        wsgi.py

    myblog/
        __init__.py
        admin.py
        migrations/
            __init__.py
        models.py
        tests.py
        views.py
```

Postavke samog projekta su smještene u *settings.py* datoteci koja se nalazi u folderu *myproject* u projektu *myproject*. U *settings.py* datoteci potrebno je napraviti nekoliko promjena nad postavkama poput: vremenske zone, baze podataka, statičkih ruta, dopuštenih aplikacijama i sl. Vremenska zona po zadanoj vrijednosti izgleda `TIME_ZONE = 'America/New_York'`, pa ju je potrebno izmijeniti u `TIME_ZONE = Europe/Zagreb'`. Sljedeća promjena tiče se postavke baze podataka. Ako se odabere PostgreSQL baza podataka, promjena izgleda kao na kodu 6. (Morris J. 2018).

```
from django.db import models
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Kod 6. Postavke baze podataka za PostgreSQL u Django postavkama

Nakon promjene podataka za bazu podataka potrebno je omogućiti Djangu da radi s PostgreSQL-om naredbom `pip install django psycopg2`. Sljedeća promjena se tiče dodavanja dopuštenih aplikacija projektu. Kreiranu aplikaciju *myblog* potrebno je dodati u listu dopuštenih aplikacija, a to izgleda kao na kodu 7. (Kumar Shil, 2015).

```
INSTALLED_APPS += (
    'myblog',
)
```

Kod 7. Aktiviranje aplikacije u Djangu

Idući korak odnosi se na kreiranje modela baze podataka za aplikaciju *myblog*, a taj se model nalazi u glavnom projektu u folderu aplikacije pod nazivom *myblog* u datoteci *models.py*. Primjer modela baze prikazan je na kodu 8. (Kumar Shil, 2015).

```
from django.db import models

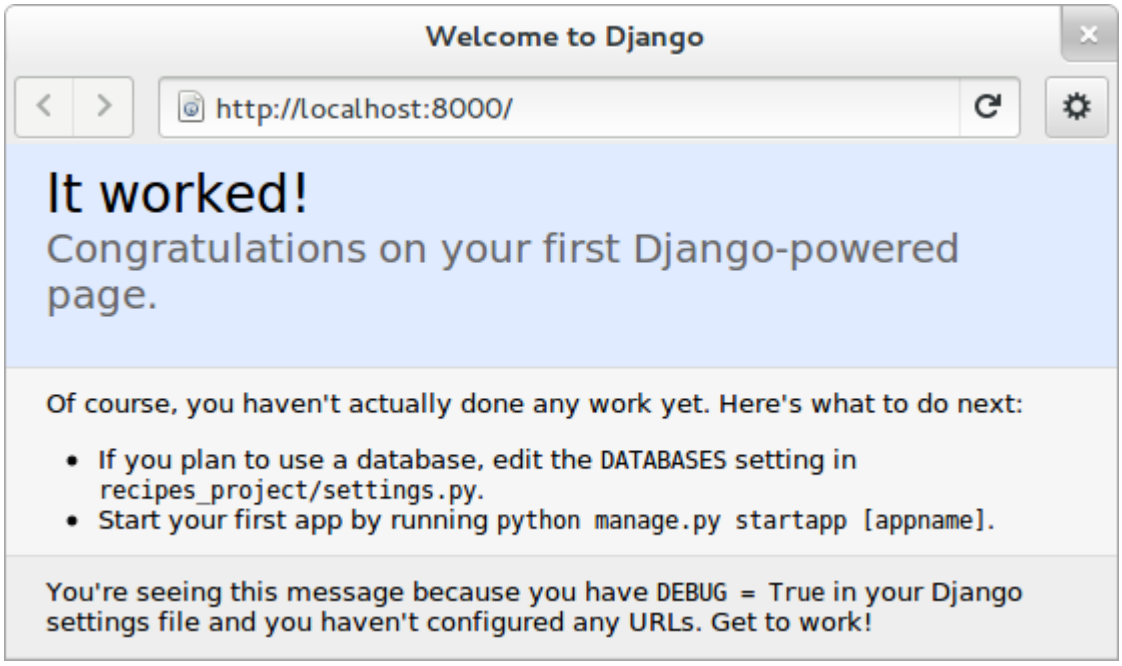
class Tag(models.Model):
    name = models.CharField(max_length=255)
    description = models.CharField(max_length=255, null=True, default='')
    def __str__(self):
        return self.name

class MyBlog(models.Model):
    title = models.CharField(max_length=255)
    body = models.CharField(max_length=20000)
    tags = models.ManyToManyField(Tag)
    def __str__(self):
        return self.title
```

Kod 8. Primjer modela baze podataka u Djangu

Kada je model kreiran, potrebno je napraviti migraciju modela baze podataka na pravu bazu podataka, odnosno, potrebno je kreirati tablice u bazi s imenima klasa modela, u ovom slučaju to su *Tag* i *MyBlog*. Prva naredba za migraciju je `python manage.py makemigrations` koja kreira datoteku po kojoj se formira baza podataka, a druga naredba je `python manage.py migrate` koja radi izmjene baze podataka na osnovu kreirane datoteke za migraciju. Ako je migracija uspješna, kreira se *Super user* odnosno korisnik koji ima administratorske ovlasti, naredbom `python manage.py createsuperuser` te se prate upute kreiranja (Kumar Shil, 2015).

Nakon što je kreiran administratorski korisnik, pokretanje servera za projekt *myproject* u terminalu iz glavnog foldera *myproject* vrši se upisivanjem naredbe `python manage.py runserver 0.0.0.0:8000`. Ako prilikom pokretanja servera nije nastala nikakva greška, u *web* pregledniku na poveznici http://127.0.0.1:8000 trebala bi biti vidljiva stranica projekta *myproject* kao na slici 25.



Slika 25. Stranica dobrodošlice u Djangu (Kumar Shil, 2015)

U glavnom projektu *myproject* u mapi *myproject* nalazi se datoteka urls.py u kojoj se dodaju putanje za admin sučelje i *myblog* aplikaciju. Početna stranica projekta je aplikacija *myblog* zahvaljujući naredbi `url(r'^$', ListView.as_view(model = MyBlog,`

`template_name = 'blog_list.html'), name='blog_list'),` koja izgleda kao na primjeru koda 9 (Kumar Shil, 2015).

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover() #this line is for making model visible in admin site

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls), name='admin-site'),
    url(r'^$', ListView.as_view(model = MyBlog, template_name =
'blog_list.html'), name='blog_list'),
    )
```

Kod 9. URL putanje u Djangu

Prilikom učitavanja stranice, putanja ide prema *blog_list.html* datoteci. U folderu aplikacije *myblog* potrebno je izmijeniti *admin.py* datoteku tako da se admin sučelju daje mogućnost upravljanja *myblog* aplikacijom kao na kodu 10. (Kumar Shil, 2015).

```
# Location myproject>myblog>admin.py
# Register your models here.

from django.contrib import admin
from django import forms

from myblog.models import MyBlog, Tag

admin.site.register(MyBlog)
admin.site.register(Tag)
```

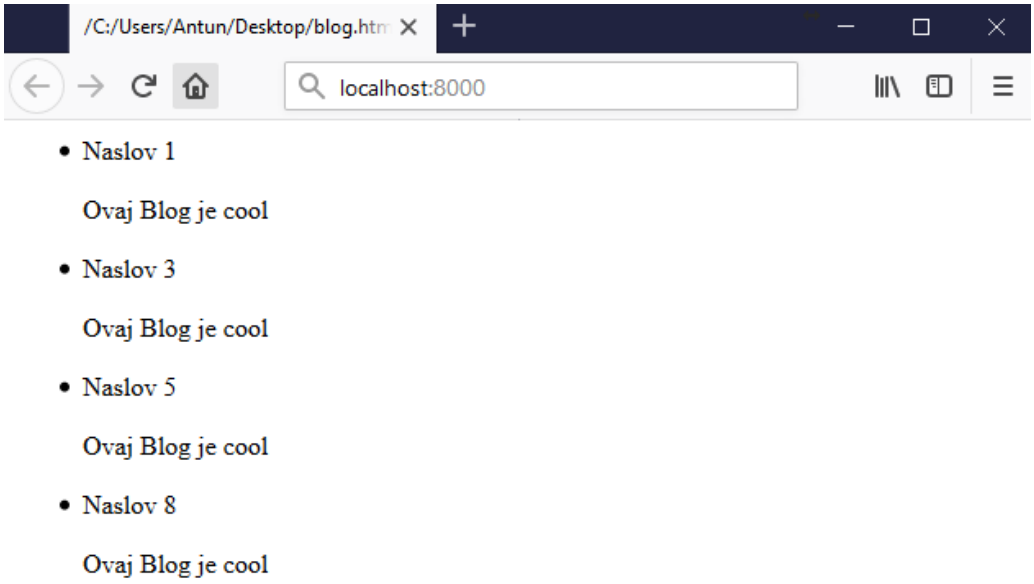
Kod 10. Registriranje aplikacije u Django administracijsko sučelje

Završna osnovna funkcionalnost blog aplikacije je ta da je vidljiv naslov i kontekst samog članka, a to se postiže tako što se unutar *blog_list.html* datoteke unese kod kao na primjeru koda 11.

