TUGAS AKHIR – KI141502

IMPLEMENTASI TEKNOLOGI BLOCKCHAIN DAN MULTI PARTY COMPUTATION DALAM SISTEM E-VOTE

R. M ISKANDAR ZULKARNAEN
NRP 5112100101

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc., Ph.D

Dosen Pembimbing II
Baskoro Adi Pratomo, S.Kom., M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

UNDERGRADUATE THESES – KI141502

BLOCKCHAIN TECHNOLOGY AND MULTI PARTY COMPUTATION IMPLEMENTATION IN E-VOTE SYSTEM

R. M ISKANDAR ZULKARNAEN
NRP 5112100101

First Advisor
Prof. Ir. Supeno Djanali, M.Sc., Ph.D

Second Advisor
Baskoro Adi Pratomo, S.Kom., M.Kom.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2016

LEMBAR PENGESAHAN

IMPLEMENTASI TEKNOLOGI BLOCKCHAIN DAN MULTI PARTY COMPUTATION DALAM SISTEM E-VOTE

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
R.M ISKANDAR ZULKARNAEN
NRP: 5112100101

Disetujui oleh Pembimbing Tugas Akhir:

1. Prof. Ir. Supeno Djanali, M.Sc., Ph.D ..... (Pembimbing 1)
   (NIP. *194806191973011001*)

2. Baskoro Adi Pratomo, S.Kom., M.Kom. ..... (Pembimbing 2)
   (NIP. *198702182014041001*)

SURABAYA
JUNI, 2016

IMPLEMENTASI TEKNOLOGI BLOCKCHAIN DAN MULTI PARTY COMPUTATION DALAM SISTEM E-VOTE

Nama Mahasiswa : R.M ISKANDAR ZULKARNAEN
NRP : 5112100101
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Prof. Ir. Supeno Djanali, M.Sc., Ph.D
Dosen Pembimbing 2 : Baskoro Adi Pratomo, S.Kom., M.Kom.

Abstrak

Sistem pemungutan suara di Indonesia yang lebih dikenal dengan nama pemilihan umum (pemilu) sampai saat ini masih dilaksanakan secara manual. Dalam sistem tersebut, dibutuhkan waktu yang relatif lama untuk menghitung suara yang masuk. Selain itu dalam sistem ini juga rawan adanya kecurangan terhadap jumlah suara untuk memenangkan kelompok atau pihak tertentu. Untuk mengatasinya, salah satu solusi adalah menggunakan sistem pemungutan suara elektronik atau dikenal dengan istilah E-Vote. Di beberapa negara, sistem pemungutan suara telah dilakukan dengan menggunakan E-Vote. Bermacam-macam metode telah digunakan dalam sistem E-Vote untuk mengamankan data hasil pemungutan suara, salah satunya dengan menggunakan teknik enkripsi dalam menyimpan data hasil. Namun hal ini masih memiliki kekurangan yaitu apabila kunci untuk melakukan proses enkripsi-dekripsi bocor atau dicuri, maka siapa saja dapat mengubah data yang ada.

Pada tugas akhir ini, penulis mencoba untuk merancang mekanisme penyimpanan data hasil pemungutan suara yang bersifat tamper proof apabila data telah masuk ke dalam sistem. Dengan menerapkan perpaduan metode Blockchain pada Bitcoin yang telah dimodifkasi sesuai kebutuhan dengan Multi Party Computation, mekanisme penyimpanan data yang bersifat tamper

proof dapat diwujudkan. Pemilih memasukkan data dari mesin voting yang tersedia. Data suara yang masuk dari pemilih akan dipecah menjadi beberapa data dengan metode Multi Party Computation dan hasilnya dikenal dengan istilah shares. Kemudian tiap share tadi akan dikirim ke jaringan Blockchain yang berbeda. Di tiap jaringan Blockchain, data share di broadcast ke semua node yang ada dan dilakukan proses Blockchain sehingga terbentuklah prinsip single shared truth. Setelah proses Blockchain selesai, data disimpan dalam bentuk blok dengan format data binary kemudian disinkronisasi dengan database lokal sebagai alat bantu tracking data blok yang disimpan. Untuk memastikan data yang masuk itu benar atau tidak, digunakanlah digital signature sebagai mekanisme validasi data.

Skenario yang diusulkan penulis tersebut terbukti dapat mencegah adanya perubahan data suara yang sudah masuk ke dalam sistem. Apabila ada data yang tidak berasal dari sumber yang benar atau data salah, data tersebut akan dieliminasi melalui beberapa proses validasi. Selain itu, apabila terdapat data yang diubah atau rusak, terdapat salinan data di node lain yang memiliki data yang serupa dengan data yang hilang atau rusak sehingga bisa dijadikan backup data. Diharapkan skenario ini dapat diterapkan dalam sistem E-Vote di masa mendatang.

Kata kunci: bitcoin, blockchain, digital signature, keamanan data, multi party computation, pemungutan suara elektronik, sistem terdistribusi.

BLOCKCHAIN TECHNOLOGY AND MULTI PARTY COMPUTATION IMPLEMENTATION IN E-VOTE SYSTEM

Student's Name	**:**	**R.M ISKANDAR ZULKARNAEN**
Student's ID	**:**	**5112100101**
Department	**:**	**Teknik Informatika FTIF-ITS**
First Advisor	**:**	**Prof. Ir.Supeno Djanali, M.Sc., Ph.D**
Second Advisor	**:**	**Baskoro Adi Pratomo, S.Kom., M.Kom.**

Abstract

Voting system in Indonesia, known as general election, until current date still being held using manual method. This system needs relatively long time to count input vote. Furthermore, it is prone to fraud against the number of votes for making a particular group or party win the election. To overcome this, one of the solutions is we could use electronic voting scheme or we know it as E-Vote. In some countries, E-Vote has been implemented in their election process. Many methods for securing election data have been implemented in E-Vote system since then, one of them is securing stored data using encryption method. However, this method still has drawback. That drawback is when key for encryption-decryption process disclosed or somehow stealed, everyone can tamper the existed data.

In this undergraduate thesis, author is trying to devises mechanism for storing tamper proof data inside the system. By applying both specifically modified Blockchain method in Bitcoin and Multi Party Computation, mechanise for storing tamper proof data can be realized. Voter inputs data from designated vote machine. Incoming data from voter will be separated into many data by using Multi Party Computation and these result known as shares. Afterward, each share will be send to each designated Blockchain network. For each Blockchain network,

share data broadcasted to all nodes and starting Blockchain process to create single shared truth principal. After Blockchain process finished, data will be stored in block form, which has binary format then synchronize it with local database as tracking helper for stored blocks. To validate incoming data, digital signature is used for validation process.

The scenario that author proposed is proven that can prevent vote data tamper attempt inside the system. If there is invalidate incoming data, it would be eliminated by many validation process. Moreover, if there is any tampered or broken data, we can request copy of similar data from another node that has the data as a backup. This scenario is expected to be applied in E-Vote system in the future.

Keywords : bitcoin, blockchain, data security, digital signature, distributed system, electronic voting, multi party computation.

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**Implementasi Teknologi Blockchain dan Multi Party Computation dalam Sistem E-Vote**"

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada :

1. Bapak R. Suprapto dan Ibu Siti Towiyah selaku orang tua penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. R. Ayu Maulidina Iskandariya selaku kakak kandung perempuan dari penulis yang memberikan dukungan kepada penulis selaku adiknya meski di tengah kesibukan yang dijalani dan tugas sebagai calon ibu dan R. Mas Akbar Al Kautsar selaku adik dari penulis yang telah menggantikan tugas kewajiban penulis di rumah di saat penulis menjalani masa studi.
3. Bapak Prof. Ir. Supeno Djanali, M.Sc., Ph.D. dan Bapak Baskoro Adi Pratomo, S.Kom., M.Kom. selaku pembimbing I dan II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Imam Kuswardayan, S.Kom., M.T. selaku dosen wali penulis yang telah memberikan arahan, masukan dan motivasi kepada penulis selama menjalani masa studi di ITS.

5. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS dan segenap dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
6. Teman seperjuangan dalam topik tugas akhir (M. Ardhinata Juari), teman seperjuangan semasa kuliah penulis (Vijay Fathur, Faishal Azka, Reyhan Arief, I Putu Pradnyana, Nabila Tsurayya Silmi), dan teman-teman lain satu angkatan yang sudah memberi dukungan, informasi dan hal lainnya yang membantu penulis menyelesaikan tugas akhir ini.
7. Sahabat penulis lainnya yang tidak bisa disebutkan satu-persatu, yang selalu membantu, menghibur, menghadapi persoalan sulit bersama, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2016

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR PSEUDOCODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dewasa ini perkembangan sistem digital semakin pesat, seiring pula dengan penggunaan peralatan elektronik dalam menyelesaikan suatu permasalahan yang ada di masyarakat. Dalam menyelesaikan permasalahan menggunakan sistem digital, diperlukan perancangan sistem yang tepat dan bisa memenuhi kebutuhan yang ada. Negara Indonesia adalah negara yang menjunjung demokrasi dalam menjalankan sistem pemerintahannya. Dalam sistem demokrasi, dalam mengambil keputusan akan suatu permasalahan biasanya digunakanlah metode *voting* yaitu pengambilan suara terbanyak dari semua pemilih yang ada dalam menentukan pilihan. Dalam periode tertentu, diadakanlah pemilihan umum atau dikenal sebagai pemilu dalam menentukan wakil rakyat. Pemilu ini merupakan salah satu bentuk *voting*. Namun, Sistem pemilihan umum yang ada sekarang, masih tidak praktis dan memerlukan biaya yang relatif mahal.

Oleh karena itu, diajukanlah sistem *E-Vote* dalam penyelenggaraan pemilu yaitu pengambilan suara menggunakan sistem yang dirancang dengan kemampuan komputasi dan terintegrasi dari perangkat elektronik. Tapi hal ini masih rentan akan kecurangan berupa perubahan data hasil *voting* dengan melakukan *hacking* ke sistem yang dapat dilakukan oleh oknum baik dari pihak luar maupun pihak ketiga yang dipercaya dalam menyelenggarakan pemilu, tidak ada bedanya dengan sistem yang ada sekarang. Hal ini dapat diatasi dengan implementasi hasil modifikasi teknologi *Blockchain* yang diterapkan pada *Bitcoin* dan *Multi Party Computation* sehingga hasil *voting* bersifat *non-repudiation* (tidak bisa disangkal). Teknologi *Blockchain* yang ada pada *Bitcoin*, memanfaatkan *hash value* dari data transaksi yang ada kemudian *hash value* tersebut diverifikasi

oleh *node* lain secara berantai sehingga menghasilkan *trust chain* antar node. Hal ini nantinya yang akan dimanfaatkan dalam sistem *E-Vote*. Sedangkan teknologi *Multi Party Computation memanfaatkan* algoritma *Shamir's Secret Sharing* dalam mengevaluasi suatu fungsi secara rahasia dengan memecah data asli menjadi beberapa bagian. Tugas akhir ini mengimplementasikan kedua teknologi ini untuk menciptakan sistem *E-Vote* dengan tingkat integritas data yang tinggi dan aman dari perubahan data dalam sistem.

1.2 Rumusan Masalah

Rumusan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan antara lain:

1. Bagaimana mengimplementasikan teknologi *Blockchain* dan *Multi Party Computation* dalam sistem *E-Vote*?
2. Bagaimana cara menjaga integritas hasil *voting* yang ada pada sistem?
3. Bagaimana cara verifikasi hasil *voting* seorang pemilih apabila hasilnya bersifat anonim?
4. Bagaimana cara menghitung hasil *voting* dari hasil *E-Vote*?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki batasan sebagai berikut:

1. Teknologi yang digunakan adalah *Blockchain* yang telah dimodifikasi dan *Multi Party Computation*.
2. Keamanan berfokus pada *confidentiality* dan *integrity* data hasil pemungutan suara.
3. Implementasi hanya sebatas bagian backend dari sistem.
4. Aplikasi *E-Vote* yang digunakan dalam implementasi hanya sebuah *interface* dan berupa aplikasi berbasis *web*

yang digunakan sebatas untuk pengujian bagian sistem *backend*.