```
<ul>
    {% for blog in object_list %}
        <li> {{ blog.title }} </li> <br/>
        <p> {{ blog.body }} </p><br/>
    {% endfor %}
</ul>
```

Kod 11. Ispis liste unutar HTML-a

Rezultat je stranica koja je prikazana na slici 26.



Slika 26. Lista naslova i tekstova bloga

Struktura samog projekta sastoji se od glavnog direktorija koji je ujedno i ime projekta. U glavnom direktoriju nalaze se direktoriji s imenima aplikacija, a u ovom slučaju je riječ o jednom direktoriju pod imenom *myblog*. Svaka aplikacija sadrži:

- **admin.py** datoteku - služi za interakciju aplikacije s administratorskim sučeljem, odnosno stvara se veza s modelom aplikacije.
- **models.py** datoteku - koristi se za kreiranje modela podataka aplikacije, na osnovu modela kreira se datoteka "0001_initial.py" u direktoriju **migrations**.
- **migrations** direktorij - sadrži pretvorene modele aplikacije u kod po kojem se vrši migracija na bazu podataka. Nakon izrade ili nadogradnje modela u datoteci models.py, potrebno je izvršiti naredbu `$ python manage.py makemigrations` za kreiranje migracijske datoteke koja se zove "0001_initial.py".
- **tests.py** datoteku - koristi se za testiranje funkcija unutar aplikacije pomoću testova jedinica[10]
- **views.py** datoteku - datoteka u kojoj se nalazi logika same aplikacije.

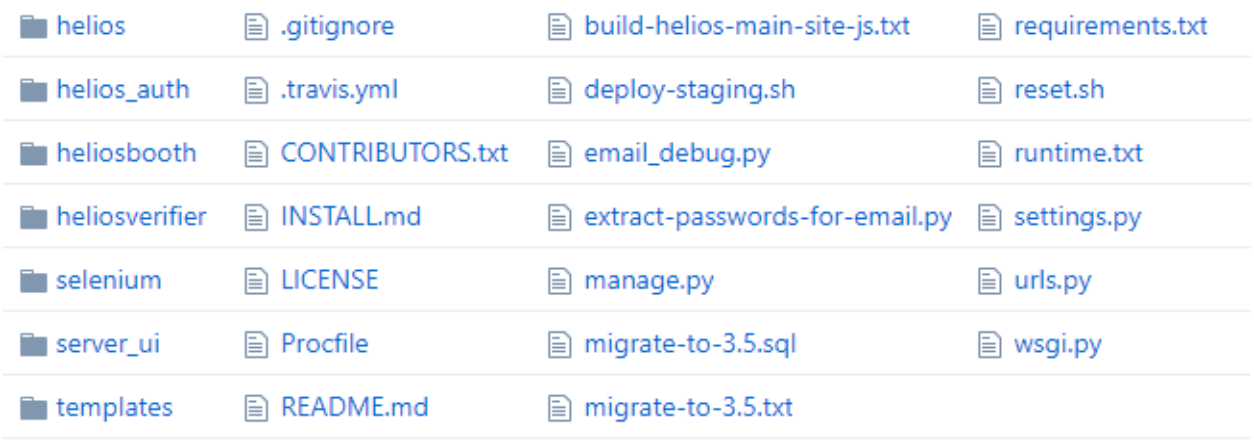
[10] Test jedinice (*Unit test*) - je metoda testiranja softvera pomoću koje se pojedinačne jedinice izvornog koda, skupovi jednog ili više modula računalnih programa zajedno s pridruženim kontrolnim podacima, postupcima korištenja i operativnim postupcima, testirani su kako bi utvrdili jesu li prikladni za upotrebu.

Projekt sadrži datoteku baze podataka **db.sqlite3** kao zadanu bazu podataka, ako se koristi neka druga baza, spomenutu je datoteku potrebno obrisati. Datoteka **manage.py** koristi se za naredbene administrativne zadatke unutar projekta. Također, unutar projekta se nalazi direktorij istog imena kao i glavni direktorij unutar kojega su datoteke:

- **settings.py** - postavke projekta,
- **urls.py** - rute unutar aplikacije,
- **wsgi.py** - pristupna točka za *WSGI* kompatibilne *web* poslužitelje za posluživanje projekta.

3.1.2.3 Django i Helios

Helios se sastoji od pet aplikacija od kojih su *helios*, *helios_auth*, *server_ui*, *heliosbooth* i *heliosverifier* što je vidljivo na slici 27.

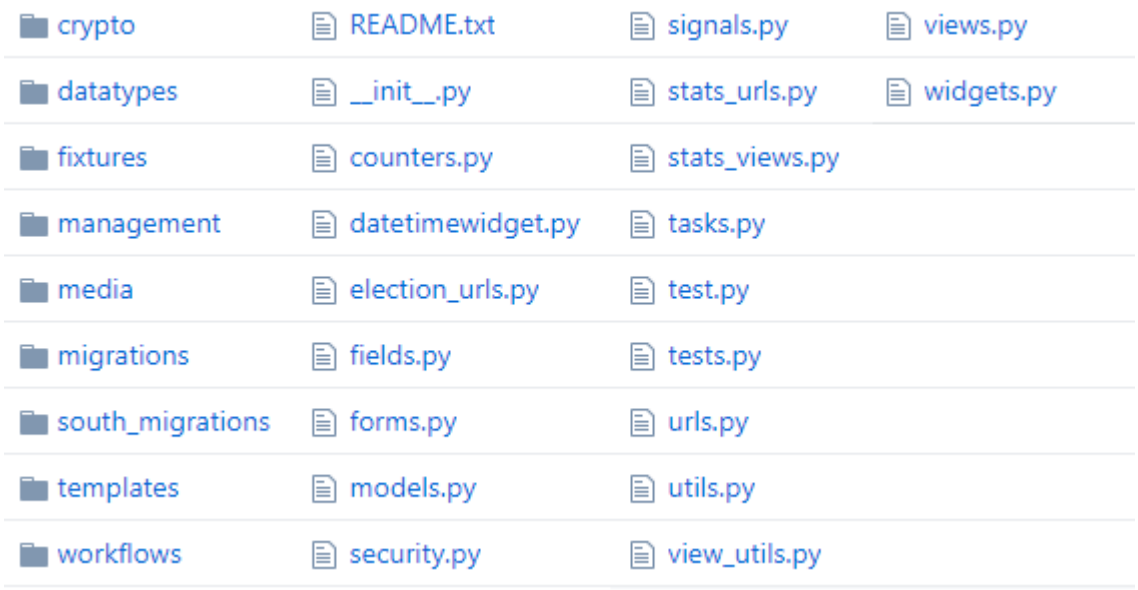


Slika 27. Struktura Helios projekta

Aplikacija *helios* je glavna aplikacija koja sadrži kriptografske algoritme, glavnu logiku cijelog projekta i sve prikaze stranica (iz foldera templates) čija je struktura prikazana na slici 28. Mapa *crypto* sadži python datoteke poput:

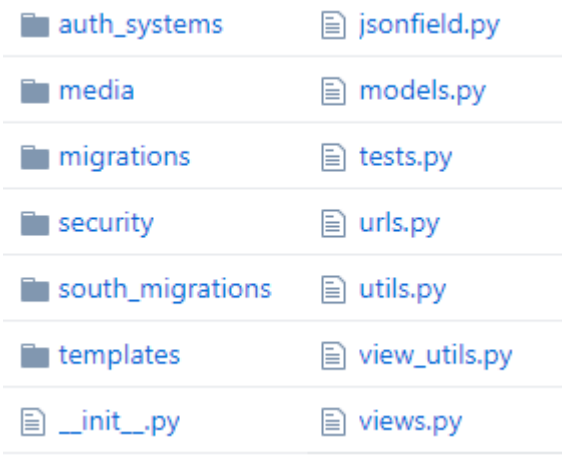
- algs.py,
- elgamal.py u kojem je napisan ElGamal algoritam,
- randpool.py za generiranje slučajnih brojeva,
- electionalgs.py u kojem su funkcije za ostale kriptosustave koje koristi Helios,
- numtheory.py u kojem su matematičke funkcije od kojih su neke poput:
    - „najveći zajednički djelitelj (gcd)“,

- „Faktorizacija cijelog broja“,
- „Linearna jednadžba modulo n“ i sl.



Slika 28. Struktura helios aplikacije

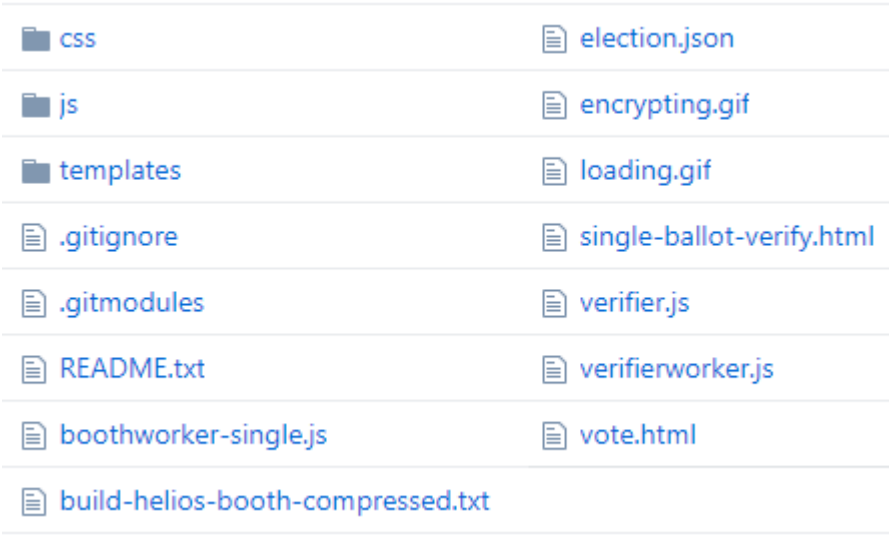
Aplikacija pod nazivom *helios_auth* je podrška za nekoliko različitih autentifikacija na Helios projektu koje omogućuju prijavu na vanjske servise poput: *Facebooka*, *Google +*, *Clevera*, *Linkedina*, *Twittera* i *Yahooa*. Struktura je prikazana na slici 29.



Slika 29. Struktura helios_auth aplikacije

Aplikacija *server_ui* prikazuje početnu stranicu i sve popratne opisne stranice kao što su: kontakt, česta pitanja, o nama i sl.

Aplikacija *heliosbooth* je JavaScript aplikacija kojoj se pristupa na temelju parametra *election_url*, odnosno poveznice za određeni izbor čija je vrijednost veza na API[11] . Rezultat toga su podaci za izbore s identifikatorom u JSON formatu. Na temelju tih podataka izrađuje se obrazac za glasovanje. Struktura *heliosbooth* aplikacije je na slici 30.



Slika 30. Struktura heliosbooth aplikacije

Aplikacija *heliosverifier* je JavaScript aplikacija koja provjerava ispravnost glasačkog šifriranja.

3.1.3 PostgreSQL

PostgreSQL je vrlo moćan i pouzdan, *open source,* objektno-relacijski sustav baze podataka. PostgreSQL moguće je pokrenuti na poznatim operacijskim sustavima uključujući: Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, macOS, Solaris, Tru64) i Windows. PostgreSQL ima nativna programska sučelja za C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC i mnoga druga. Vrlo je skalabilan u broju istodobnih korisnika koje može ugostiti. Neke opće granice PostgreSQL nalaze se u tablici 4 (PostgreSQL) .

[11] API (engl. *application programming interface*) je aplikacijsko programsko sučelje pomoću kojeg programeri imaju pristup nekoj aplikaciji. Slijedom određenih pravila mogu dobivati, ažurirati, brisati i stvarati nove objekte u aplikaciji. Najčešći korišteni formati teksta koji se koriste u API-u su JSON I XML.

Tablica 4. Ograičenja PostgresSql baze (PostgreSQL)

Ograničenje (Limit)	Vrijednost (Value)
Maksimalna veličina baze podataka	Neograničeno
Veličina tablice	32 TB
Maksimalna veličina reda	1.6 TB
Maksimalna veličina polja	1 GB
Maksimalno redova po tablici	Neograničeno
Maksimalno stupaca po tablici	250 - 1600, ovisno o vrsti stupca
Maksimalno indeksiranje po tablici	Neograničeno

SQL standard definira četiri razine izolacije transakcija. Razine izolacije su pravila po kojima se pristupa redovima tablice unutar baze podataka u isto vrijeme. Prva razina izolacije „Read uncommitted“, obično se koristi samo za čitanje jer su promjene koje naprave druge transakcije vidljive i dostupne odmah. Druga razina „Read Committed“ dopušta pristup redovima na kojima su završene druge transakcije, to sprječava čitanje bilo kakvih "prljavih" podataka. Treća razina „Repeatable read“ čeka završetak druge transakcije da bi nastavila. To također sprječava čitanje bilo kakvih "prljavih" podataka. Četvrta razina „Serializable“ dopušta da se istodobno vrše dvije ili više transakcija, a da je rezultat isti kao da je izveden sekvencijalno (IBM). PostgresSql po zadanim postavkama ima namještenu izolaciju „Read uncommitted“. Tablica 5. prikazuje razine izolacije i njihove sklonosti greškama iz koje je vidljivo da razina „Serializable“ pruža najstrožu izolaciju transakcija (PostgreSQL *Transaction Level*).

Tablica 5. Razine izolacije i njihove sklonosti greškama (PostgreSQL *Transaction Level*)

Razina izolacije	Prljavo čitanje	Neponovljivo čitanje	Fantomsko čitanje	Serialization Anomaly
Read uncommitted	Dopušteno, ali ne u PG	Moguće	Moguće	Moguće
Read committed	Nemoguće	Moguće	Moguće	Moguće
Repeatable read	Nemoguće	Nemoguće	Dopušteno, ali ne u PG	Moguće
Serializable	Nemoguće	Nemoguće	Nemoguće	Nemoguće

PostgreSQL baza usklađena je po *ANSI-SQL:2008* standardu. Podržava razine za izolaciju transakcija koje se čitaju i serijaliziraju. PostgreSQL je relacijski sustav koji sam po sebi podržava više shema po bazi podataka. Značajke integriteta podataka obuhvaćaju primarne ključeve i strane ključeve, od kojih strani ključevi mogu biti s ograničavanjem, kaskadnim ažuriranjima/brisanjima, provjeravanjem ograničenja i jedinstvenim ograničenjima. PostgreSQL podržava složene, jedinstvene, djelomične i funkcionalne indekse, oni mogu koristiti bilo koji od njegovih B-stabala, R-stabla, *hash* ili GiST metoda pohrane (PostgreSQL).

U nastavku je objašnjeno kako instalirati i konfigurirati PostgreSQL za korištenje u Django aplikacijama. Potrebno je: instalirati softvere koji su nužni za normalan rad PostgreSQL-a, stvoriti vjerodajnice baze podataka za aplikaciju, a zatim promijeniti postavke za bazu podataka unutar Django projekta. U ovom objašnjenju instalacije korišten je Ubuntu[12] operacijski sustav. Prilikom instalacije potrebno je biti prijavljen u operacijski sustav kao glavni korisnik, odnosno *root* korisnik. To se postiže tako da se upisuje riječ “sudo” ispred svake naredbe u komandnoj liniji. Ubuntu operacijski sustav

[12] Ubuntu – je operacijski sustav izveden od Debian operacijskog sustava koji je baziran Unixu operacijskom sustavu. Ubuntu za osobna računala sadrži grafičko korisničko sučelje. (engl. graphical user interface, GUI).“Ubuntu je drevna afrička riječ koja podrazumijeva rad "za dobrobit društva". To također znači "Ja sam ono što jesam zbog onoga tko smo svi". Ubuntu operativni sustav donosi duh u Ubuntu svijet računala.” https://www.ubuntu.com/about/about-ubuntu

posjeduje repozitorij s velikim brojem paketa, softvera i aplikacija koje se instaliraju iz komandne linije što je detaljno opisano u 6. poglavlju, na primjeru koda 38.

Nakon uspješne instalacije potrebno je ući unutar komande linije u PostgreSQL bazu i prvo izraditi bazu podataka za Django projekt. Svaki projekt bi trebao imati vlastitu izoliranu bazu podataka iz sigurnosnih razloga.

Potrebno je kreirati korisnika baze podataka koji će se koristi za povezivanje i interakciju s bazom podataka. Poželjno je postaviti što sigurniju lozinku koristeći velika i mala slova, brojeve i znakove. Naredba za kreiranje korisnika je `CREATE USER HELIOS WITH PASSWORD 'password';`.

Lozinka se sprema u `pgpass.conf` datoteku u formatu `hostname:port:database:username:password`. Na *Unix* sustavima treba voditi računa o ovlastima nad `pgpass.conf` datoteci. Potrebno je zabraniti pristup bilo kojem korisniku ili grupi, a to je moguće postići naredbom `chmod 0600 ~ / .pgpass`.

Nakon toga, potrebno je izmijeniti nekoliko parametara veze za korisnika koji je izrađen za ovaj projekt. To će ubrzati operacije baze podataka tako da ispravne vrijednosti ne moraju biti upitane i postavljene svaki put kada se uspostavlja veza. Postavlja se zadano kodiranje UTF-8 naredbom `ALTER ROLE HELIOS SET client_encoding TO 'utf8';`, koju Django očekuje. Također postavlja se zadana shema izolacije transakcija za "read committed" pomoću naredbe `ALTER ROLE HELIOS SET default_transaction_isolation TO 'read committed';`, što blokira čitanje od neizvršenih transakcija. Potrebno je postaviti vremensku zonu naredbom `ALTER ROLE HELIOS SET timezone TO 'UTC';`. Prema zadanim postavkama, Django projekt postavljen je na UTC. Dodavanje prava korisniku *helios* prema bazi *helios* dodaje se naredbom `GRANT ALL PRIVILEGES ON DATABASE HELIOS TO HELIOS`. Ako su uspješno kreirani novi korisnik i baza podataka te dodane ovlasti novog korisnika na istu bazu, potrebno je podatke korisnika i baze unijeti u postavke Django projekta i izvršiti migraciju. Ako dolazi do grešaka prilikom migracije, postoji mogućnost da: podaci za bazu nisu dobro uneseni, da nije instaliran paket za korištenje PostgreSQL baze na Django projektu ili da korisnik računala na kojem se projekt nalazi nema pravo koristiti PostgreSQL bazu podataka. Postoje i grafička sučelja za upravljanje PostgreSQL bazom poput pgAdmin III, Team PostgreSQL i sl. (Ellingwood, 2016).

3.1.4 RabbitMQ

RabbitMQ je softver za razmjenu poruka otvorenog koda koji je usmjeren na poruke. RabbitMQ ima svoj napredni protokol za usklađivanje poruka (AMQP) i podržava Streaming Text Oriented Messaging Protocol (STOMP) i MQTT protokole. RabbitMQ poslužitelj napisan je u programskom jeziku Erlang i izgrađen je na okviru Open Telecom platforme za klasteriranje i *failover*. Klijentske biblioteke za sučelje s brokerom dostupne su za sve glavne programske jezike i poznate *frameworke*, a to su: Java, Spring Framework, .NET, Ruby, Python, PHP, Objective-C and Swift, Scala, JavaScript, C / C++, Go, Erlang, Haskell, Perl, OCaml, Unity 3D i drugi.

3.1.5 Celery

Celery je distribuirani sustav, odnosno Python paket za obradu golemih količina poruka, pružajući operacijama alate potrebne za održavanje takvog sustava. To je red čekanja s fokusom na obradu u stvarnom vremenu, a također podržava i raspoređivanje zadataka.

> *„Jedan proces Celerya može obrađivati milijune zadataka u minuti, uz latenciju podmilosekundnog povratnog kruga (koristeći dobro optimizirane RabbitMQ, librabbitmq brokere).“*[13] (Celery Doc)

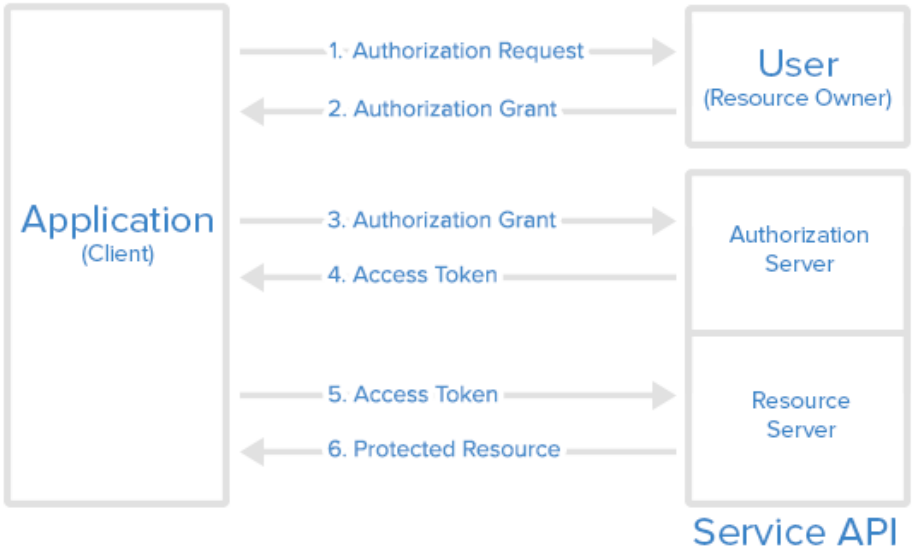
3.1.6 OAuth2

OAuth 2.0 je standard za autorizaciju koji omogućuje aplikaciji treće strane da dobije ograničeni pristup HTTP usluzi, u ime vlasnika resursa i HTTP usluge ili pak dopuštajući aplikaciji treće strane dobivanje pristupa u svoje ime. OAuth 2.0 je poboljšana verzija OAuth protokola koji je objavljen 2006. godine. OAuth 2.0 fokusira se na jednostavnost korištenja, a istodobno pruža specifične procese autorizacije za: *web* aplikacije, aplikacije za stolna računala, mobilne telefone, pametne televizije i sl. Ova se specifikacija i njegova proširenja razvijaju unutar „IETF OAuth Working Group“ (Hardt, 2012).

[13] Vlastiti prijevod

OAuth 2.0 ima nekoliko tipova odobrenja:

- Authorization Code - koristi se s aplikacijama na strani poslužitelja
- Implicit - koristi se s mobilnim aplikacijama ili *web*-aplikacijama (aplikacije koje se pokreću na uređaju korisnika)
- Password - upotrebljava se u pouzdanim aplikacijama, poput onih u vlasništvu same usluge
- Client Credentials - koristi se s pristupom API aplikacijama
- Device Code - upotrebljavaju uređaji bez preglednika ili ulaznih ograničenja uređaja za izmjenu prethodno dobivenog koda uređaja za pristupni token.
- Refresh Token – omogućuje klijentima da nastave imati valjani pristupni token bez daljnje interakcije s korisnikom.



Slika 31. Komunikacija protokola OAuth 2.0 (Anicas, 2014)

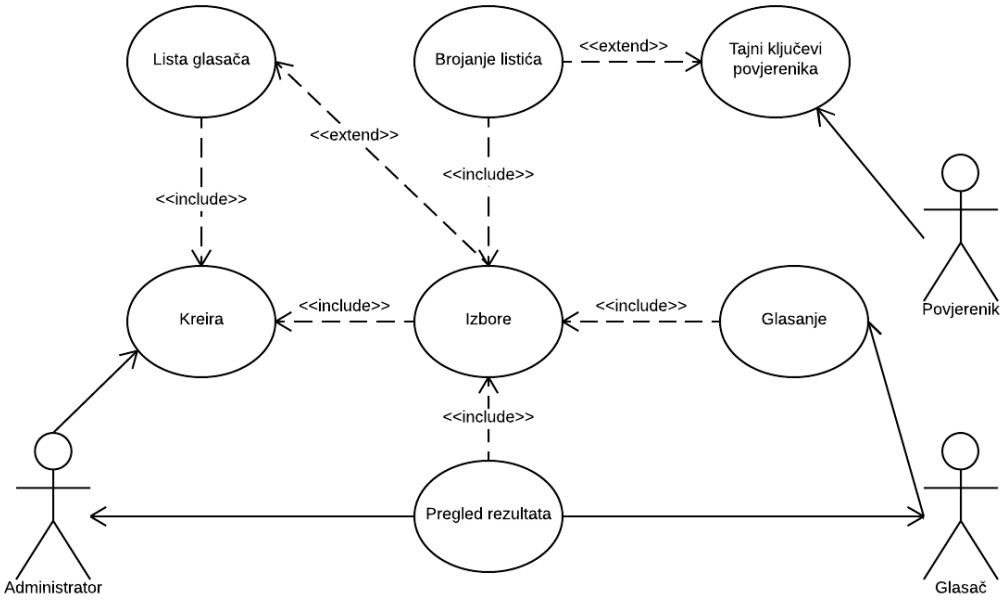
Anicas navodi šest koraka principa rada komunikacijskog protokola OAuth 2.0:

1. Aplikacija traži zahtjev, tj. autorizaciju za pristup resursima usluge od korisnika.
2. Ako je korisnik odobrio zahtjev, aplikacija dobiva odobrenje za autorizaciju.
3. Aplikacija zahtjeva pristupni token s poslužitelja za autorizaciju (API) tako da prikaže autentifikaciju vlastitog identiteta i odobrenja za autorizaciju.
4. Ako je valjano odobrenje za autorizaciju, poslužitelj za autorizaciju (API) izdaje pristupni token aplikaciji.

5. Nakon što je autorizacija gotova, aplikacija zahtijeva resurs s poslužitelja resursa (API) i predstavlja pristupni token za provjeru autentičnosti.
6. Ako je token pristupa točan, poslužitelj resursa (API) služi resursu aplikaciji.

3.3 Dijagrami

Osnovna funkcija sustava Helios je glasovanje, a prikazana je pomoću dijagrama obrazaca uporabe na slici 32.



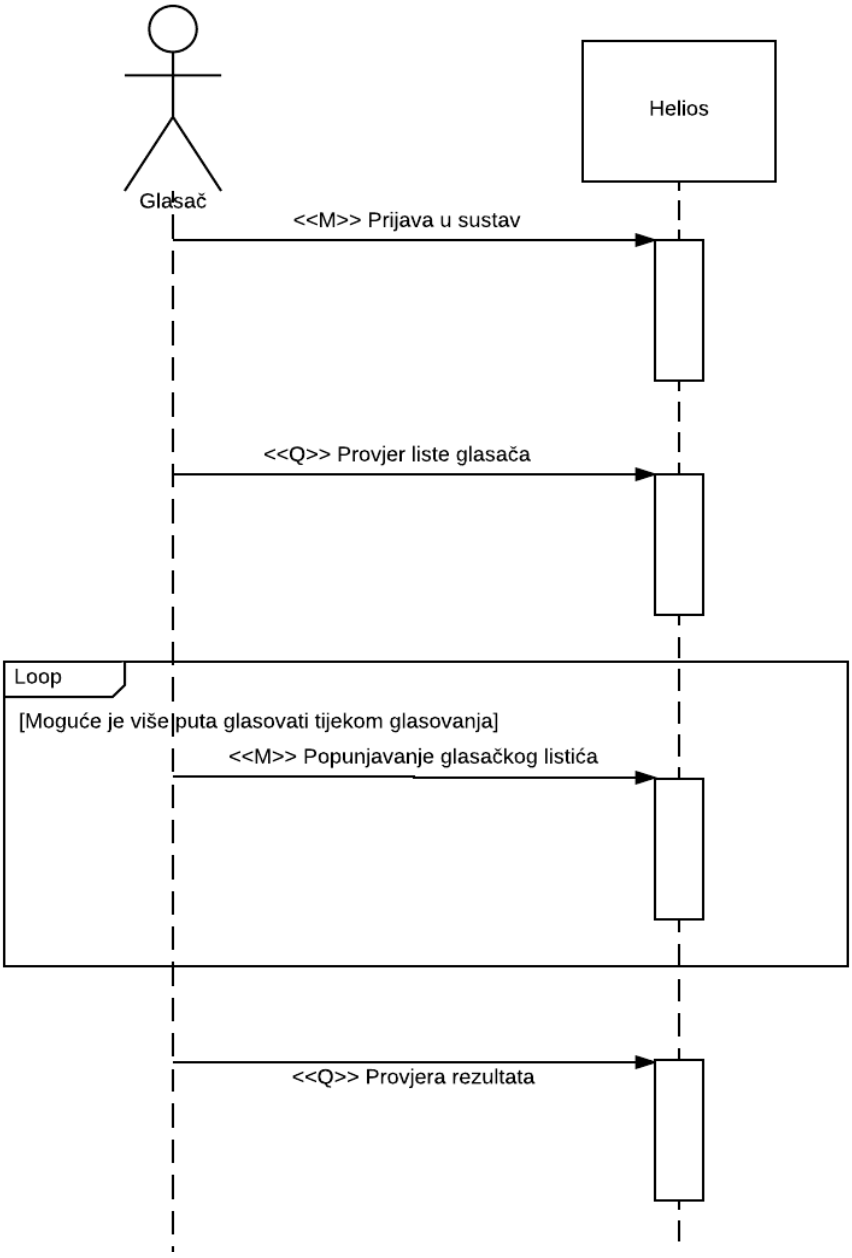
Slika 32. Dijagram obrazaca uporabe (use case) sustava za glasovanje Helios

Koraci kod korištenja sustava Helios:

1. Administrator kreira izbore
2. Administrator kreira listu glasača
3. Nakon otvorenja izbora, glasač glasuje, ima pravo glasovati više puta dok nisu završili izbori, u obzir dolazi zadnji ubačeni listić, svako glasovanje se šifrira
4. Nakon završetka izbora, glasački listići se prebrojavaju uz dešifriranje povjerenikovog tajnog ključa, prema zadanim postavkama povjerenik je sustav Helios, može se dodati nekoliko povjerenika.
5. Nakon prebrojavanja, konačni rezultati vidljivi su na stranici izbora, a šalju se i e-poštom svakom glasaču od strane Helios sustava na zahtjev administratora.

3.3.1 Dijagram slijeda obrazaca uporabe

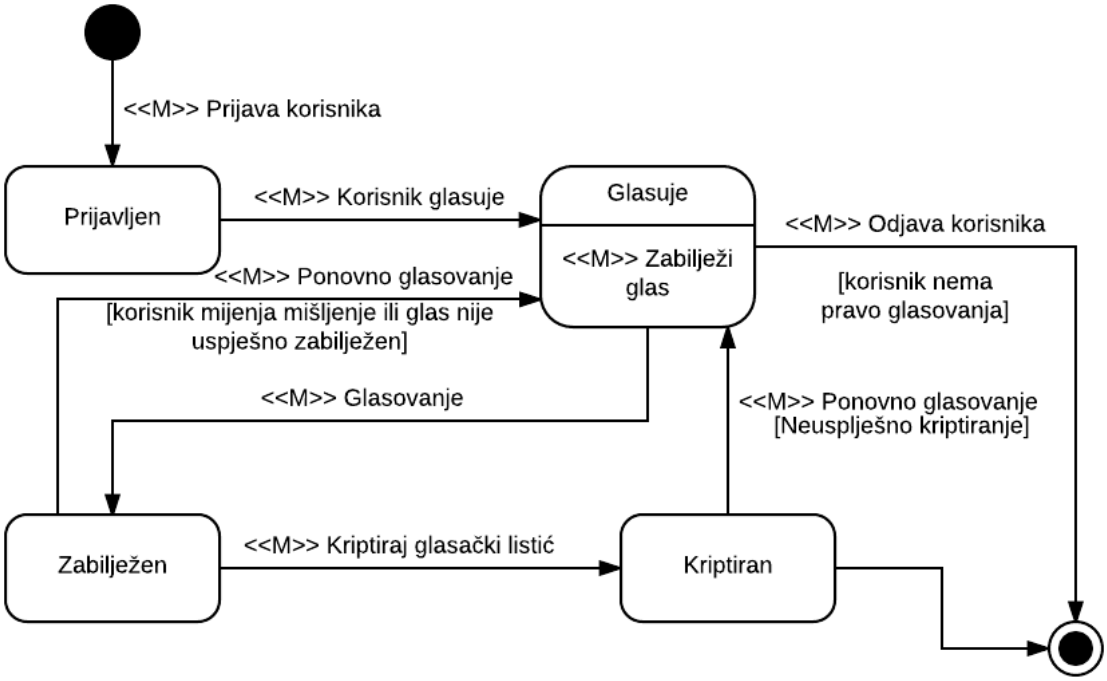
Dijagram slijeda obrazaca uporabe slikovito prikazuje slijed samog glasovanja u sustavu Helios. Slika 33 prikazuje dvije strane koje komuniciraju, a to su Glasač i Helios. Oznaka „<<M>>“ unutar dijagrama označava mijenjanje stanja (eng. mutation), dok oznaka „<<Q>>“ označava upit.



Slika 33. Dijagram slijeda obrazaca uporabe

3.3.2 Dijagram stanja

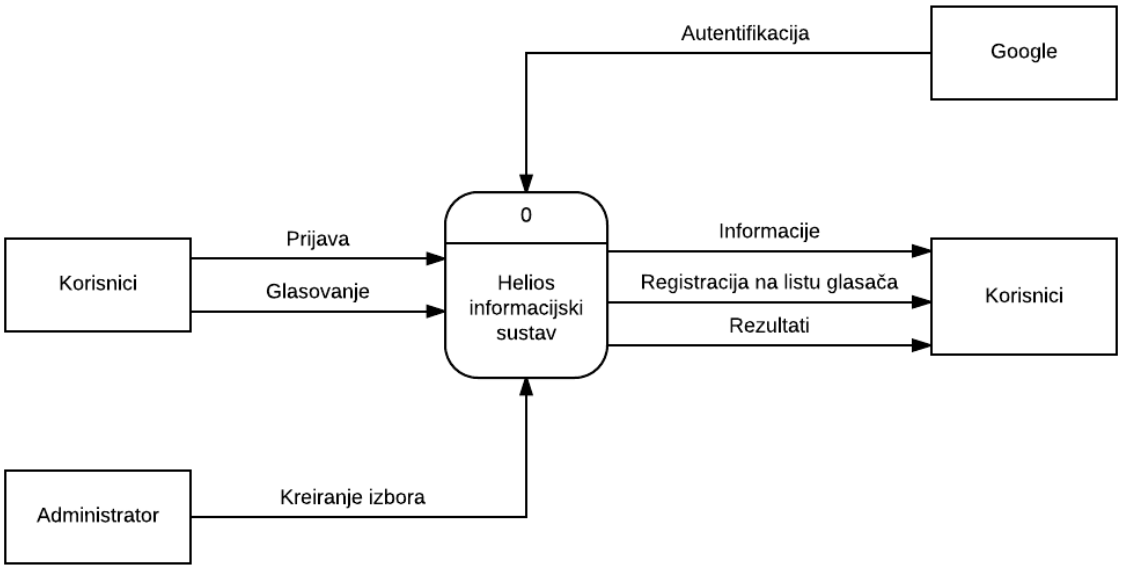
Dijagram stanja na slici 34 prikazuje logiku samog glasovanja u sustavu Helios. Dijagram prikazuje različita stanja u kojima se odlučuje u kojem će smjeru teći glasovanje. Nakon prijave u sustav osoba može glasovati samo ako je na izbornoj listi. Ako je osoba glasovala, a nije zadovoljna svojim odabirom, može ponovno glasovati, listić se zabilježi, kriptira te spremi u bazu podataka. Time je osoba završila svoje glasovanje.



Slika 34. Dijagram stanja

3.3.3 Kontekstualni dijagram

Kontekstualni dijagram (0) nulte razine prikazuje Helios informacijski sustav kao zasebnu cjelinu, a uz to i njegovu komunikaciju s korisnicima i vanjskim servisima (slika 35). Strelice na dijagramu prikazuju u kojem smjeru, po kojem redoslijedu ide komunikacija, primjerice kada Administrator daje zahtjev za kreiranje izbora, tada Helios kreira izbore na osnovu popisa glasača, a zatim svaki glasač dobije e-poštu od Heliosa s informacijama o izborima.



Slika 35. Kontekstualni dijagram kriptografskog informacijskog sustava Helios

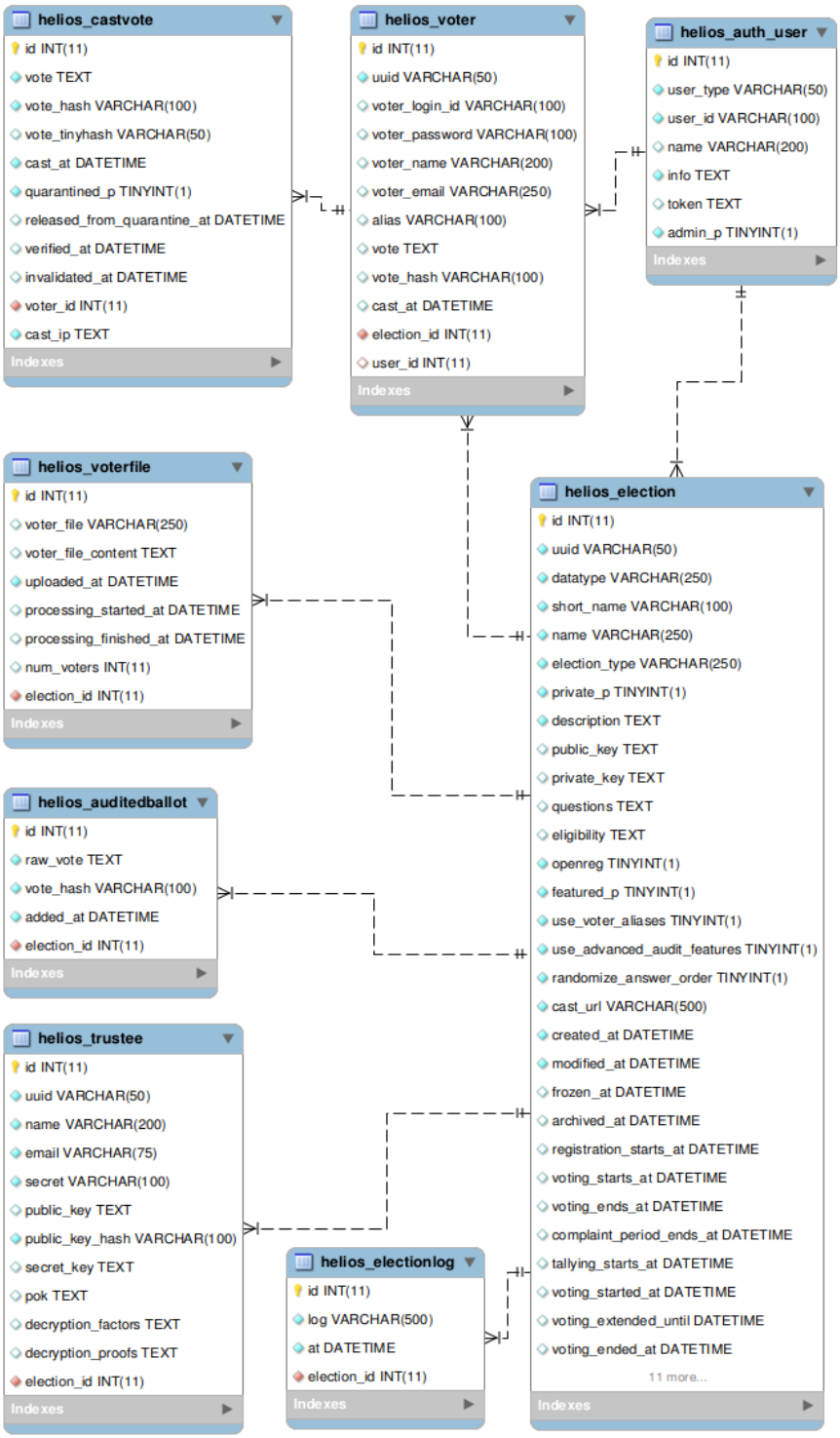
3.4 Relacijski model podataka

Slika 36 prikazuje relacije bitnih Heliosovih tablica. Glavna tablica pod imenom „helios_election“ na sebe veže direktno ili indirektno još sedam tablica. Glavna tablica sadrži najbitnije informacije o izborima poput: imena izbora, opisa, datuma početka i kraja izbora, javnog i privatnog ključa izbora, izborna pitanja i druge. Glavna tablica na sebe veže pet tablica s relacijom jedan ili više (1..*) od kojih su:

- “helios_voter” – Sadrži podatke glasača poput, imena, lozinke, e-pošte i sl.
- “helios_voterfile” – Sadrži popis glasača koji sudjeluju u nekom određenom izboru
- “helios_auditedbalot” – Glasački listići za reviziju
- “helios_trustee” – Povjerenici za izbore
- “helios_electionlog” – Dnevnik događaja za izbore

Dvije tablice koje nisu direktno povezane na glavnu tablicu su:

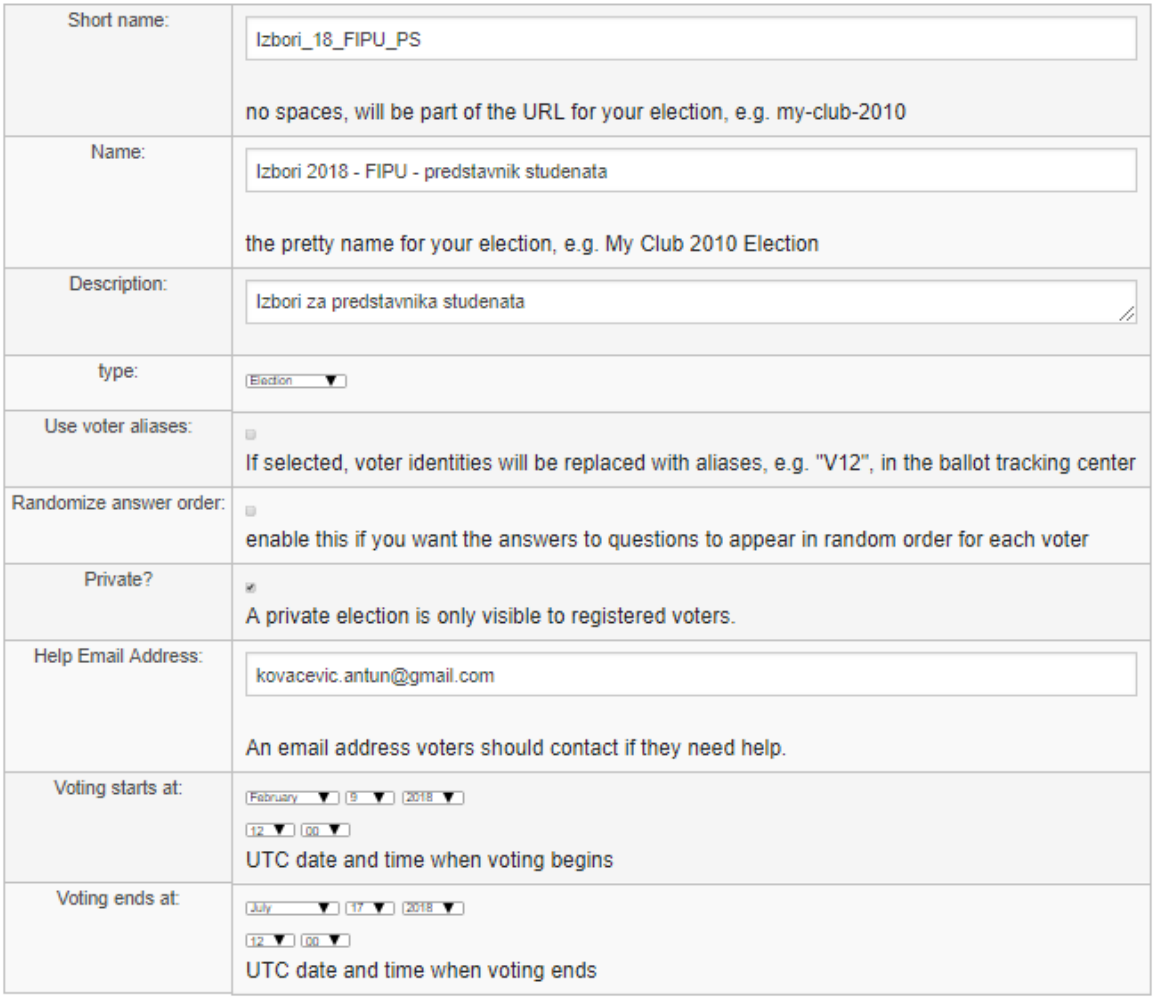
- “helios_auth_user” – Podaci za autentifikaciju korisnika koji se prijavljuju pomoću vanjskih servisa (*Google, Facebook* i ostali)
- “helios_castvote” – Tu se nalaze “ubačeni” kriptirani glasački listići



Slika 36. Relacijski model baze podataka sustava Helios

4. Korisničko korištenje Heliosa

Helios organizira izbore samo za registrirane korisnike. Registracija se obrađuje prijavom pomoću nekih od poznatih javnih servisa kao što su: *Facebook*, *Twitter*, *Google*, *Yahoo*, ili pak pomoću internog Heliosovog sustava za autentifikaciju. Samo administrator može odabrati tip autentikacije, a može dopustiti jedan ili više njih. Nakon uspješne prijave registrirani korisnik ima mogućnost stvaranja izbora s imenom izbora, datumom i vremenom kada se očekuje početak i kraj glasovanja. To se može učiniti tako da se popuni obrazac poput onog na slici 37.

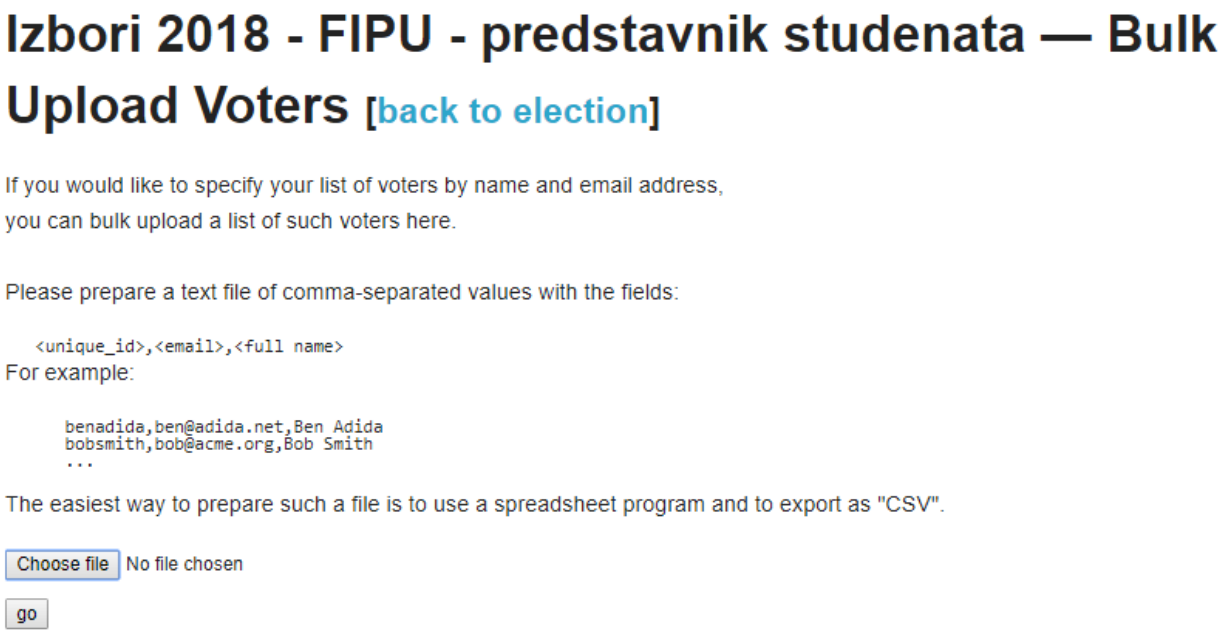


Slika 37. Kreiranje izbora u Heliosu

Nakon stvaranja izbora, Helios generira i pohranjuje novi ključ ElGamala za te izbore. Registriranom korisniku dostupan je samo javni ključ, a Helios čuva privatni ključ. Korisnik koji je stvorio izbore postaje administratorom tih izbora.

4.1. Upravljanje glasačima

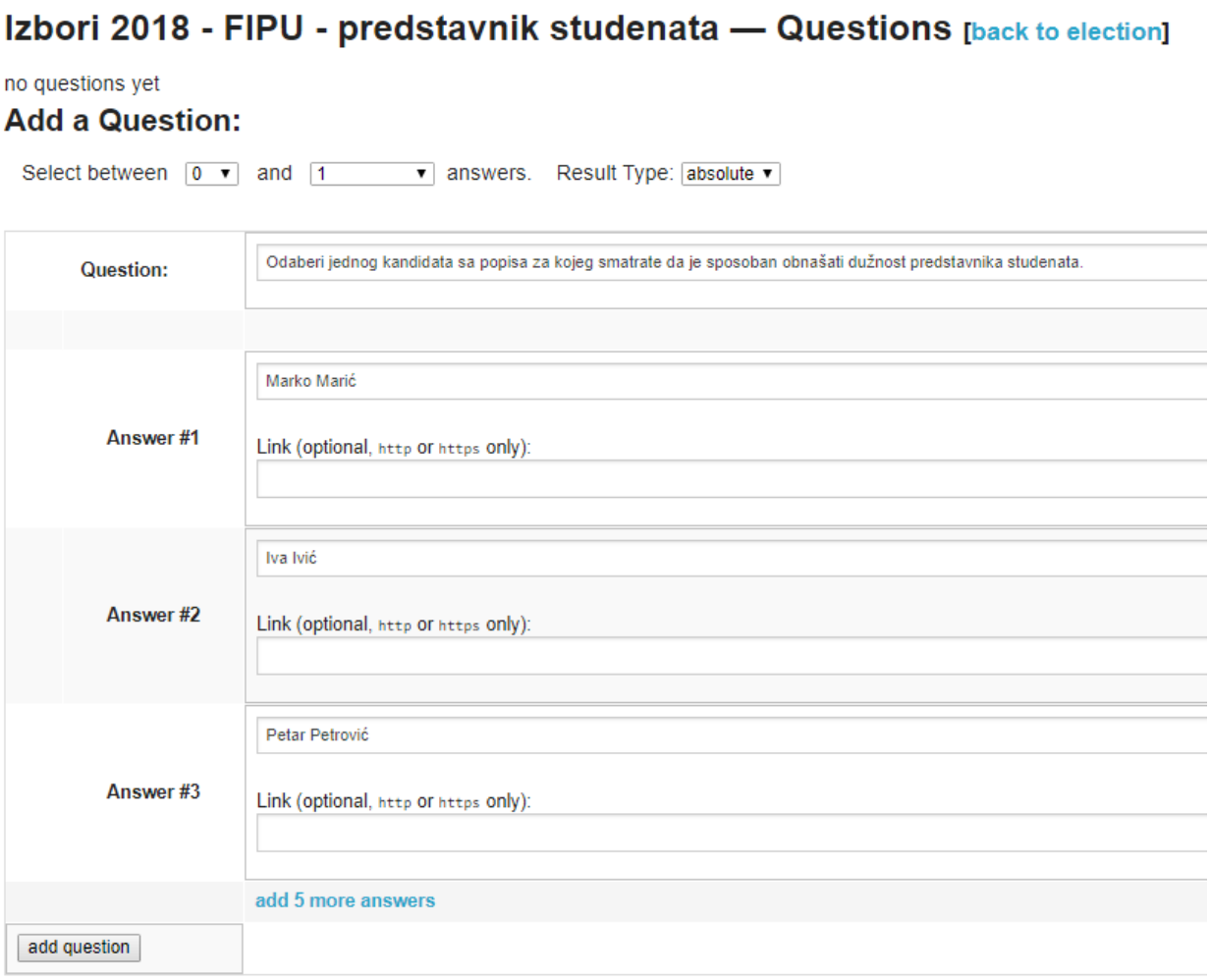
Administrator izbora može dodavati, ažurirati i ukloniti glasače po volji. Popis glasača se unosi u obliku CSV datoteke. Glasač je identificiran imenom i adresom e-pošte, a specifičan je za određene izbore. Helios automatski stvara lozinku od 10 znakova za svakog glasača. Administrator u bilo kojem trenutku može slati poruke glasačima putem Heliosovog administrativnog sučelja. Poruka automatski sadrži lozinku glasača koju administrator nije u mogućnosti vidjeti. Slika 38 prikazuje mjesto gdje se dodaje datoteka s popisom glasača i primjer formata CSV datoteke.



Slika 38. Unos popisa birača u sustavu Helios

4.2. Priprema pitanja/listića

Nakon što je lista glasača unesena, potrebno je pripremiti glasački listić. Forma za unos pitanja i odgovora izgleda kao na slici 39. Za svako pitanje može se dodati broj mogućih odgovora po želji, a može se izabrati i minimalan i maksimalan broj odgovora. Po početnim postavkama tip rezultata je apsolutan, no on može se promijeniti u relativan. Pod apsolutan tip rezultata spada onaj kod kojeg više od 50% glasača odabere isti odgovor. Kod parnog broja glasača formula glasi $broj_glasača\ /\ 2\ +\ 1$ glas, a kod neparnog broja glasača formula glasi $1\ +\ broj_glasača\ /\ 2$.



Slika 39. Unos izbornog pitanja

4.3. Zamrzavanje izbora

Kada su izbori spremni, administrator zamrzava izbore u kojima je popis glasača, datum početka i završetka izbora, a pojedinosti glasača postaju nepromjenjive i dostupne za preuzimanje u obliku JSON-a. Administrator prima poruku e-poštom od strane Heliosa sa SHA2 *hashom* spomenutog JSON-a. Nakon svega toga izbori su spremni za glasovanje. Pomoću Heliosovog administrativnog sučelja Administrator obavještava glasače da su izbori otvoreni.

U nastavku je objašnjen proces glasovanja između Alise kao glasača i BPS-a (eng. Ballot Preparation System), odnosno Sustava za pripremu glasačkih listića:

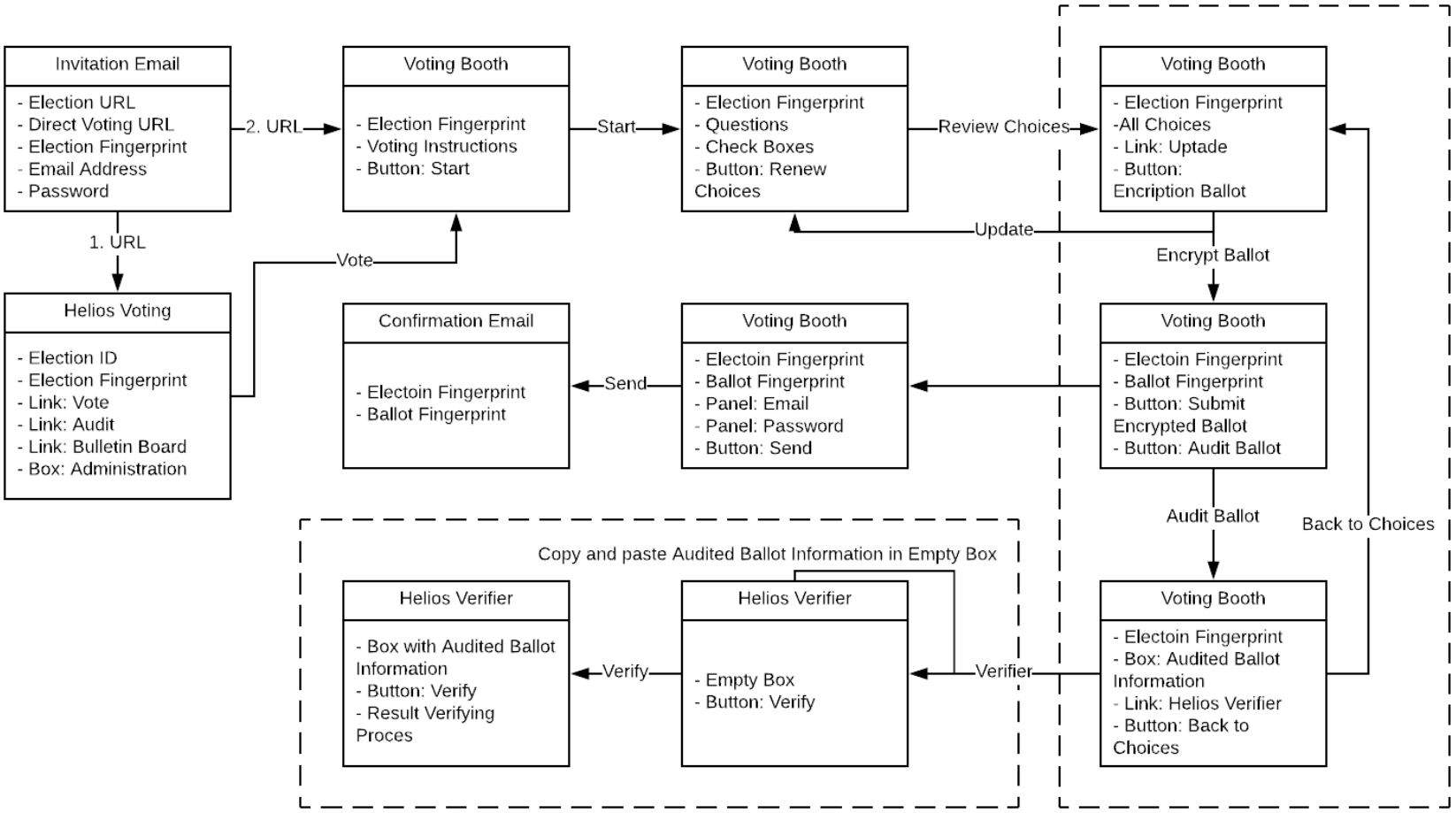
1. Alisa započinje proces glasovanja ukazujući na kojim izborima želi sudjelovati.
2. BPS vodi Alisu kroz sva glasačka pitanja te bilježi njezine odgovore.
3. Nakon što Alisa potvrditi svoje izbore, BPS šifrira izbore i prikazuje *hash* od pripadajućeg šifrata.
4. Alisa sada može odabrati mogućnost revizije glasačkog listića. U tom slučaju BPS prikazuje šifrirani tekst njezinog izbora i tako da Alisi može potvrditi da je ispravno kriptirao izbore. Ako je ova opcija odabrana, BPS od Alise traži da generira novi šifrat svojih izbora.
5. Ako Alisa ima želju promijeniti svoj odabir može ponovno glasovati. U tom slučaju BPS odbacuje sve prijašnje informacije, njihove slučajnosti i jasnoće, ostavljajući samo novi šifrirani tekst koji je spreman za brojanje.

Cijeli protokol glasovanja u Heliosu odvija tako da:

1. Alisa priprema i revidira onoliko glasačkih listića koliko želi. Kada je zadovoljna izborom i uvjerena u sigurnost sustava ubacuje svoj šifrirani glasački listić.
2. Helios oglasna ploča postavlja Alisino ime i njezin šifrirani glasački listić. Svatko može provjeriti oglasnu ploču i pronaći svačiji objavljeni šifrirani glas.
3. Kada se izbori zatvore Helios miješa sve šifrirane glasačke listiće i stvara neinteraktivni dokaz pravilnog miješanja, a odvajajući glasove od imena i miješajući ih dodatno garantira anonimnost glasačkih listića.
4. Nakon određenog razdoblja predviđenog za prigovor, Helios dešifrira i evidentira sve prethodno promiješane glasačke listiće, ali uz to također osigurava i dokazuje sigurnost svakog tog listića.
5. Revizor može preuzeti cjelokupne podatke o izborima i potvrditi miješanje, dešifriranje i podudaranje glasačkih listića.

Na slici 40 prikazan je dijagram korištenja sustava Helios, odnosno objedinjuje procese prethodno objašnjene u potpoglavljima: Priprema pitanja/listića i Zamrzavanje izbora.

-



Slika 40. Dijagram korištenja sustava Helios (Olembo, 2010., str. 6.)[14]

[14] Izmjenio autor

4.4. URI-ovi i primjeri JSON podataka

4.4.1 Revizija

Svi podaci za glasovanje dostupni su putem raznih varijanti HTTP zahtjeva koji zajedno imaju pristup svim podacima iz određenog izbora . Sustav je izgrađen tako da omogućuje statičku pohranu tih podataka na jednostavnom datotečnom sustavu koji je dostupan preko *weba* čime se dugoročno pojednostavljuje robusnost. Veza na određene izbore, na primjer, sadrži *HELIOS_HOST*, također sadrži i */elections/* što znači da se radi o izborima, a naposljetku, sadrži */ELECTION_ID* što predstavlja jedinstveni broj izbora. Taj i ostali primjeri veza URI-ja prikazani su u tablici 6. (Adida, 2012):

Tablica 6. URI-jevi u sustavu Helios (Adida, 2012)[15]

ELECTION_ID	{HELIOS_HOST}/elections/ELECTION_ID
Popis glasača, označen kao VOTER_LIST, dostupan je na:	{HELIOS_HOST}/elections/ELECTION_ID/voters/
Ovaj poziv će servirati listiće kronološki	{HELIOS_HOST}/elections/ELECTION_ID/ballots/
Popis svih glasača koji su glasovali:	{HELIOS_HOST}/elections/ELECTION_ID/ballots/VOTER_ID/all
Zadnji ubačeni listić je:	{HELIOS_HOST}/elections/ELECTION_ID/ballots/VOTER_ID/last
Rezultat izbora je dostupan na:	{HELIOS_HOST}/elections/ELECTION_ID/result
Popis povjerenika:	{HELIOS_HOST}/elections/ELECTION_ID/trustees/

[15] Izmjenio autor (tekst prebačen u tablicu)

Svaki izbor ima jednog ili više povjerenika te popis povjerenika, uključujući javni ključ, faktor za dešifriranje i dokaz.

Svi podaci dostupni su od strane Heliosa u JSON formatu čiji je oblik definiran na temelju dva uvjeta:

1. podaci trebaju biti poredani po abecedi,
2. podaci nemaju dodatnih razmaka (eng. extraneous whitespace).

„Ta dva uvjeta su važna jer će se kreirati isti *hash* samo ako se oni poštuju.“[16] (Adida, 2012).

4.4.2 Kriptomehanizam

Svi veliki cijeli brojevi su predstavljeni kao *base64* stringovi. ElGamal javni ključ sadrži vrijednosti p, q, g i y (po abecednom redu), kao na primjeru kod 12. (Adida, 2012):