5. Sistem berjalan di *Virtual Private Network (VPN)*.
6. *Digital Signature* yang digunakan menggunakan kriptografi RSA dan *hash* menggunakan SHA256
7. Studi kasus sistem adalah pemilihan suara kandidat pasangan seperti pada pemilihan umum capres-cawapres yang ada di Indonesia

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah mengimplementasikan hasil modifikasi teknologi *Blockchain* yang digunakan pada *Bitcoin* dan *Multi Party Computation* dalam sistem *E-Vote* sebagai mekanisme validasi dan menjaga integritas data hasil pemungutan suara.

1.5 Manfaat

Manfaat dari hasil pembuatan Tugas Akhir ini adalah :

1. Mendapatkan pemahaman implementasi asas pemilu ke dalam sistem *E-Vote*.
2. Mendapatkan wawasan mengenai implementasi *Blockchain* pada *Bitcoin* dan *Multi Party Computation.*
3. Mendapatkan pemahaman mengenai proses *signing* menggunakan *digital signature* sebagai proses validasi data.
4. Mencegah perubahan data yang dapat di lakukan pada data hasil pemungutan suara yang disimpan dalam sistem *E-Vote*.
5. Sebagai rujukan untuk sistem penyimpanan data baru yang bersifat *decentralized*.

1.6 Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir adalah sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal Tugas Akhir ini berisi tentang deskripsi pendahuluan dari Tugas akhir yang akan dibuat. Pendahuluan ini terdiri dari hal yang menjadi latar belakang diajukannya usulan Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, tujuan dari pembuatan Tugas Akhir, dan manfaat dari hasil pembuatan Tugas Akhir. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Subbab metodologi berisi penjelasan mengenai tahapan penyusunan Tugas Akhir mulai dari penyusunan proposal hingga penyusunan buku Tugas Akhir. Terdapat pula subbab jadwal kegiatan yang menjelaskan pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini dilakukan studi literatur mengenai alat dan metode yang digunakan. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal dan dokumentasi dari internet. Hal-hal yang dipelajari yaitu mengenai pelaksanaan Pemilu di Indonesia, sistem *E-Vote*, sistem kriptografi RSA dan *digital signature*, konsep *hash*, *Virtual Private Network* (VPN)*, message queue,* cara kerja *Blockchain* pada *Bitcoin,* dan *Multi Party Computation* yang meliputi *Shamir's Secret Share Algorithm*. *Paper* yang digunakan sebagai referensi utama antara lain *"Decentralizing Privacy : Using Blockchain to Protect Personal Data"*. *Paper* tersebut menjelaskan mengenai implementasi teknologi *blockchain* berupa *public ledger* dalam melindungi privasi sebuah personal data [5]. Dan yang kedua adalah *paper* dengan judul *"An Improved E-Voting Scheme using*

Secret Sharing based Secure Multi-Party Computation" sebagai acuan metode *Multi Party Computation* yang digunakan [4].

1.6.3 Implementasi Mekanisme

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir ke dalam sistem *E-Vote* yang dirancang sesuai dengan alur Pemilu yang ada saat ini. Di bagian *backend server*, disediakan beberapa komputer baik virtual maupun fisikal sesuai jumlah yang dibutuhkan untuk membangun sistem terdistribusi. Komunikasi yang dilakukan antar komputer atau dikenal istilah *node* dalam jaringan, menggunakan *Inter-process message queue.* Sistem *backend* dibangun dengan bahasa pemorgraman Node.js. Untuk uji coba *backend*, dirancang pula bagian *frontend* untuk mesin *voting* menggunakan bahasa C++ dengan rencana kakas bantu yang digunakan Qt Framework dan API (*Application Programming Interface*) *server* menggunakan bahasa Node.js dengan rencana kakas bantu ExpressJs.