```
{
    "g": "Hbb3mx34sd",
    "p": "mN3xc34",
    "q": "J3sRtxcwqlert",
    "y": "U8cnsvn45234wsdf"
}
```

Kod 12. ElGamal javni ključ (Adida, 2012)

ElGamalovom shemom šifrirana je JSON struktura koja sadrži vrijednosti *alpha* i *beta* koje su prikazane na kodu 13.

```
{
    "alpha": "6BtdxuEwbcs+dfs3",
    "beta": "nC345Xbadw3235SD"
}
```

Kod 13. ElGamal kriptirani tekst (Adida, 2012)

Alpha i *beta* iz prethodnog primjera izvode se na temelju navedenih formula čije su vrijednosti u konačnici prikazane u *base64* stringu:

$$\alpha = g^r \bmod p, \qquad \beta = g^m y^r \bmod p$$

[16] Vlastiti prijevod

U Heliosu su sve *hash* vrijednosti temeljene na *base64* stringu i uvijek su *hashirane* pomoću SHA256 kao što je prikazano na primjeru koda 14.

```
{
    "election_hash": "c0D1TVR7vcIvQxuwfLXJHa5EtTHZGHpDKdulKdE1oxw"
}
```

Kod 14. Primjer hash vrijednosti (Adida, 2012)

4.4.3 Glasač

Jedan glasač u Heliosu je predstavljen pomoću nekoliko polja koji utvrđuju identitet glasača (Adida, 2012) kao što je prikazano na primjeru koda 15..

```
{
    "name": "Ben Adida",
    "uuid": "60435862-65e3-11de-8c90-001b63948875",
    "voter_id": "benadida@gmail.com",
    "voter_type": "email"
}
```

Kod 15. Podaci glasača u sustavu Helios s identifikacijom pomoću e-pošte (Adida, 2012)

Polje *uuid* koristi se kao jedinstveni identifikator za izbore u Heliosu, točnije, koristi se u URI-u samog izbora (ELECTION_ID).

Glasači se mogu identificirati pomoću *OpenID* URL-a što je prikazano JSON strukturom na primjeru koda 16.:

```
{
    "name": "Ben Adida",
    "uuid": "4e8674e2-65e3-11de-8c90-001b63948875",
    "voter_id": "http:\/\/benadida.myopenid.com",
    "voter_type": "openid"
}
```

Kod 16. Podaci glasača u sustavu Helios s identifikacijom pomoću OpenID (Adida, 2012)

4.4.4 Zaštita privatnosti glasača

Kod nekih izbora poželjno je da se nikad ne otkrije identitet glasača zbog čega se dodaje zamjensko ime (eng. *alias*), a tada JSON izgleda kao na kodu 17. (Adida, 2012):

```
{
    "alias": "voter_123",
    "uuid": "b7dbd90a-65e3-11de-8c90-001b63948875"
}
```

Kod 17. Podaci za zamjensko ime u Heliosu (Adida, 2012)

Glasač koji je pod zamjenskim imenom i dalje ima isti *uuid*, pa bi se daljnjom analizom i pretragom sustava moglo otkriti o kome se radi (Adida, 2012).

4.4.5 Glasovanje

Kada glasač glasuje, kodirano je zabilježen kao na kodu 18. i 19.

```
{
    "cast_at": "2009-07-15 12:23:46",
    "vote": VOTE,
    "vote_hash": "8bncn23nsfsdk234234",
    "voter_hash": "2bxksdlkxnsdf",
    "voter_uuid": "b7dbd90a-65e3-11de-8c90-001b63948875"
}
```

Kod 18. Ubačeni listić u sustavu Helios (Adida, 2012)

Parametar *cast_at* označuje vrijeme kada je osoba glasovala, a format za vrijeme je (`Y-m-d H:i:s`). Parametar *vote_hash* može omogućiti kraću verziju ove strukture podataka (Adida, 2012):

```
{
    "cast_at": "2009-07-15 12:23:46",
    "vote_hash": "c0D1TVR7vcIvQxuwfLXJHa5EtTHZGHpDKdulKdE1oxw",
    "voter_hash": "2bxksdlkxnsdf",
    "voter_uuid": "b7dbd90a-65e3-11de-8c90-001b63948875"
}
```

Kod 19. Kraća verzija ubačenog listića (Adida, 2012).

4.4.6 Izbori

Izbori se prikazuju kao na kodu 20.

```
{
    "cast_url": "https:\/\/heliosvoting.org\/cast\/",
    "description": "... blah blah blah ... info about the election",
    "frozen_at": null,
    "name": "Student President Election at Foo University 2010",
    "openreg": false,
    "public_key": ELGAMAL_PUBLIC_KEY,
    "questions": QUESTION_LIST,
    "short_name": "fooprez2010",
    "use_voter_aliases": false,
    "uuid": "1882f79c-65e5-11de-8c90-001b63948875",
    "voters_hash": "G6yS\/dAZm5hKnCn5cRgBGdw3yGo"
}
```

Kod 20. Podaci izbora u Heliosu

Parametri *short_name*, *name* i *description* opisuju izbore. Parametar *short_name* je skraćeno ime koje se sastoji od nekoliko znakova bez razmaka, *name* predstavlja ime izbora koje može sadržavati nekoliko riječi, a *description* detaljnije opisuje o kakvom se izboru radi. Parametar *cast_url* pokazuje URL izbora, *frozen_at* označuje vrijeme do kada traju izbori, dok *openreg* pokazuje mogućnost dodavanja glasača na popis nakon što su izbori počeli. Parametar *use_voter_aliases* pokazuje da li su ovi izbori sakrili identitete svojih glasača pomoću skraćenih imena, a *uuid* je jedinstveni identifikator za izbore. Vrijednost parametra *public_key* je *ELGAMAL_PUBLIC_KEY*, El-Gamal javni ključ. Vrijednost parametra *questions* je *QUESTION_LIST* , a to je struktura podataka koja predstavlja popis pitanja (kod 21.) i dostupnih odgovora na ta pitanja (kod 22.) (Adida, 2012).

```
[QUESTION, QUESTION, ...]
```

Kod 21. QUESTION_LIST (Adida, 2012)

Na primjeru koda 22. je prikazan primjer jednog pitanja sa izbora:

```
{
    "answer_urls": [
      "http:\/\/example.com\/alice",
      null
    ],
    "answers": [
      "alice",
      "bob"
    ],
    "choice_type": "approval",
    "max": 1,
    "min": 0,
    "result_type": "absolute",
    "question": "Who Should be President?",
    "short_name": "President",
    "tally_type": "homomorphic"
  }
```

Kod 22. Izborno pitanje, QUESTION (Adida, 2012)

Prikazani primjer koda 22. sadrži dva moguća odgovora, sadrži i URL koji detaljnije opisuje te odgovore, a također sadrži i tekst izbornog pitanja i skraćeni naziv za to pitanje. Parametar *max* označava maksimalni broj opcija koje glasač može odabrati, što je najčešće jedna opcija, a parametar *min* označava minimalan broj opcija koje glasač može odabrati. Parametar *choice_type* ukazuje na vrstu pitanja. Ako parametar *max* ima vrijednost *null*, onda je broj odabira opcija neograničen. Parametar *tally_type* može imati vrijednosti kao što su *homomorphic* ili *mixnet* (Adida, 2012).

Voters_hash je *hash* popisa glasača za izbore. Popis glasača je JSON niz.

```
[
    {
      "id": "benadida@gmail.com",
      "name": "Ben Adida",
      "type": "email",
      "uuid": "60435862-65e3-11de-8c90-001b63948875"
    },
    {
      "id": "ben@adida.net",
      "name": "Ben2 Adida",
      "type": "email",
      "uuid": "4e8674e2-65e3-11de-8c90-001b63948875"
    }
]
```

Kod 23. Lista glasača, VOTER_LIST (Adida, 2012)

Administrator može odabrati između otvorenih i zatvorenih izbora. Kod otvorenih izbora, popis glasača nije potreban jer izborima mogu pristupiti svi. U tom slučaju, struktura izbornih podataka ima vrijednost *null, dok voters_hash* i *openreg* imaju vrijednost *true* (Adida, 2012).

4.4.7 Otisak prsta izbora (eng. fingerprint)

„Nakon što su izbori spremni za glasovanje, administrator zamrzava izbore i time više nije dopušteno mijenjanje parametara i postavki za registraciju glasača: otvoreni izbori ostaju otvoreni za sve, a zatvoreni ostaju otvoreni samo za fiksni određeni popis glasača. Polje *frozen_at* označava vrijeme kada su izbori zamrznuti.“[17] (Adida, 2012)

Izbori koji su zamrznuti mogu biti označeni po Heliosovom otisku prsta, a to je *hash* stvoren od JSON strukture izbora. Heliosov otisak prsta ovisi o popisu glasača ako je zatvoren tip izbora. U svakom slučaju, ovaj otisak prsta ne ovisi o glasovanju ili *hashu* glasovanja (Adida, 2012).

[17] Vlastiti prijevod

4.4.8 Glas

Glasovanje sadrži popis šifriranih odgovora i dvije reference o izborima: *uuid* i *hash*, koji su prikazani na kodu 24.

```
{
    "answers": [ENCRYPTED_ANSWER, ENCRYPTED_ANSWER, ...],
    "election_hash": "Nz1fWLvVLH3eY3Ox7u5hxfLZPdw",
    "election_uuid": "1882f79c-65e5-11de-8c90-001b63948875"
}
```

Kod 24. Listić, VOTE (Adida, 2012).

Svaki "kodiran odgovor" odgovara jednom izbornom pitanju. Svaki odabir sadrži: popis kriptiranih odgovora (jedan za svaki mogući izbor pitanja), popis odgovarajućih dokaza da je šifrirani odgovor ispravno oblikovan te još sadrži i ukupni dokaz da su svi kriptirani tekstovi za to izborno pitanje zajedno pravilno oblikovani (Adida, 2012).

```
{
    "choices": [ELGAMAL_CIPHERTEXT, ELGAMAL_CIPHERTEXT, ...],
    "individual_proofs": [ZK_PROOF_0..1, ZK_PROOF_0..1, ...],
    "overall_proof": ZK_PROOF_0..max
}
```

Kod 25. Kriptirani odgovor, *ENCRYPTED_ANSWER* (Adida, 2012)

Vrijednost *max* u parametru *overall_proof* odgovara vrijednosti *max* iz definicije izbornog pitanja. Kada glasač ubaci listić, Helios kao potvrdu ispisuje glasački otisak prsta (Adida, 2012).

4.4.9 Revizija izbora i dokazi bez znanja

Oznaka *ZK_PROOF_0..max* je dokaz da je odgovarajući šifrirani tekst cijeli broj između *0 i max*, a dolazi pod nazivom homomorfan[18]. U Heliosu, svi *0..max* dokazi su disjunktni dokazi, što znači da transkript sadrži *max + 1* dokaza, po jedan za svaku moguću vrijednost čistog teksta, od *0* preko *max* (Adida, 2012).

```
[ZK_PROOF(0), ZK_PROOF(1), ..., ZK_PROOF(max)]
```

Kod 26. ZK_PROOF_0..max (Adida, 2012)

[18] Homomorfna enkripcija je oblik šifriranja koji omogućuje računanje na šifriranim tekstovima, stvarajući šifrirani rezultat koji, kada se dešifrira, odgovara rezultatu operacija kao da su izvršeni na jasnom tekstu. Svrha homomorfne enkripcije je dopustiti računanje na šifriranim podacima. (Stuntz, 2010)

Jedan ZK[19] dokaz sastoji se od tri argumenata: opredjeljenje, izazov i odgovor. Opredjeljenje se sastoji od dvije vrijednosti, A i B. Primjer ZK dokaza nalazi se na kodu 27. (Adida, 2012)

```
{
    "challenge": "2342342",
    "commitment": {
      "A": "28838",
      "B": "9823723"
    },
    "response": "970234234"
}
```

Kod 27. ZK_PROOF, otvoreni tekst

U Heliosu glasovanje nije obavezno, a dokazati se može samo izazov i odgovor, što značajno smanjuje količinu dokazivanja. To je izvedivo jer obavezne vrijednosti moraju biti nadoknadive kao (Adida, 2012):

$$A = g^{response} / \alpha^{challenge}$$

$$B = y^{response} / (\beta / g^{m})^{challenge}$$

Efektivno, radi se više računanja u zamjenu za puno manji dokaz, jer su A i B u skupini, dok su izazov i odgovor u podskupini (Adida, 2012).

4.4.10 Dokaz

Da bi se stvorio izazov u cjelini, potrebno je stvoriti JSON strukturu podataka kao i kod ostalih JSON objekta u Heliosu (Adida, 2012).

Za dokaz jednostrukog izbora unutar važećeg listića, ubraja se: *election_hash, question_num, choice_num* te kriptirani tekst *chipertext* (kod 28.).

Za ukupni dokaz unutar važećeg listića ubraja se: *election_hash, question_num, chipertext*, gdje je kriptirani tekst homomorfna kombinacija svih odgovora u kriptiranom tekstu. Za dokaz dešifriranja, ubraja se: *chipertext*, *election_hash, trustee_email* (Adida, 2012).

Na primjer, u dokazu jednostrukog izbora generira se izazov u kojem povjerenik mora odgovoriti.

[19] *Zero-knowledge proof* (Dokaz bez znanja) - U kriptografiji je dokaz bez znanja metoda kojom jedna strana može dokazati drugoj stranci da je ta izjava istinita, bez prenošenja bilo kakvih informacija osim činjenice da je izjava zaista istina. (Lexie, 2017)

```
{
    "A": "3bZcd35GAS",
    "B": "7bXcd352sd",
    "choice_num": 0,
    "ciphertext": {
      "alpha": "6BtdxuEwbcs+dfs3",
      "beta": "nC345Xbadw3235SD"
    },
    "election_hash": "Nz1fWLvVLH3eY3Ox7u5hxfLZPdw",
    "question_num": 2
}
```

Kod 28. Dokaz u JSON formatu

4.4.11 Postupak revizije glasovanja

Kada glasač glasuje, svaki kodirani odgovor sadržava dodatne informacije koje se odnose na stvarni odabrani izbor te se u tom slučaju koristi slučajnost za šifriranje svakog izbora odnosno odgovora. Slijedi JSON struktura *VOTE_WITH_PLAINTEXTS* na kodu 29. (Adida, 2012).

```
{
    "answers": [
      "ENCRYPTED_ANSWER_WITH_PLAINTEXT",
      "ENCRYPTED_ANSWER_WITH_PLAINTEXT"
    ],
    "election_hash": "B64_HASH",
    "election_uuid": "ELECTION_UUID"
}
```

Kod 29. VOTE_WITH_PLAINTEXTS

Slijedi šifrirani odgovor s čistim tekstom *ENCRYPTED_ANSWER_WITH_PLAINTEXT* na primjeru u kodu 30.

```
{
    "answer": 1,
    "choices": [
      "ELGAMAL_CIPHERTEXT",
      "ELGAMAL_CIPHERTEXT"
    ],
    "individual_proofs": [
      "ZK_PROOF_0..1",
      "ZK_PROOF_0..1"
    ],
    "overall_proof": "ZK_PROOF_0..max",
    "randomness": [
      "BIGINT_B64",
      "BIGINT_B64",
      "BIGINT_B64"
    ]
}
```

Kod 30. Šifrirani odgovor s čistim tekstom

4.4.12 Rezultat

Rezultat izbora je zastupljen pomoću *RESULT* strukture podataka (kod 31.). Dokazi o dešifriranju vrše se na razini administratora, tj. povjerenika. Rezultat prikazuje broj glasova za svakog kandidata unutar svakog pitanja u formatu polja (Adida, 2012).

```
[
    [QUESTION_1_CANDIDATE_1_COUNT,
    QUESTION_1_CANDIDATE_2_COUNT,
    QUESTION_1_CANDIDATE_3_COUNT],
    [QUESTION_2_CANDIDATE_1_COUNT,
    QUESTION_2_CANDIDATE_2_COUNT]
]
```

Kod 31. Rezultat, RESULT

5. Garancije sustava Helios

5.1 Provjera jedinstvenog glasačkog listića

Chaum-Pedersenov dokaz dokazuje da šifrirani tekst (α, β) iz javnog ključa $(y, (g, p, q))$, kodira vrijednost m dokazujući znanje r, a slučajnost se koristi za stvaranje šifriranog teksta, posebno $g, y, \alpha, \beta/g^m$, uz napomenu da je $a = g^r$ i $p/g^m = y^r$ (Adida, 2012).

- Povjerenik šalje $= g^w \, mod \, p$ i $B = y^w \, mod \, p$ za slučajnih w.
- Verifikator šalje izazov, slučajni *challenge mod q*.
- Povjerenik šalje $odgovor = w + izazov \times r$.
- Verifikator provjerava da:
  - $\mathrm{g}^{\mathrm{odgovor}} = \mathrm{A} * \alpha^{\mathrm{izazov}}$
  - $y^{odgovor} = B \times (\beta/g^m)^{izazov}$

(Adida, 2012)

```
def verify_proof(ciphertext, plaintext, proof, public_key):
    if pow(public_key.g, proof.response, public_key.p) !=
        ((proof.commitment.A * pow(ciphertext.alpha, proof.challenge,
public_key.p)) % public_key.p):
        return False

    beta_over_m = modinverse(pow(public_key.g, plaintext, public_key.p),
public_key.p) * ciphertext.beta
    beta_over_m_mod_p = beta_over_m % public_key.p

    if pow(public_key.y, proof.response, public_key.p) !=
    ((proof.commitment.B * pow(beta_over_m_mod_p, proof.challenge,
public_key.p)) % public_key.p):
        return False

    return True
```

Kod 32. Python kod za provjeru jedinstvenog glasačkog listića (Adida, 2010)

U dokazu da je šifrirani tekst kodiran jednom vrijednošću između *0* i *max*, svi dokazi $max + 1$ su provjereni, a zbroj izazova se uspoređuje s očekivanom vrijednošću izazova. Budući da je ovaj dokaz korišten u interaktivnom/neinteraktivnom obliku, možemo proizvesti očekivani izazov vrijednosti kao:

$$\mathrm{SHA1}\ (\mathrm{A}_0 + "" + \mathrm{B}_0 + "" + \mathrm{A}_1 + "" + \mathrm{B1} + \ldots + "\mathrm{A}_{max}" + "," + \mathrm{B}_{max})\ \text{sa}\ \mathrm{A}_0, \mathrm{B}_0, \mathrm{A}_1, \mathrm{B}_1, \ldots, \mathrm{A}_{max}, \mathrm{B}_{max}$$

u decimalnom obliku (A_i i B_i su komponente obvezne za i'te dokaze.) (Adida, 2012)

Metoda *verify_disjunctive_0..max_proof* na kodu 34. provjerava dokaze u *ELGAMAL_CIPHERTEXT-u*.

```
def verify_disjunctive_0..max_proof(ciphertext, max, disjunctive_proof,
public_key):
    for i in range(max+1):
    if not verify_proof(ciphertext, i, disjunctive_proof[i], public_key):
        return False
    computed_challenge = sum([proof.challenge for proof in disjunctive_proof])
% public_key.q
    list_of_values_to_hash = sum([[p.commitment.A, p.commitment.B] for p in
disjunctive_proof], [])

    str_to_hash = ",".join(list_of_values_to_hash)
    expected_challenge = int_sha(str_to_hash)

    return computed_challenge == expected_challenge
    verify_vote(election, vote):
    computed_hash = base64.b64encode(hash.new(election.toJSON()).digest())[:-1]
    if computed_hash != vote.election_hash:
        return False

    for question_num in range(len(vote.answers)):
    encrypted_answer = vote.answers[question_num]
    question = election.questions[question_num]

    homomorphic_sum = 0

    for choice_num in range(len(encrypted_answer.choices)):
        ciphertext = encrypted_answer.choices[choice_num]
        disjunctive_proof = encrypted_answer.individual_proofs[choice_num]

        if not verify_disjunctive_0..max_proof(ciphertext, 1,
disjunctive_proof, election.public_key):
            return False

        homomorphic_sum = ciphertext + homomorphic_sum

    if not verify_disjunctive_0..max_proof(homomorphic_sum, question.max,
                                           encrypted_answer.overall_proof,
                                           election.public_key):
        return False

    return True
```

Kod 33. Provjera glasačkog listića (Adida, 2010)

5.2 Revizija jednog glasačkog listića

Vrši se provjera na osnovu parametara: *VOTE_WITH_PLAINTEXTS* u JSON obliku, izbora *ELECTION_ID* i otisaka prsta glasača, koja vodi do dokaza da glasački listić pripada određenoj osobi (Adida, 2012). Python programski kod za provjeru jednog glasačkog listića prikazan je u kodu 35.

```
def verify_ballot_audit(vote_with_plaintexts, election, vote_fingerprint):
    if not verify_vote(election, vote_with_plaintexts):
        return False
    for encrypted_answer in vote_with_plaintexts.answers:
        for choice_num in range(len(encrypted_answer.choices)):
        ciphertext = encrypted_answer.choices[choice_num]
        randomness = encrypted_answer.randomness[choice_num]

        if choice_num == encrypted_answer.answer:
            plaintext = public_key.g
        else:
            plaintext = 1

        if pow(public_key.g, randomness, public_key.p) != ciphertext.alpha:
            return False

        expected_beta = (pow(public_key.y, randomness, public_key.p) *
plaintext) % public_key.p
        if expected_beta != ciphertext.beta:
            return False
    vote_without_plaintexts = vote_with_plaintexts.remove_plaintexts()
    computed_fingerprint =
base64.b64encode(hash.new(vote_without_plaintexts.toJSON()).digest())[:-1]

    return computed_fingerprint == vote_fingerprint
```

Kod 34. Python kod za reviziju glasačkog listića (Adida, 2010)

5.3 Provjera kompletnog izbora

Kako bi se provjerili kompletni izbori potrebno je:

- prikazati broj prebrojanih otisaka prstiju,
- treba se osigurati da popis glasača odgovara *hashu* izborne glasačke liste,
- prikazati otisak svakog glasovanja,
- provjeriti da je svaki ubačeni listić pravilno formiran,
- izračunati je li se homomorfno (homomorphically) podudara,
- potvrditi je li svaki povjerenik djelomično dešifriran,
- kombinirati djelomična dešifriranja i potvrditi ta dešifriranja, podudaranje kodiranog homomorfa i da su rezultati konzistentni (Adida, 2012).

Drugim riječima, potpuni rezultati potvrđenih izbora uključuju: otisak prsta izbora, popis glasačkih otisaka prstiju, povjerenika za dešifriranje glasača i dokaza i konačne rezultate.

Povjerenici su odgovorni za dešifriranje izbora. Svaki povjerenik generira ključne riječi i podnosi javni dio Heliosu. Kada je vrijeme za dešifriranje, svaki povjerenik mora pružiti tajni ključ. Helios je automatski prvi povjerenik i automatski će se nositi s generiranjem i dešifriranjem ključeva. Može se dodati dodatne povjerenike, a može se ukloniti i Heliosovog povjerenika (Adida, 2012).

Šifrirani tekst (α, β) i vrijednost privatnog ključa x odgovaraju povjerenikovom javnom ključu y, a djelomično dešifriranje izvodi se formulom:

$$dec_factor = \alpha^x \bmod p.$$

Povjerenik zatim pruža dokaz da su vrijednosti $g, y, alpha, dec_factor$ pravilne, što potvrđuje Chaum-Pedersenov dokaz o diskretnom logaritmiranju jednakosti. Provjera se odvija kao što je prikazano na Python programskom kodu 36.

```
def verify_partial_decryption_proof(ciphertext, decryption_factor, proof,
public_key):

    if pow(public_key.g, proof.response, public_key.p) !=
    ((proof.commitment.A * pow(public_key.y, proof.challenge, public_key.p)) %
public_key.p):
        return False

    if pow(ciphertext.alpha, proof.response, public_key.p) !=
    ((proof.commitment.B * pow(decryption_factor, proof.challenge,
public_key.p)) % public_key.p):
        return False

    str_to_hash = str(proof.commitment.A) + "," + str(proof.commitment.B)
    computed_challenge = int_sha(str_to_hash)

    return computed_challenge == proof.challenge
    Then, the decryption factors must be combined, and we check that:

        dec_factor_1 * dec_factor_2 * ... * dec_factor_k * m = beta (mod p)

def retally_election(election, voters, result, result_proof):
    election_fingerprint = b64_sha(election.toJSON())

    vote_fingerprints = []

    tallies = [[0 for a in question.answers] for question in
election.questions]

    for voter in voters:
        if not verify_vote(election, voter.vote):
            return False

        vote_fingerprints.append(b64_sha(voter.vote.toJSON()))

        for question_num in range(len(election.questions)):
            for choice_num in
range(len(election.questions[question_num].answers)):
                tallies[question_num][choice_num] =
voter.vote.answers[question_num].choices[choice_num] +

tallies[question_num][choice_num]


    for question_num in range(len(election.questions)):
```

```
        for choice_num in range(len(election.questions[question_num].answers)):
            decryption_factor_combination = 1
            for trustee_num in range(len(election.trustees)):
                trustee = election.trustees[trustee_num]
                if not
verify_partial_decryption_proof(tallies[question_num][choice_num],
                    trustee.decryption_factors[question_num][choice_num],
                    trustee.decryption_proof[question_num][choice_num],
                    trustee.public_key):
                return False
                decryption_factor_combination *=
trustee.decryption_factors[question_num][choice_num]
            if (decryption_factor_combination *
election.result[question_num][choice_num]) % election.public_key.p
                != tallies[question_num][choice_num].beta %
election.public_key.p:
                    return False
    return {
        'election_fingerprint': election_fingerprint,
        'vote_fingerprints' : vote_fingerprints,
        'verified_tally' : result
    }
```

Kod 35. Python kod za provjeru kompletnih izbora (Adida, 2010)

6. Instalacija sustava Helios na server (VPS)

6.1 Instalacija Helios sustava na Linux operacijski sustav

Projekt Heliosa nalazi se na https://github.com/benadida/helios-server. Za instalaciju je potrebno pokrenuti terminal, a nakon toga otići u */usr/local/src* direktorij i kreirati direktorij pod nazivom Helios kao na kodu 37.

```
antun@antun-HP:~$ /usr/local/src
antun@antun-HP:~$ mkdir helios
```

Kod 36. Kreiranje direktorija Helios

Helios koristi PostgreSQL bazu podataka koja se instalira naredbama kao na kodu 38.:

```
antun@antun-HP:~$ sudo apt-get update
antun@antun-HP:~$ sudo apt-get install postgresql postgresql-contrib
antun@antun-HP:~$ sudo su - postgres
psql
CREATE DATABASE helios;
CREATE USER helios WITH PASSWORD 'password';
ALTER ROLE helios SET client_encoding TO 'utf8';
```

Kod 37. Instalacija PostgreSQL baze podataka i kreiranje tablice helios

Po zadanim postavkama razina izolacije u PostgreSQL-u je „Read uncommitted“. Poželjno je postaviti „Read Committed“ kao u kodu 39.

```
ALTER ROLE helios SET default_transaction_isolation TO 'read committed';
```

Kod 38. Komanda za postavljanje razine izolacije „Read Committed”

Po zadanim postavkama vremenska zona je „GMT“, potrebno je postaviti „UTC“ kao na kodu 40.

```
ALTER ROLE helios SET timezone TO 'UTC';
GRANT ALL PRIVILEGES ON DATABASE helios TO helios
\q
exit
```

Kod 39. Postavljanje vremenske zone za tablicu Helios

Nakon što su direktorij helios i baza podataka izrađeni, potrebno je povući projekt s repozitorija na računalo. Ako je poznato da se projekt nalazi na *githubu*, potrebno je instalirati *git* na računalo.

```
$ sudo apt-get update
$ sudo apt-get install git
```

Kod 40. Instaliranje programa Git na Linux operacijski sustav

Poslije instalacije *gita* potrebno je povući projekt pomoću naredbe `git clone` kao što je učinjeno na kodu 42.:

```
$ git clone https://github.com/benadida/helios-server.git
```

Kod 41. Preuzimanje Helios projekta

Uz instalaciju Ubuntu operacijskog sustava dolazi Python 2.7 koji je potreban za funkcioniranje Helios projekta. U slučaju da nije instaliran, potrebno ga je instalirati naredbom u terminalu kao na kodu 43.:

```
$ sudo apt-get install python python-pip
```

Kod 42. Instalacija Pythona i paketa python-pip

Ako je instalacija uspješna, potrebno je napraviti virtualno okruženje (eng. virtual environment) samo za Helios. To je potrebno jer "Svaki projekt" sadrži karakteristične pakete, čije verzije ne moraju biti kompatibilne, također postoji mogućnost da različiti projekti rade pomoću različitih verzija programskog jezika, različitih *web frameworka* ili pomoću različitih vrsta baza podataka. Zato je potrebno za svaki projekt izgraditi virtualno okruženje.

Virtualno okruženje kreira se sa *virtualenv* aplikacijom koja se instalira naredbom kao na kodu 44.