1.6.4 Pengujian dan Evaluasi

Pada tahapan ini dilakukan uji coba terhadap aplikasi yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan. Tahap pengujian difokuskan pada pengamanan data yang memenuhi aspek *confidentiality* (menjamin kerahasiaan data yang disimpan), dan *integrity* (memastikan keaslian terhadap data yang disimpan) dalam sistem *E-Vote*.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan Tugas Akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan Tugas Akhir ini.
3. Bab III. Desain dan Perancangan
   Bab ini berisi perancangan metode yang nantinya akan diimplementasikan dan dilakukan uji coba.
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi dari rancangan sistem yang telah dijabarkan pada bab sebelumnya. Penjelasan berupa implementasi hasil modifikasi dari teknologi *Blockchain* dan *Multi Party Computation* di bagian *backend server*, Proses validasi data dengan menggunakan *digital signature,* dan implementasi contoh *frontend* sebagai alat uji coba terhadap cara kerja *backend.*
5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini menjelaskan tahap pengujian sistem dengan memasukkan data dari bagian *frontend*, pengujian API (*Application Programming Interface*) yang dirancang, pengecekan data yang disimpan di *backend*, dan

bagaimana sistem dapat melakukan proses verifikasi dan validasi data yang masuk maupun disimpan dalam sistem.

6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan terhadap rumusan masalah yang ada dan saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan pengimplementasian mekanisme. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap sistem yang akan dibangun, mekanisme yang dimasukkan dalam sistem, dan definisi yang digunakan dalam pembuatan Tugas Akhir.

2.1 *E-Vote*

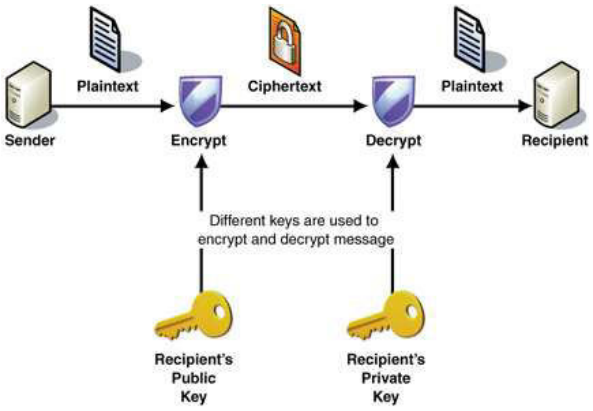
Electronic Vote (E-Vote) adalah sistem pemungutan suara dengan memanfaatkan sarana teknologi informasi atau perangkat elektronik, dimana sebagian atau seluruh proses pelaksanaannya yang terdiri dari pendaftaran pemilih, pemungutan suara, hingga perhitungan suara, dilakukan secara digital. Beberapa negara sudah menerapkan *E-Vote* dalam penyelenggaraan pemilunya, salah satunya adalah Brazil dan India [6]. Di Indonesia sudah ada beberapa daerah yang berhasil menerapkan *E-Vote* meski baru bisa diterapkan dalam lingkup Pilkades, yaitu Kabupaten Jembrana di Bali dan Boyolali di Jawa Tengah [7].

Kelebihan yang dimiliki dari sistem *E-Vote* antara lain mempercepat proses tabulasi suara, mengurangi resiko kesalahan teknis yang dilakukan oleh manusia *(human error)*, dan bisa menghemat pemakaian kertas dan tinta dalam proses pemungutan suara. Selain itu, untuk melakukan kecurangan dalam pemungutan suara berbasis *E-Vote*, dibutuhkan kemampuan di bidang ilmu komputer sebagai dasar ilmu untuk mencoba memanipulasi hasil data *E-Vote*, sehingga mengurangi jumlah oknum potensial yang dapat melakukan kecurangan terhadap hasil pemungutan suara.