```
$ sudo apt-get install python-virtualenv
```

Kod 43. Instalacija virtualnog okruženja za Python

Nakon instalacije paketa *python-virtualenv*, potrebno je postaviti virtualno okruženje naredbom:

```
antun@antun-HP:~$ cd ~/.venvs/
antun@antun-HP:~$ python2 -m virtualenv env-helios
```

Kod 44. Kreiranje virtualnog okruženja za projekt Helios

Provjera se može izvršiti tako da se upiše naredba `ls` koja pokazuje postoje li folderi helios-server i env-Helios kao što je prikazano u kodu 46.

```
antun@antun-HP:~$ ls
env-helios  helios-server
```

Kod 45. Naredba "ls"

U developerskom radu potrebno je aktivirati virtualno okruženje tako da se upišu komande kao na kodu 47.

```
$ source env-helios/bin/activate
(env-helios) antun@antun-HP:~/helios$
```

Kod 46. Aktivacija virtualnog okruženja za Helios projekt

Ako na se na prvom mjestu unutar terminala nalazi (env-helios) može se zaključiti da je virtualno okruženje aktivirano.

Ako je virtualno okruženje instalirano, slijedi instalacija projekta i njegovih paketa u virtualno okruženje. Ponekad neki paketi ne postoje u klasičnom repozitoriju pa ih je potrebno posebno skinuti i instalirati. Paketi koji ne postoje u klasičnom repozitoriju preuzimaju se s drugih repozitorija u kreiranu mapu „paketi“ i instalira ih tako da se raspakiraju te se upiše naredba s koda 48.

```
$ python setup.py install
```

Kod 47. Instalacija python paketa

Ako su svi paketi uredno instalirani potrebno je upisati naredbu za pokretanje servera naredbom:

```
$ ./reset.sh
$ python manage.py runserver
```

Kod 48. Pokretanje Django projekta (sustava Helios) u razvojnom modu

Ako nema grešaka i ako na IP adresi http://127.0.0.1:8000 postoji stranica, onda je pokrenut server u developerskom modu. Nakon toga potrebno je instalirati i pokrenuti RabbitMQ server naredbama kao što je prikazano na kodu 50.

```
$ sudo apt-get install -y build-essential libssl-dev ncurses-dev m4
$ sudo mkdir -p /opt/src/erlang /opt/erlang
$ cd /opt/src/erlang
$  sudo wget http://www.erlang.org/download/otp_src_19.3.tar.gz
$ sudo tar -xvzf otp_src_19.3.tar.gz
$ sudo mv otp_src_19.3 19.3
$ cd 19.3/
$ sudo ./configure --prefix=/opt/erlang/19.3 --enable-hipe --with-ssl
$ sudo make
$ sudo make install
$ sudo ln -s /opt/erlang/19.3/bin/dialyzer /usr/bin
$ cd /tmp && wget $ http://www.rabbitmq.com/releases/rabbitmq-
server/v3.6.9/rabbitmq-server_3.6.9-1_all.deb
$ sudo dpkg -I rabbitmq-server_3.6.9-1_all.deb
$ sudo dpkg -i --ignore-depends=erlang-nox
$ sudo dpkg -i rabbitmq-server_3.6.9-1_all.deb
$ sudo rabbitmqctl status
$ sudo apt-get install tree
$ easy_install celery
$ easy_install django-celery
$ python manage.py celery
```

Kod 49. Instalacija RabbitMQ servera

Ukoliko zadnja upisana naredba pokreće server bez greške, aplikacija odnosno projekt Helios može raditi sa svim svojim funkcionalnostima.

6.2 Konfiguracija Apache servera za Django *web* framework za produkciju

Instalacija samog Django *web frameworka* na produkcijski *web* server je vrlo slična. Razlika je u tome da se sustav pokreće na http serveru kao što su Apache ili NginX u odnosu na lokalno računalo gdje se projekt pokreće u *developerskom* modu s naredbom `python manage.py runserver`. Da bi Helios radio, potrebno je kreirati korisničko ime za PostgreSQL bazu podataka naredbama isto kao i za *developerski* način rada.

Nakon kreiranja korisničkog imena *helios*, prijavljuje se s istim korisničkim imenom na bazu *helios* s naredbom kao s koda 51.

```
psql helios
```

Kod 50. Prijava u bazu helios

Da bi se vidjele tablice, potrebno je upisati naredbu “\l” kao na slici 42.

```
antun@ubuntu:~/diplomski$ source env-helios-server/bin/activate
(env-helios-server) antun@ubuntu:~/diplomski$ psql helios
psql (9.5.7)
Type "help" for help.

helios=> \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 helios    | antun    | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 postgres  | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 template0 | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(4 rows)

helios=> \q
(env-helios-server) antun@ubuntu:~/diplomski$
```

Slika 41. Prikaz tablica i njihovih vlasnika u PostgreSQL bazi

Kao i na lokalnom računalu potrebno je instalirati RabbitMQ server s istim postupkom. Tijekom postupka instalacije RabbitMQ servera potrebni su dodatni pozadinski programi, a slika 42 pokazuje programe koji fale nakon naredbe `$ sudo ./configure --prefix=/opt/erlang/18.3 --enable-hipe --with-ssl`.

```
    ********************************************************************
    **********************  APPLICATIONS DISABLED  **********************
    ********************************************************************

    jinterface     : No Java compiler found
    odbc           : ODBC library - link check failed

    ********************************************************************
    ********************************************************************
    **********************  APPLICATIONS INFORMATION  ******************
    ********************************************************************

    wx             : wxWidgets not found, wx will NOT be usable

    ********************************************************************
    ********************************************************************
    **********************  DOCUMENTATION INFORMATION  *****************
    ********************************************************************

    documentation  :
                     xsltproc is missing.
                     fop is missing.
                     xmllint is missing.
                     The documentation can not be built.

    ********************************************************************
    (env-helios-server) antun@ubuntu:/opt/src/erlang/18.3$
```

Slika 42. Provjera paketa koje još treba instalirati za funkcionalnost RabbitMQ-a

Prva stavka je instalacija *Jave* koja se instalira naredbama kao na kodu 52.

```
$ sudo apt-get update
$ sudo apt-get install default-jre
$ sudo apt-get install default-jdk
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

Kod 51. Instalacija Jave

Druga stavka je *odbc* driver, instalira se naredbom kao na kodu 53.

```
$ sudo apt-get install unixodbc-dev
```

Kod 52. Instalacija ODBC drivera

Treći paket koji nedostaje je wdWidgets, a instalira se naredbama kao na kodu 54.

```
$ sudo apt-get install libgtk-3-dev
$ sudo wget
https://github.com/wxWidgets/wxWidgets/releases/download/v3.1.0/wxWidgets-
3.1.0.tar.bz2
$ sudo tar -xjvf wxWidgets-3.1.0.tar.bz2
$ cd wxWidgets-3.1.0
$ mkdir buildgtk
$ cd buildgtk
$ ../configure --with-gtk
$ make
$ sudo make install
```

Kod 53. Instalacija wxWidgets-a

Ostali paketi instaliraju se naredbom:

```
$ sudo apt-get install fop xsltproc
```

Kod 54. Instalacija fop i xsltproc paketa

Nakon uspješne instalacije potrebno je podesiti *apache* server. Ako se projekt nalazi u */home/* direktoriju pod korisnikom */home/antun* potrebno je korisničko ime „antun“ dodati u grupu www-data kako bi *user* antun imao prava objavljivati stranice na serveru. Primjer dodjele *usera* u grupu *www-dana* kao na kodu 56.:

```
$ sudo adduser -g www-data antun
chown -R antun:www-data ../helios-server/ ../env-helios/
```

Kod 55. Dodjela prava www-data useru antun

Ako su dodana prava korisnika za korištenje *apache2 http servera* potrebno je kreirati i podesiti konfiguracijsku datoteku „001-helios.conf“ u direktoriju */etc/apache2/sites-available/* naredbom:

```
$ sudo nano /etc/apache2/sites-available/001-helios.conf
$ sudo service apache2 restart
```

Kod 56. Kreiranje apache2 konfiguracijske datoteke

Na slici 43 je primjer konfiguracije za „001-helios.conf“ koja omogućuje putanju do direktorija projekta Helios i njegovog virtualnog okruženja.

```
 GNU nano 2.5.3                    File: /etc/apache2/sites-enabled/000-default.conf

<VirtualHost *:80>
        #ServerName helios
        #ServerAlias www.helios
        #DocumentRoot /home/antun/diplomski/helios
        Alias /site_media/static /home/antun/diplomski/helios/site_media/static

        <Directory /home/antun/diplomski/helios/site_media/static>
                Require all granted
        </Directory>

        <Directory /home/antun/diplomski/env-dipl>
                Require all granted
        </Directory>

        <Directory /home/antun/diplomski/helios>
                Require all granted
        </Directory>

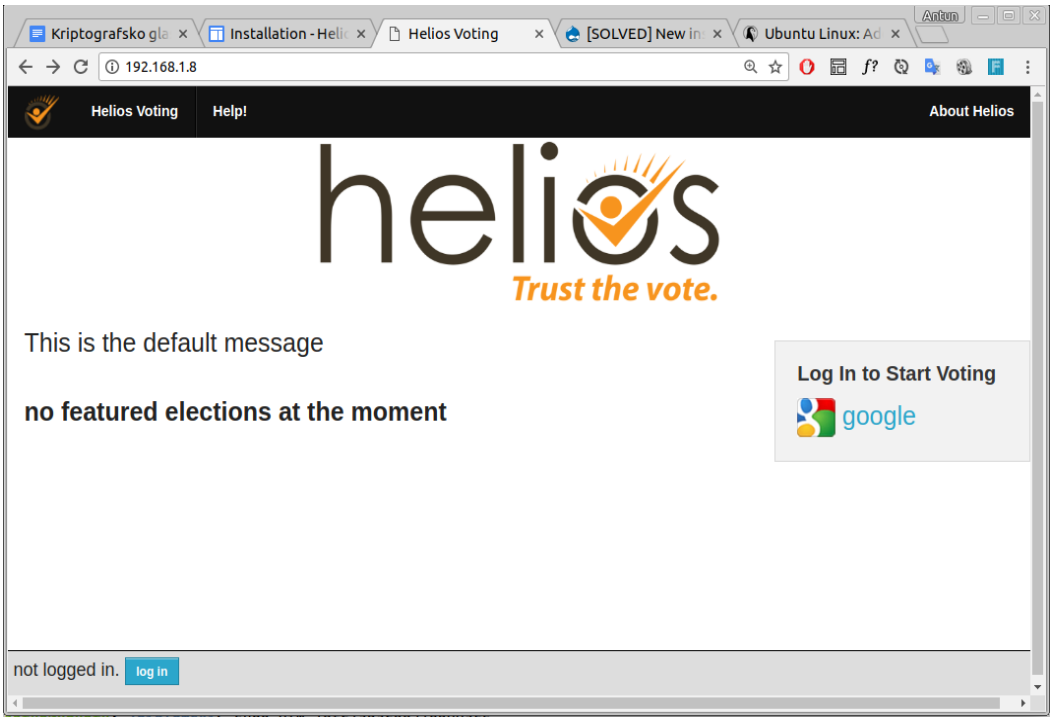
        WSGIDaemonProcess helios python-path=/home/antun/diplomski/helios python-home=/home/antun/diplomski/env-dipl
        WSGIProcessGroup helios
        WSGIScriptAlias / /home/antun/diplomski/helios/wsgi.py
        WSGIPassAuthorization On

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Slika 43. Datoteka 000-default.conf za apache server

Nakon pokrenute naredbe `$ sudo service apache2 restart` na IP adresi računala na kojem se nalazi projekt trebala bi se prikazati *web* stranica projekta kao na slici 44.



Slika 44. Početna stranica projekta Helios

Ako je RabbitMQ server uspješno instaliran po gore navedenim uputama potrebno je u postavkama projekta dati naredbu da se prilikom pokretanja *apache2* servera pokrene i RabbitMQ server kao na slici 45.

```
273  # set up django-celery
274  # BROKER_BACKEND = "kombu.transport.DatabaseTransport"
275  BROKER_URL = "django://antun@antun@127.0.0.1:8000"
276  CELERY_RESULT_DBURI = DATABASES['default']
277  import djcelery
278  djcelery.setup_loader()
279
280
281  # for testing
282  TEST_RUNNER = 'djcelery.contrib.test_runner.CeleryTestSuiteRunner'
283  # this effectively does
284  CELERY_ALWAYS_EAGER = True
```

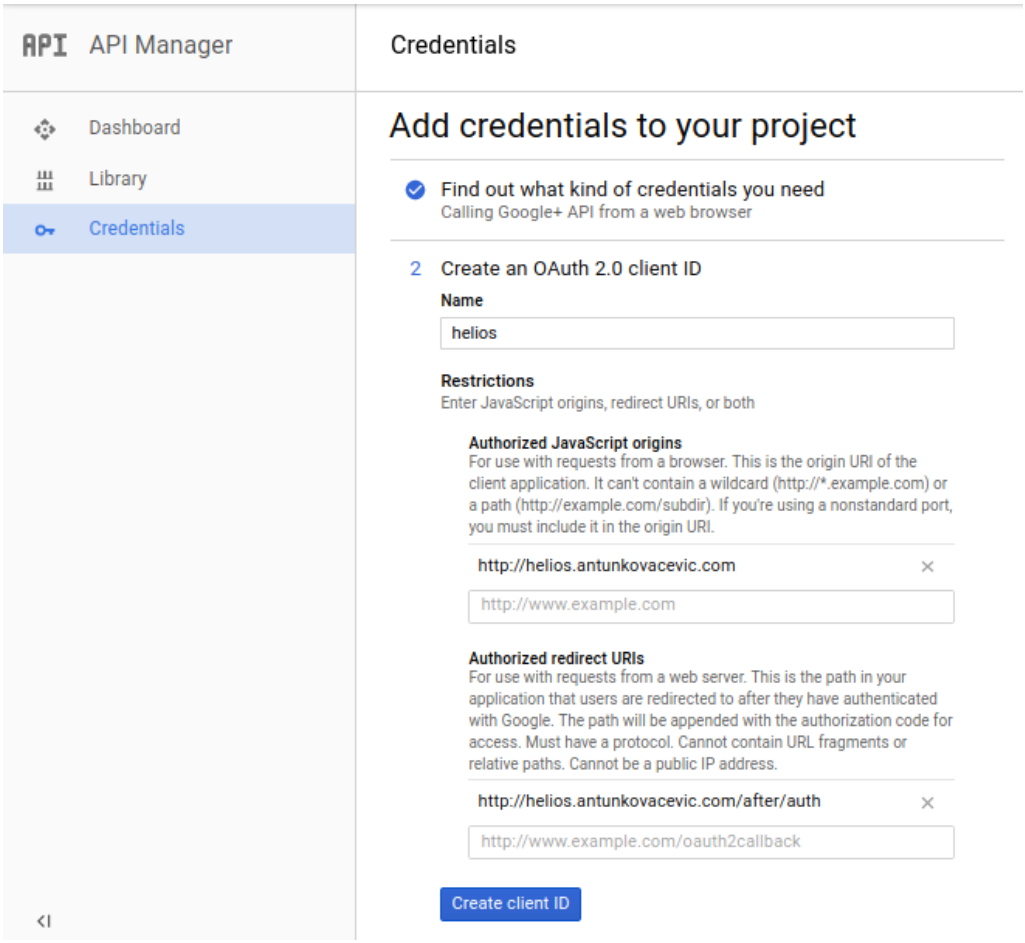
Slika 45. Postavke Django Helios projekta, postavka za *djcelery*, odnosno RabbitMQ (settings.py)

6.3 Helios autentifikacija

Helios omogućuje više načina autentifikacije preko različitih servisa kao što su na primjer: Facebook, Twitter, Google i Yahoo. Svaki taj servis zahtijeva prijavu u njihov sustav te je potrebno zatražiti dopuštenje korištenja njihovih računa za autentifikaciju u drugim sustavima. Nakon što je zahtjev odobren, dobiju se podaci poput identifikacijskog i tajnog ključa.

6.3.1 Podešavanje Google oauth2 za prijavu u sustav Helios

Google autentifikacija zahtijeva dva podatka među kojima su *ID* klijenta i tajni ključ klijenta. *ID* i tajni ključ klijenta dobivaju se tako da se podnese zahtjev na *Google web* stranici koja se nalazi na poveznici https://console.developers.google.com/. Dolaskom na *web* stranicu potrebno se prijaviti u sustav i odabrati *Social APIs -> Google+ API*, pročitati i prihvatiti uvijete korištenja, kreirati projekt pritiskom na gumb ‘Create Project’ i nazvati ga imenom Helios. Nakon kreiranja projekta potrebno je pritisnuti na gumb ‘Enable’, nakon toga pritisnuti na gumb ‘Create credentials’ i popuniti podatke o aplikaciji kao na slici 46.



Slika 46. Podaci za kreiranje pristupnih podataka za autentifikaciju (Google OAuth2)

Nakon pritiska na gumb ‘Create client ID’ dobe se pristupni podatci u obliku JSON datoteke koja se preuzima na lokalno računalo. Bitni podaci su *ClientID* i *ClientSecret* koje je potrebno upisati u datoteku *settings.py* u projektu Helios na mjesta gdje su varijable *GOOGLE_CLIENT_ID* i *GOOGLE_CLIENT_SECRET* kao na slici 48.

```
220  # google
221  GOOGLE_CLIENT_ID = get_from_env('GOOGLE_CLIENT_ID', '531801616930-0pnguhlgenvl7rnf31v1teerrtclejqj.apps.g
222  GOOGLE_CLIENT_SECRET = get_from_env('GOOGLE_CLIENT_SECRET', 'APwKrHmSHqTseqSwNPzrJo4M')
223
```

Slika 47. Prikaz podataka za google autentifikaciju[20]

[20] Podaci nisu uslikani u potpunosti iz sigurnosnih razloga.

7. Potencijalni korisnici sustava

Sustav Helios nije namijenjen za izbore gdje postoji rizik prisile poput predsjedničkih ili vladinih gdje glasač može biti napadnut i prisiljen od napadača koji gleda preko ramena (Adida, 2008).

> *„Brojna poduzeća, udruge, zadruge, softverske zajednice otvorenog koda i drugi ne pate od gotovo jednakog prisilnog rizika za razliku od vladinih izbora. Ipak te skupine još uvijek trebaju tajnost glasovanja i pouzdane rezultate izbora, svojstva koja trenutačno ne mogu postići bez osobnih, fizički vidljivih i dobro orkestriranih izbora, što često nije mogućnost.* Helios *je proizveden upravo za ove grupe s niskim rizikom od prisile.“*[21] *(Adida, 2008)*

Helios je jednostavniji od većine kriptografskih protokola za glasovanje jer se usredotočuje na dokazivanje integriteta. Kao kompromis, Helios čini slabije jamstvo privatnosti.

Kao što je već rečeno, Helios je pogodniji za manje zajednice poput zadruga. Kada zadruga treba donijeti neku odluku, predsjednik zadruge u Heliosu kreira nove izbore s imenom zadruge i nazivom izbora, datumima trajanja izbora i nekim potrebnim informacijama koje se odnose na te izbore. Predsjednik zadruge Dodaje listu glasača u CSV formatu te dodaje izborna pitanja koja se odnose na problematiku donošenja odluke. Kao povjerenike može dodati članove nadzornog odbora. Izbori su spremni nakon što predsjednik zadruge zaključa, odnosno zamrzne izbore. Nakon toga nije moguće mijenjati listu glasača i povjerenike. Tako su izbori spremni i čeka se datum i vrijeme početka izbora.

Helios nije pogodan za izbore u nekim zajednicama ili poduzećima poput dioničkih društava gdje vlasnici imaju različite udjele jer time njihovi glasovi imaju različitu težinu. Dakle, Helios je isključivo namijenjen za princip „jedan član jedan glas“.

[21] Vlastiti prijevod

8. Usporedba Heliosa s Prêt-à-Voter (PaV) glasačkim sustavom

Helios je otvoreni *web* sustav glasovanja koji koristi provjerene kriptografske tehnike *(Adida, 2008)*. Sa sigurnosnog stajališta, najvažnije značajke sustava uključuju: šifriranje na temelju preglednika, homomorfno bilježenje, distribuirano dešifriranje preko više povjerenika, autentifikaciju korisnika putem adrese e-pošte, lozinke za izbor i osiguranje glasovanja putem različitih razina revizije.

Sustav **Prêt-à-Voter** (**PaV**), zasnovan Chaumovom[22] vizualnom kriptografskom shemom, glasački je sustav koji omogućuje glasačima da glasaju s papirnatim oblicima (nasumičnim redoslijedom i odabirom) koji se mogu fizički mijenjati kako bi služili kao šifrirani glasački listići. Ova metoda glasovanja može se revidirati u brojnim fazama od strane glasača i timova revizora. Sustav je fleksibilan jer omogućuje različite sheme šifriranja i različite kriptografske mehanizme koji se koriste po potrebi. U nastavku su navedeni koraci glasovanja u sustavu PaV, kao i slika 48.

„Da bi glasovao s PaV sustavom, glasač slijedi ove korake:

1. *Zapečaćena omotnica koja sadrži papirnate glasove daje se glasaču. Glasač otvara omotnicu, u njoj nalazi upute i kartice koje čine glasački listić.*
2. *Kako bi označio svoje izbore na glasačkim listićima, u desnom kutu označava s križićem (x) mjesto pored imena kandidata ili prijedlog koji želi odabrati.*
3. *Nakon popunjavanja glasačkog listića, glasač odvaja liste kandidata od njegovih odabira ili oznaka.*
4. *Popisi kandidata se sjeckaju, odnosno uništavaju.*
5. *Glasač dostavlja glasačke listiće u skener.*
6. *Glasački listići se stavljaju u glasačku kutiju.*
7. *Glasač uzima tiskani primjerak koji prikazuje slike skeniranih glasačkih listića zajedno s poveznicom za web i kontrolnim kodom koji je potreban za potvrdu glasovanja."*[23] (Acemyan i ostali, 2014)

[22] Kriptograf i izumitelj David Chaum 1994. uveo prvi sustav glasovanja u kojem su glasači elektronički glasali na biračkom mjestu i kriptografski potvrdili da sustav nije izmijenio glasove.

[23] Vlastiti prijevod i korekcija i izgleda



Slika 48. Uobičajeni postupak glasovanja kod sustava Prêt-à-Voter (Acemyan i ostali, 2015)

U nastavku su koraci potvrde glasačkog listića, a prikazani su slikom 49.

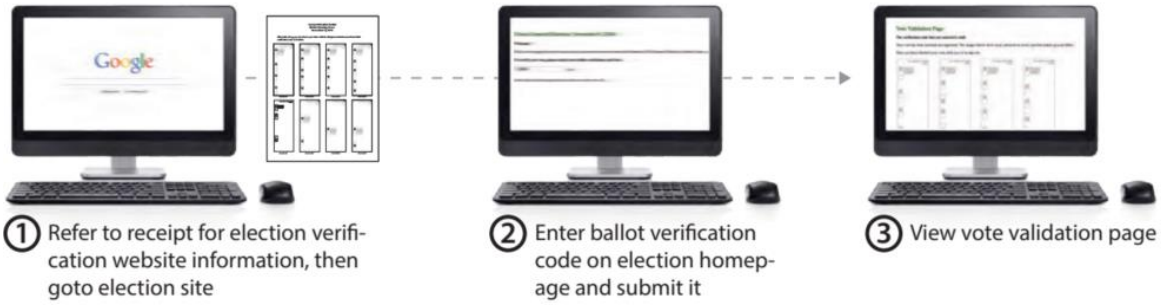
Da bi glasač mogao potvrditi svoj glas koristeći PaV, glasač obično izvodi sljedeću sekvencu na računalu ili mobilnom uređaju:

1. " Potrebno je otići na web stranicu za provjeru izbora gdje je poveznica tiskana na dodijeljenom primitku nakon izbora.

2. Nakon toga mora se unijeti i poslati kod za provjeru glasa na početnoj stranici.

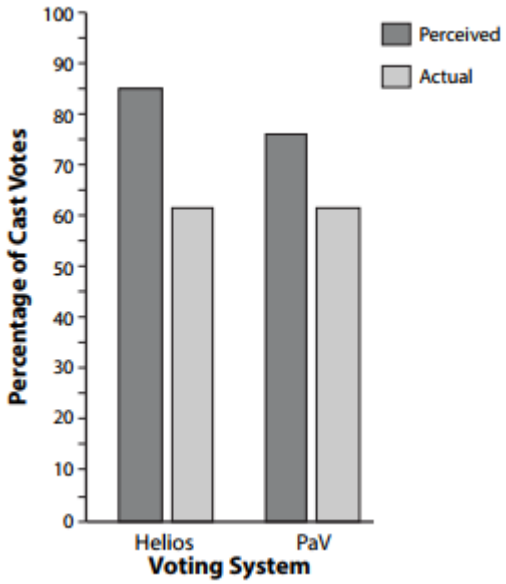
3. *Na stranici je prikaz potvrde glasovanja koja potvrđuje da je unesen valjani kontrolni kod. Ova stranica također prikazuje slike svakog glasačkog listića i time prikazuje svaki izbor na svakoj kartici (bez popisa kandidata) koji čine njihov glasački listić."*[24] (Acemyan i ostali, 2014)

[24] Vlastiti prijevod



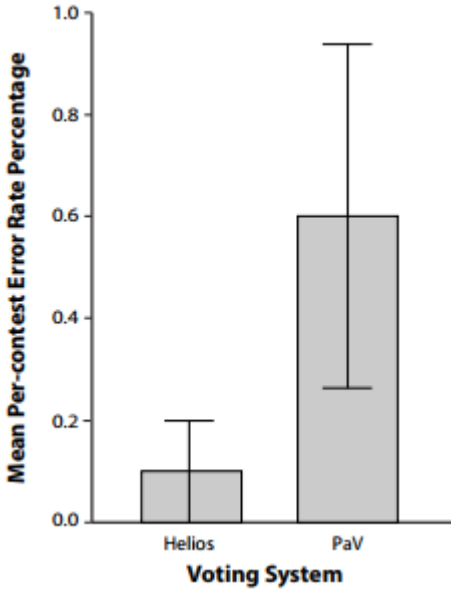
Slika 49. Uobičajeni postupak provjere glasačkog listića kod sustava Prêt-à-Voter (Acemyan i ostali, 2014)

Jedna od razlika između Heliosa i PaV-a je ta da je Helios sustav potpuno na *webu*, dok se za PaV uz *web* još koristi papir, printer, skener i uređaj za uništavanje papira. Za svaki od ovih sustava postoji definiran postupak kako se glasuje. Kod jednog i kod drugog postupak je pomalo kompliciran. Grafikon 1 pokazuje da sustav Helios ima visoku stopu glasača koji su bili uvjereni da su glasovali, a nisu. To se događa jer je postupak bilježenja glasa tek nakon prijave u sustav, a to je predzadnji korak. Dakle, nakon popunjavanja glasačkog listića, kriptiranja i pritiskanja tipke "ubaci listić", dolazi forma za prijavu, kad se osoba prijavi, ima pravo ubaciti listić. To može biti zbunjujuće jer glasači stisnu tipku "ubaci listić" i budu uvjereni da su glasovali jer misle da ih je sustav odjavio, a trebali bi se prijaviti i dovršiti glasovanje (Acemyan i ostali, 2014).



Grafikon 1. Postotak glasačkih listića kao funkcija sustava glasovanja, sa stupićima koji predstavljaju percipirane i stvarne glasove (Acemyan i ostali, 2014)

„Stope pogrešaka kao funkcije sustava mogu se vidjeti na grafikonu 2. E2E sustavi imaju bolju izvedbu od drugih testiranih sustava glasovanja koji imaju pogreške rasponu od manje od 0,5% do oko 3,5% (Byrne et al., 2007).“[25] (Acemyan i ostali , 2014)



Grafikon 2. Srednja stopa postotka pogrešaka prema natjecanju kao funkcija tipa glasačkog sustava, stupci prikazuju standardnu pogrešku srednje vrijednosti (Acemyan i ostali, 2014)

Tablica 7 *„prikazuje učestalost glasačkih listića koje sadrže pogreške u sustavu glasovanja. Sveukupno, 5 od 111 (5%) glasačkih listića sadržavalo je najmanje jednu pogrešku. Na temelju obje stope pogrešaka i stope pogrešaka po glasovanju po glasaču, glasači koji koriste e2e sustave čine manje pogrešaka u odabiru kandidata i prijedloga na svojim glasačkim listićima.“*[26] (Acemyan i ostali, 2014)

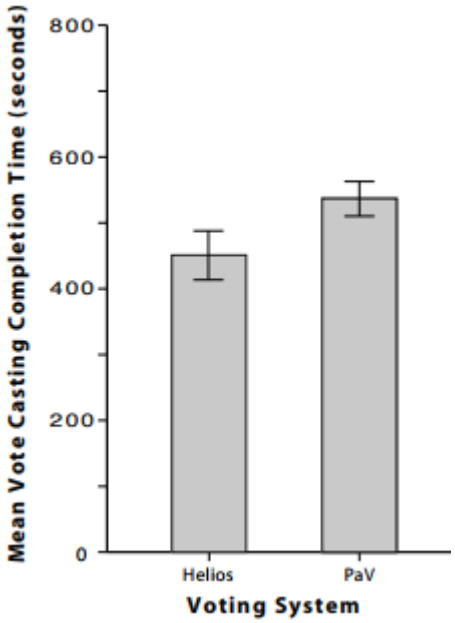
[25] Vlastiti prijevod

[26] Vlastiti prijevod

Tablica 7. Broj i postotak glasačkih listića s jednom ili više pogrešaka u funkciji glasačkog sustava (Acemyan i ostali, 2014)

	Helios	PaV
Broj glasačkih listića s pogreškama	1 (3%)	4 (11%)

Prosječno vrijeme dovršetka izbora kao funkcija sustava glasovanja prikazano je na grafikonu 3. Kao što se može vidjeti, postoje razlike u vremenima glasovanja. Sudionici su najmanje vremena za glasovanje potrošili s Heliosom i više vremena za glasovanje sa *Prêt-à-Voterom*. Prosječno vrijeme završetka glasovanja pomoću glasačkih listića i papirnate kartice iznosi oko 231 sekunde (Acemyan i ostali, 2014). Dakle, *E2E* sustavi traju duže.

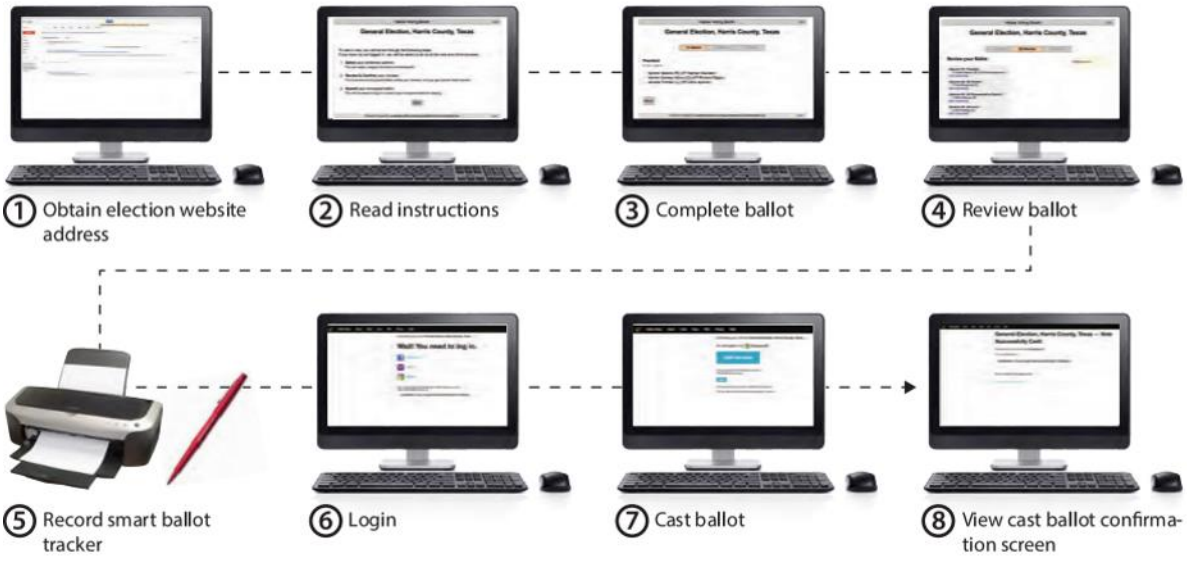


Grafikon 3. Srednja vrijednost glasovanja dovršena je kao funkcija sustava glasovanja, s pogreškama koje prikazuju standardnu pogrešku srednje vrijednosti (Acemyan i ostali, 2014)

Uobičajeni postupak glasovanja kod sustava Helios ima osam koraka što je prikazano na slici 50. Za razliku od sustava Helios, Prêt-à-Voter ima sedam koraka. Prije svakog glasovanja na bilo kojem drugom sustavu, poželjno je proučiti i savladati postupak glasovanja kako bi se izbjegli predvidivi problemi.

Osam koraka za glasovanje u sustavu Helios (slika 50):

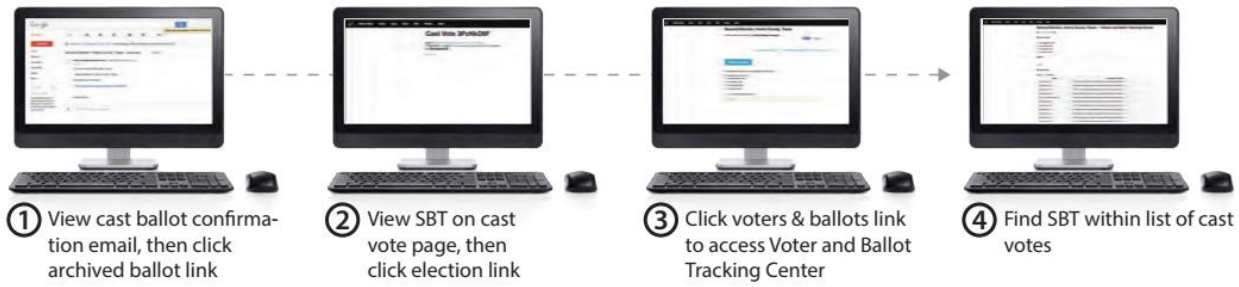
1. Odlazak na *web* adresu izbora koja se nalazi u e-pošti
2. Čitanje uputa za glasovanje
3. Pristupanje listiću
4. Pregled glasačkog listića
5. Bilježenje otiska prsta za praćenje
6. Prijava u sustav
7. Ubacivanje listića
8. Poruka na ekranu o uspješnosti ili neuspješnosti glasovanja



Slika 50. Uobičajeni postupak glasovanja kod sustava Helios (Acemyan i ostali, 2015)

Četiri uobičajena koraka za provjeru glasačkog listića u sustavu Helios (slika 51):

1. Nakon uspješno provedenog glasovanja, glasač dobiva potvrdnu poruku e-poštom. Ako želi provjeriti glasački listić, potrebno je kliknuti na poveznicu koja vodi na *web* stranicu s tim arhiviranim listićem.
2. Prikaže se stranica s arhiviranim listićem te je potrebno pritisnuti na poveznicu izbora u koji je ubačen navedeni listić.
3. Nakon prikaza stranice potrebno je pritisnuti na poveznicu “voters & ballots” za pristup centru za praćenje glasača i glasačkih listića.
4. Pretraga glasačkog listića



Slika 51. Uobičajena provjera glasačkog listića u sustavu Helios (Acemyan i ostali, 2015)

9. Zaključak

Rad je koncipiran tako da je na početku prikazana kratka povijest kriptografije. Opisani su simetrični i asimetrični kriptosustavi poput AES-a i ElGamala te *hash* algoritam SHA2 koji su korišteni u Heliosu . Sa sigurnosnog stajališta, najvažnije značajke Heliosa uključuju šifriranje na temelju preglednika, homomorfno bilježenje, dešifriranje preko više povjerenika, autentifikaciju korisnika putem adrese e-pošte, *Googlea* i ostalih servisa. Opisane su tehnologije u kojem je Helios rađen, na koji način funkcionira te kako se koristi. Prikazani su i pojašnjeni dijagrami korištenja Heliosa te relacijski model baze podataka. Helios je povjerljiv sustav za glasovanje, ponajprije zbog toga što osoba koja je glasovala može provjeriti je li njezin glas pribrojen konačnom skupu glasova, odnosno može se napraviti revizija gdje se svi glasovi prikazuju kao šifrirani. Helios je također povjerljiv jer brine o osiguranju glasača, pri čemu svaki glasač dobiva osobnu sigurnost da je njihov glas pravilno pohranjen i provjerljivost gdje bilo koji promatrač može potvrditi da su svi prikupljeni glasovi ispravno zabilježeni.

Detaljno je opisan postupak instalacije Heliosa na Linux *web* poslužitelj, konfiguracija google autentifikacije i postavljanje sigurnosnih postavki. U predzadnjem poglavlju argumentirano je za koga je Helios namijenjen, odnosno zašto je bolji za manje skupine. Na kraju rada Helios je uspoređen s drugim glasačkim kriptosustavom pod imenom Prêt-à-Voter tako da su opisani i slikovno prikazani koraci glasovanja, vrijeme trajanja i stupanj pogreške glasovanja. Istraživanje je pokazalo da Helios ima veći stupanj pogreške od sustava Prêt-à-Voter jer se pojedini glasači nakon glasovanja ne prijave u sustav što je posljednji korak glasovanja u Heliosu. Prema tome, potrebno je educirati glasače prije glasovanja. Ima mjesta za poboljšanje Heliosa u smislu dizajna same *web* aplikacije i smanjenju broja koraka kako bi bila pristupačnija glasaču te tako bi se smanjio i stupanj pogreške glasovanja.

Jedna od pozitivnih strana Heliosa je da je otvorenog koda te ga svatko može koristiti. Na stranici heliosvoting.org svatko može postaviti izbore i pozvati glasače na glasovanje.

Literatura

1. About JavaScript [Online] Dostupno na: https://developer.mozilla.org/en-US/docs/*Web*/JavaScript/About_JavaScript [Pristupljeno: 8. rujna 2017.]
2. Adida B. (2008) *HELIOS: Web-based Open-Audit Voting* [Online] 17th USENIX Security Symposium. str.335.-348. Dostupno na: https://www.usenix.org/legacy/events/sec08/tech/full_papers/adida/adida.pdf [Pristupljeno: 12. rujna 2017.]
3. Adida B. (2010) *Helios v3 Verification Specs* [Online] Dostupno na: http://documentation.heliosvoting.org/verification-specs/helios-v3-verification-specs [Pristupljeno: 18. rujna 2017.]
4. Adida B. (2011) *Installation* [Online] Dostupno na: http://documentation.heliosvoting.org/install [Pristupljeno: 21. rujna 2017.]
5. Adida B. (2012) *Helios v4* [Online] Dostupno na: http://documentation.heliosvoting.org/verification-specs/helios-v4 [Pristupljeno: 11. listopada 2017.]
6. Al-Tamimi A. (2006) *Performance Analysis of Data Encryption Algorithms* [Online] Dostupno na: https://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf [Pristupljeno: 11. svibnja 2017.]
7. Anderson J. (2008) Security *Engineering: A Guide to Building Dependable Distributed System* Second Edition str.129-183. Inc., Indianapolis, Indiana
8. Anicas M. (2014) An Introduction to OAuth 2 [Online] Dostupno na: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 [Pristupljeno: 16. rujna 2017.]
9. Bendoraitis A. (2014) *Web Development with Django Cookbook*, UK: Packt Publishing Ltd.
10. Blažević M. (2005) *Asimetrična kriptografija* [Online] Dostupno na: http://*web*.zpr.fer.hr/ergonomija/2005/blazevic/asimkripto.htm [Pristupljeno: 16. rujna 2017.]
11. Bray T. (2014) RFC 7159:*The JavaScript Object Notation (JSON) Data Interchange Format* (2014) [Online] Dostupno na: https://tools.ietf.org/html/rfc7159 [Pristupljeno: 8. rujna 2017.]

12. CARNet CERT (2007) *Sigurnost elektroničkog glasovanja* [Online] CCERT-PUBDOC-2007-04-188 Dostupno na: http://www.cert.hr/sites/default/files/CCERT-PUBDOC-2007-04-188.pdf [Pristupljeno: 25. svibnja 2017.]
13. Celery Doc [Online] Dostupno na: http://docs.celeryproject.org/en/latest/getting-started/introduction.html#what-s-a-task-queue [Pristupljeno: 11. lipnja 2017.]
14. Costa E. (2013) *How to install RabbitMQ with the latest Erlang release on Debian* [Online] Dostupno na: https://blog.eriksen.com.br/en/how-install-rabbitmq-latest-erlang-release-debian [Pristupljeno: 16. rujna 2017.]
15. Daniel J. Barrett, Richard E. Silverman, Robert G. Byrnes (2005) *SSH, the Secure Shell The Definitive Guide*, Second edition, USA: O'Reilly Media, Inc
16. Django (2017) [Online] Dostupno na: https://devopedia.org/django [Pristupljeno: 16. rujna 2017.]
17. Django Instalacija [Online] Dostupno na: https://docs.djangoproject.com/en/2.0/faq/install/ [Pristupljeno: 25. svibnja 2017.]
18. Doctorow C. (2017) A brief history of Alice & Bob, cryptography's first couple, 2017. [Online] Dostupno na: https://boingboing.net/2017/07/19/heteronormative-exemplars.html [Pristupljeno: 8. rujna 2017.]