2.2 Kriptografi RSA dan Digital Signature

Sistem kriptografi kunci publik atau lebih dikenal dengan istilah kriptografi kunci asimetrik pertama kali diusulkan oleh Diffie dan Helman pada tahun 1976. Prinsip cara kerja sistem ini mirip dengan cara kerja kunci gembok di dunia nyata. Misal ada sebuah brankas berisi data rahasia lalu brankas tersebut di kunci menggunakan gembok yang dimiliki oleh pemilik brankas. Kemudian brankas ini dikirim dengan keadaan terkunci ke penerima brankas. Penerima dapat membuka brankas apabila memiliki kunci yang cocok dengan gembok yang digunakan untuk mengunci brankas.

Secara singkat proses enkripsi dan dekripsi dalam sistem kriptografi ini adalah sebagai berikut. Pengirim pesan berperan sebagai enkriptor dan penerima pesan sebagai dekriptor. Penerima menunjukkan kunci publiknya, namun tetap menjaga kerahasiaan kunci privatnya. Pengirim mengenkripsi teks asli dengan kunci public miliknya sehingga menghasilkan teks sandi yang kemudian dikirim ke penerima melalui saluran yang tidak aman. Terakhir, penerima mendekripsi teks sandi dengan kunci privatnya untuk mendapatkan teks asli.



Gambar 2.1 Sistem kriptografi asimetrik

Salah satu jenis kriptografi asimetrik adalah kriptografi RSA. kriptografi ini dirumuskan oleh tiga orang mahasiswa dari Massachusetts Institute of Technology (MIT) : Ron Rivest, Adi Shamir, dan Len Adleman pada tahun 1977, meskipun tiga tahun sebelumnya dirumuskan terlebih dahulu oleh Clifford Cock dan baru dipublikasikan oleh badan sandi Inggris pada tahun 1997. Kriptografi ini menggunakan algoritma enkripsi dan dekripsi yang bersandar pada fungsi satu arah (*one way function).* Fungsi tersebut dibangun oleh fungsi eksponensial modular. Terdapat tiga algoritma pada sistem kriptografi ini yaitu : pembuatan kunci, enkripsi dan dekripsi.

Pertama, penerima pesan pendekripsi membuat pasangan kunci publik dan kunci privat miliknya. Pasangan kunci tersebut dihasilkan dengan algoritma berikut.

1) Hasilkan bilangan prima besar (misalnya *p* dan *q*) dengan menggunakan algoritma pengujian bilangan prima (misalnya algoritma Miller-Rabin).
2) Kalikan p dan q sehingga didapat hasil kali n untuk faktorisasi kunci publik dan kunci privat serta digunakan sebagai panjang kunci.
3) Hitung $\varphi(n) = (p-1)(q-1)$.
4) Pilih sebuah bilangan bulat e secara acak sebagai eksponen kunci publik hingga memenuhi $1 < e < \varphi(n)$ dan factor persekutuan terbesar (FPB) dari e dan $\varphi(n)$ adalah 1.
5) Hitung $d \equiv e^{-1} \left(mod\ \varphi(n)\right)$.
6) Tetapkan (e, n) sebagai kunci privat dan d sebagai kunci publik.

Setelah pasangan kunci penerima pesan dihasilkan, maka sembarang pengirim pesan dapat menggunakan kunci publik penerima untuk mengirim pesan teks sandi kepada penerima dengan rumus:

$$C = P^e \ mod\ n$$

Dengan C adalah *cipher text* atau teks hasil enkripsi dan P adalah *plain text* atau teks asli.

Akhirnya setelah pesan teks sandi sampai pada penerima, penerima dapat menggunakan kunci privatnya untuk mengembalikan pesan teks hasil enkripsi menjadi teks asli dengan rumus:

$$P = C^d \bmod n$$

Besar p dan q disarankan $\geq$ 512 bit sehingga besar n minimal 1024 bit agar kriptografi ini menjadi aman dan n susah untuk difaktorkan.