19. Dujella A. (2007) *ElGamalov kriptosustav* [Online] Dostupno na: https://*web*.math.pmf.unizg.hr/~duje/ecc/elgamal.html [Pristupljeno: 25. svibnja 2017.]
20. Ellingwood J. (2015) *How To Serve Django Applications with Apache and mod_wsgi on Ubuntu 14.04* [Online] Dostupno na: https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-14-04 [Pristupljeno: 29. kolovoza 2017.]
21. Ellingwood J. (2016) *How To Install and Use PostgreSQL on Ubuntu 16.04* [Online] Dostupno na: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-16-04 [Pristupljeno: 16. rujna 2017.]

22. End-to-end auditable voting systems (2017) [Online] Dostupno na: https://en.wikipedia.org/wiki/End-to-end_auditable_voting_systems [Pristupljeno: 19. travnja 2017.]
23. Extended Euclidean algorithm (2017) [Online] Dostupno na: https://en.wikibooks.org/wiki/Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm [Pristupljeno: 29. kolovoza 2017.]
24. Ferguson N., Schneier B., Kohno T. (2010) *Cryptography Engineering: Design Principles and Practical Applications* str.195 Inc., Indianapolis, Indiana
25. First steps with Django (2012) [Online] Dostupno na: http://docs.celeryproject.org/en/2.5-archived/django/first-steps-with-django.html [Pristupljeno: 1. kolovoza 2017.]
26. *FizzBuzz Python Solution* (2017) [Online] Dostupno na: https://gist.github.com/jaysonrowe/1592775 [Pristupljeno: 16. rujna 2017.]
27. Google OAuth2 [Online] Dostupno na: https://console.cloud.google.com/apis/credentials/oauthclient/ [Pristupljeno: 11. veljače 2017.]
28. Hardt D. (2012) *RFC 6749 The OAuth 2.0 Authorization Framework* [Online] Dostupno na: https://tools.ietf.org/html/rfc6749 [Pristupljeno: 7. kolovoza 2017.]
29. IBM *Veze s izvorom podataka, Razine izolacije* [Online] Dostupno na: https://www.ibm.com/support/knowledgecenter/hr/SSEP7J_10.2.2/com.ibm.swg.ba.cognos.ug_fm.10.2.2.doc/c_workingwithdatasourceconnections.html [Pristupljeno: 19. travnja 2017.]
30. Jakić M. (2015) *ElGamal algoritam.* Specijalistički rad, Prirodno-matematički fakultet Podgorica [Online] Dostupno na: http://www.vladimirbozovic.net/univerzitet/bozovic/wp-content/uploads/2011/06/Marijana.pdf [Pristupljeno: 16. rujna 2017.]
31. Khai S. (2003) *ElGamal* [Online] Dostupno na: http://users.monash.edu/~skcho5/Thesis/node47.html [Pristupljeno: 7. kolovoza 2017.]
32. Kuchling A. *Python PEP 206 -- Python Advanced Library* [Online] Dostupno na: https://www.python.org/dev/peps/pep-0206/ [Pristupljeno: 19. travnja 2017.]

33. Kumar Shil A. (2015) *Make a Blog using Django* [Online] Dostupno na: http://ruddra.com/2015/09/18/make-a-blog-using-django-part-1-2/ [Pristupljeno: 16. rujna 2017.]
34. Lexie (2017) *Zero-knowledge proofs explained: Part 1* [Online] Dostupno na: https://www.expressvpn.com/blog/zero-knowledge-proofs-explained/ [Pristupljeno: 18. veljače 2018.]
35. Malidžan V., Đaković B. (2012) *Homomorfna enkripcija* [Online] 11 (8) str.735-741 Dostupno na: http://infoteh.etf.unssa.rs.ba/zbornik/2012/radovi/RSS-5/RSS-5-1.pdf [Pristupljeno: 8. rujna 2017.]
36. Mesić M. (2002) *El Gamal kriptografija* [Online] Dostupno na: http://sigurnost.zemris.fer.hr/algoritmi/asimetricni/2002_mesic/index.htm [Pristupljeno: 7. kolovoza 2017.]
37. Morrell D. (2010) *Secret Ballots, Verifiable Votes* [Online] Dostupno na: http://harvardmagazine.com/2010/05/secret-ballots-verifiable-votes [Pristupljeno: 16. rujna 2017.]
38. Morris J. (2018) *How To Create a Django App and Connect it to a Database* [Online] Dostupno na: https://www.digitalocean.com/community/tutorials/how-to-create-a-django-app-and-connect-it-to-a-database [Pristupljeno: 11. veljače 2017.]
39. Olembo M. (2010) *Usable Verifiable Remote Electronic Voting case study HELIOS* [Online] Dostupno na: http://secvote2010.uni.lu/slides/mvolkamer-usability.pdf [Pristupljeno: 16. rujna 2017.]
40. OpenCores: AES, 2015. [Online] Dostupno na: https://opencores.org/project,tiny_aes [Pristupljeno: 9. lipnja 2017.]
41. PostgreSQL *About PostgreSQL* [Online] Dostupno na: https://www.postgresql.org/about [Pristupljeno: 8. rujna 2017.]
42. PostgreSQL *Transaction Level* [Online] Dostupno na: https://www.postgresql.org/docs/9.5/static/transaction-iso.html [Pristupljeno: 16. rujna 2017.]
43. RabbitMQ (2018) [Online] Dostupno na: http://www.rabbitmq.com/install-debian.html [Pristupljeno: 11. veljače 2017.]

44. Rouse M. (2017) *Advanced Encryption Standard (AES)* [Online] Dostupno na: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard [Pristupljeno: 16. rujna 2017.]
45. Schneier B. (1996) *Applied Cryptography* 2nd Ed, John Wiley & Sons. str.476-478 [Online] Dostupno na: http://sigurnost.zemris.fer.hr/algoritmi/asimetricni/2002_mesic/ElGamal_Algorithm_Schneier_2nd_Ed_p476.htm [Pristupljeno: 9. lipnja 2017.]
46. Shostack A. (2014)*Threat Modeling Designing for Security* str. 336 Inc., Indianapolis, Indiana
47. Stallings W. (2017) *Cryptography and network security principles and practice*, 7th Edition, Malaysia: SPi Global
48. Stuntz C. (2010) *What is Homomorphic Encryption, and Why Should I Care? [Online] Dostupno na: https://community.embarcadero.com/blogs/entry/what-is-homomorphic-encryption-and-why-should-i-care-38566 [Pristupljeno: 15. siječnja 2018.]*
49. Using RabbitMQ (2012) [Online] Dostupno na: http://docs.celeryproject.org/en/2.5-archived/getting-started/brokers/rabbitmq.html#broker-rabbitmq [Pristupljeno: 8. rujna 2017.]
50. W3C: JavaScript JSON [Online] Dostupno na: https://www.w3schools.com/js/js_json.asp [Pristupljeno: 9. lipnja 2017.]
51. Wang J., Kissel Z. (2015) *Introduction to network security theory and practice*, Second edition, John Wiley & Sons Singapore Pte Ltd
52. WxWidgets (2017) [Online] Dostupno na: http://codelite.org/LiteEditor/WxWidgets31Binaries#toc2 [Pristupljeno: 24. lipnja 2017.]
53. Ziegler Acemyan C. i ostali (2014) *Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for HELIOS, Prêt-à-Voter, and Scantegrity II* [Online] 2 (3/6). str.26-56. Dostupno na: https://www.usenix.org/system/files/conference/evtwote14/jets_0203-acemyan.pdf [Pristupljeno: 16. rujna 2017.]

54. Ziegler Acemyan C. i ostali (2015) *From Error to Error: Why Voters Could not Cast a Ballot and Verify Their Vote With HELIOS, Prêt-à-Voter, and Scantegrity II* [Online] 3 (2/8). str.1-25. Dostupno na: https://www.usenix.org/system/files/jets/issues/0302/overview/jets_0302-acemyan.pdf [Pristupljeno: 24. lipnja 2017.]

Slike

Slika 1. Postupak simetrične kriptografije (Al-Tamimi, 2006) 5

Slika 2. Prikaz ECB načina rada (Stallings, 2017., str. 215.) 6

Slika 3. Primjer ECB načina rada na slici, Wikipedija 7

Slika 4. Prikaz CFB moda (Stallings, 2017., str. 219.) 8

Slika 5. Prikaz OFB moda (Stallings, 2017., str. 221.) 9

Slika 6. Prikaz CTR načina rada (Stallings W., 2017., str. 223.) 10

Slika 7. Općenito funkcioniranje simetričnog kriptosustava u komunikacij (Stallings, 2017., str. 130.) 12

Slika 8. Operacija "Zamjena bajtova" (Stallings, 2017., str. 180.) 15

Slika 9. Rezultat zamjene bajtova (Stallings, 2017. str. 182.) 16

Slika 10. Zamjena redova (Stallings, 2017., str. 186.) 16

Slika 11. Rezultat promjenjenih redova (Stallings, 2017., str. 185.) 16

Slika 12. Transformacija stupaca (Stallings, 2017. str. 186.) 17

Slika 13. Način računanja jednog stupca matrice (Stallings, 2017., str. 187.) 17

Slika 14. Rezultat promjenjenih stupaca matrice (Stallings, 2017., str. 187.) 17

Slika 15. Primjer računanja jednog supca matrice (Stallings, 2017., str. 187.) 17

Slika 16. Način dobivanja rezultata korištenjem operacije "isključivo ILI" (Stallings, 2017., str. 187.) 18

Slika 17. Rezultat operacije "AddRoundKey" (Stallings, 2017., str. 189.) 18

Slika 18. AES struktura podatala (Stallings, 2017., str. 176.) 19

Slika 19. Prikaz jedne runde AES šifriranja (Stallings, 2017., str. 190.) 19

Slika 20. Postupak simetrične kriptografije (Al-Tamimi, 2006) 22

Slika 21. Šifriranje s javnim ključem pomoću RSA algoritma (Stallings, 2017., str. 287.) 24

Slika 22. Tok šifriranja i dešifriranja poruke u ElGamalovom kriptosustavu (Khai, 2003) 27
Slika 23. Virtualno okružnje za projekte napisane u Python programskom jeziku 33
Slika 24. Django arhitektura (Django, 2017) 38
Slika 25. Stranica dobrodošlice u Djangu (Kumar Shil, 2015) 41
Slika 26. Lista naslova i tekstova bloga 43
Slika 27. Struktura Helios projekta 44
Slika 28. Struktura helios aplikacije 45
Slika 29. Struktura helios_auth aplikacije 45
Slika 30. Struktura heliosbooth aplikacije 46
Slika 31. Komunikacija protokola OAuth 2.0 (Anicas, 2014) 51
Slika 32. Dijagram obrazaca uporabe (use case) sustava za glasovanje Helios 52
Slika 33. Dijagram slijeda obrazaca uporabe 53
Slika 34. Dijagram stanja 54
Slika 35. Kontekstualni dijagram kriptografskog informacijskog sustava Helios 55
Slika 36. Relacijski model baze podataka sustava Helios 56
Slika 37. Kreiranje izbora u Heliosu 57
Slika 38. Unos popisa birača u sustavu Helios 58
Slika 39. Unos izbornog pitanja 59
Slika 40. Dijagram korištenja sustava Helios (Olembo, 2010., str. 6.) 61
Slika 41. Prikaz tablica i njihovih vlasnika u Postgresql bazi 84
Slika 42. Provjera, paketi koje još treba instalirati za funkcionalnost RabbitMQ-a 84
Slika 43. Datoteka 000-default.conf za apache server 86
Slika 44. Početna stranica projekta Helios 87

Slika 45. Postavke Django Helios projekta, postavka za djcelery, odnosno RabbitMQ (settings.py) 87

Slika 46. Podaci za kreiranje pristupnih podataka za autentifikaciju (Google OAuth2) . 89

Slika 47. Prikaz podataka za google autentifikaciju 89

Slika 48. Uobičajeni postupak glasovanja kod sustava Prêt-à-Voter (Acemyan i ostali, 2015) 92

Slika 49. Uobičajeni postupak provjere glasačkog listića kod sustava Prêt-à-Voter (Acemyan i ostali, 2014) 93

Slika 50. Uobičajeni postupak glasovanja kod sustava Helios (Acemyan i ostali, 2015)96

Slika 51. Uobičajena provjera glasačkog listića u sustavu Helios (Acemyan i ostali, 2015) 97

Tablice

Tablica 1. S-kutija (eng. S-Box) 15

Tablica 2. Usporedba rezultata pomoću Crypto++ (Al-Tamimi, 2006) 21

Tablica 3. Uspoređivanje SHA parametara (Stallings, 2017., str. 356.) 30

Tablica 4. Ograičenja PostgresSql baze (PostgreSQL) 47

Tablica 5. Razine izolacije i njihove sklonosti greškama (PostgreSql Transaction Level) 48

Tablica 6. URI-jevi u sustavu Helios (Adida, 2012) 62

Tablica 7. Broj i postotak glasačkih listića s jednom ili više pogrešaka u funkciji glasačkog sustava (Acemyan i ostali, 2014) 95

Kod

Kod 1. Python implementacija za Euklidov algoritam (Extended Euclidean algorithm, 2017) 28

Kod 2. Primjer pretvaranja riječi u hash SHA224 29

Kod 3. Primjer Python koda na primjeru FizzBuzz igre (FizzBuzz Python Solution, 2017) 32
Kod 4. JSON format 35
Kod 5. Pretvaranje JSON-a u object (W3C: JavaScript JSON) 35
Kod 6. Postavke baze podataka za PostgreSQL u Django postavkama 40
Kod 7. Aktiviranje aplikacije u Djangu 40
Kod 8. Primjer modela baze podataka u Djangu 40
Kod 9. URL putanje u Djangu 42
Kod 10. Registriranje aplikacije u Django administracijsko sučelje 42
Kod 11. Ispis liste unutar HTML-a 42
Kod 12. ElGamal javni ključ (Adida, 2012) 63
Kod 13. ElGamal kriptirani tekst (Adida, 2012) 63
Kod 14. Primjer hash vrijednosti (Adida, 2012) 64
Kod 15. Podaci glasača u sustavu Helios s indentifikacijom pomoću e-pošte (Adida, 2012) 64
Kod 16. Podaci glasača u sustavu Helios s indetifikacijom pomoću OpenID (Adida, 2012) 64
Kod 17. Podaci za zamjensko ime u Heliosu (Adida, 2012) 65
Kod 18. Ubačeni listić u sustavu Helios (Adida, 2012) 65
Kod 19. Kraća verzija ubačenog listića (Adida, 2012). 65
Kod 20. Podaci izbora u Heliosu 66
Kod 21. QUESTION_LIST (Adida, 2012) 66
Kod 22. Izborno pitanje, QUESTION (Adida, 2012) 67
Kod 23. Lista glasača, VOTER_LIST (Adida, 2012) 68
Kod 24. Listić, VOTE (Adida, 2012) 69

Kod 25. Kriptirani odgovor, ENCRYPTED_ANSWER (Adida, 2012) 69
Kod 26. ZK_PROOF_0..max (Adida, 2012) 69
Kod 27. ZK_PROOF, otvoreni tekst 70
Kod 28. Dokaz u JSON formatu 71
Kod 29. VOTE_WITH_PLAINTEXTS 71
Kod 30. Šifrirani odgovor s čistim tekstom 72
Kod 31. Rezultat, RESULT 72
Kod 33. Python kod za provjeru jedinstvenog glasačkog listića (Adida, 2010) 73
Kod 34. Provjera glasačkog listića (Adida, 2010) 75
Kod 35. Python kod za reviziju glasačkog listića (Adida, 2010) 76
Kod 36. Python kod za provijeru kompletnih izbora (Adida, 2010) 79
Kod 37. Kreiranje direktorija Helios 80
Kod 38. Instalacija PostgreSQL baze podataka i kreiranje tablice helios 80
Kod 39. Komanda za postavljanje razine izolacije „Read Committed” 80
Kod 40. Postavljanje vremenske zone za tablicu Helios 80
Kod 41. Instaliranje Gita na Linux operacijski sustav 81
Kod 42. Preuzimanje Helios projekta 81
Kod 43. Instalacija Pythona i paketa python-pip 81
Kod 44. Instalacija virtualnog okruženja za Python 81
Kod 45. Kreiranje virtualnog okruženja za projekt Helios 81
Kod 46. Naredba "ls" 81
Kod 47. Aktivacija virtualnog okruženja za Helios projekt 82
Kod 48. Instalacija python paketa 82
Kod 49. Pokretanje Django projekta (sustava Helios) u razvojnom modu 82

Kod 50. Instalacija RabbitMQ servera 83
Kod 51. Prijava u bazu helios 83
Kod 52. Instalacija Jave 85
Kod 53. Instalacija ODBC drivera 85
Kod 54. Instalacija wxWidgets-a 85
Kod 55. Instalacija fop i xsltproc paketa 85
Kod 56. Dodjela prava www-data useru antun 85
Kod 57. Kreiranje apache2 konfiguracijske datoteke 86

Sažetak

Cilj ovog diplomskog rada je analiza E2E kriptografskog glasačkog sustava Helios. To obuhvaća opis tehnologija na temelju kojih je Helios rađen, opis kriptografskih algoritama koji su korišteni poput ElGamala, a također je i detaljno opisana logika i princip glasovanja. Pozadinski sustav (eng. *backend*) baziran je na Django *web frameworku* što znači da je pisan u Python programskom jeziku, dok je sučelje (eng. *frontend*) pisano u JavaScript programskom jeziku. Prikazani su razni dijagrami, relacijski model baze podataka, struktura podataka, primjeri podataka, pseudo i originalni Python kodovi. Detaljno je opisana instalacija Heliosa na Linux apache2 *web* poslužitelj, a zatim je analizirana razlika između Heliosa i drugog kriptografskog glasačkog sustava Prêt-à-Voter.

Ključne riječi: Helios, Glasovanje, Kriptografija, ElGamal, E2E, apache2, Python, JavaScript, Linux

Summary

This thesis analyses is the E2E cryptographic voting system Helios. The analyses includes a describing the technologies which Helios was made, the cryptographic algorithms used, such as ElGamal, as well as a detailed review of the voting principle. The backend is based on the Django *web* framework, written in the Python programming language, while the frontend is written in JavaScript. The theoris presents mains diagrams, including the relational database model, data structure, data examples, pseudo code and original Python code. The installation of Helios on the Linux Apache2 *web* server is described in detail, followed by an analysis of the difference between Helios and another Prêt-à-Voter cryptographic voting system.

Key words: Helios, voting, Cryptography, ElGamal, E2E, apache2, Python, JavaScript, Linux