Pemanfaatan lain dari kriptografi kunci asimetrik adalah sebagai *digital signature* [8]. *Digital signature* ini memiliki manfaat sebagai metode untuk memastikan bahwa pesan yang dikirim benar dan berasal dari pemilik yang sebenarnya. Proses yang dilakukan dalam melakukan *signing* menggunakan *digital signature* adalah sama dengan proses enkripsi-dekripsi yang dijelaskan sebelumnya, namun kunci yang digunakan untuk mengenkripsi pesan adalah kunci privat dari pengirim pesan sedangkan kunci yang digunakan untuk mendekripsi pesan adalah kunci publik yang disebar kepada orang lain sehingga orang lain dapat melakukan validasi dari pesan yang dikirim memang berasal dari pemilik kunci privat. Karena semua orang dapat melihat isi dari pesan yang dienkripsi dengan kunci privat pengirim, untuk memastikan bahwa data tersebut berasal dari pengirim biasanya pesan yang dienkripsi merupakan *hash value* dari pesan yang dikirim oleh pemilik kunci privat.

2.3 Konsep *Hash* (SHA-256)

Fungsi *hash* adalah sebuah fungsi yang masukannya berupa pesan dengan panjang sembarang dan hasil keluarannya berupa nilai *hash* (*hash value*) dengan panjang yang tetap dan biasa disebut *digest* atau *message digest*. Fungsui *hash* dapat dituliskan dengan persamaan:

$$h = H(M)$$

dimana *H* merupakan fungsi *hash* yang menerima masukan berupa pesan *M* untuk menghasilkan keluaran berupa nilai *hash* atau *digest h*.

Fungsi hash dapat digunakan untuk mewujudkan salah satu prinsip keamanan jaringan dan informasi yaitu keutuhan data (*data integrity*). Fungsi tersebut berguna sebaga parameter yang mengecek ada tidaknya perubahan terhadap data sebelum dan sesudah data itu dikirim atau disebar ke tempat lain dalam satu komputer maupun melalui jaringan.

Salah satu cara untuk menghasilkan *digest* adalah dengan memanfaatkan penggunaak fungsi kompresi secara berulang terhadap sebuah pesan yang telah dibagi sebelumnya menjadi sejumlah blok pesan. Salah satu fungsi *hash* yang menggunakan cara tersebut adalah SHA (*Secure Hash Algorithm*).

SHA adalah fungsi *hash* satu arah yang dibuat oleh National Institute of Standards and Technology (NIST). SHA didasarkan pada MD4 yang dibuat oleh Ronald L.Rivest. SHA sampai saat ini memiliki 4 versi, mulai dari SHA-0, SHA-1, SHA-2, dan SHA-3. SHA-2 memiliki beberapa varian, salah satunya adalah SHA-256.

SHA-256 adalah fungsi *hash* SHA yang menghasilkan nilai *hash* sepanjang 256 bit dengan iterasi terhadap sejumlah blok pesan yang masing-masing panjangnya 512 bit.

Secara singkat, langkah-langkah untuk menghasilkan *digest* adalah sebagai berikut. Misalkan ada pesan P. P tersebut direpresentasikan ke bilangan biner tiap karakter pesannya. Selanjutnya adalah menambah *padding* terhadap P berupa satu bit karakter "1", diikuti sejumlah bit karakter "0" dan representasi biner dari panjang *byte* pesan, agar panjang P tepat berkelipatan 512 bit. Representasi pesan P ber-*padding* tersebut kemudian diubah menjadi bilangan heksadesimal. Lalu, memanggil fungsi kompresi dengan parameter masukan berupa delapan *word* nilai *digest* awal yang diambil dari bagian pecahan pada akar kuadrat delapan bilangan prima pertama dengan panjang *word* masing-masing 32 bit dan pesan P ber-*padding*. Dalam fungsi kompresi,

dilakukan ekspansi *word* dari pesan P ber-*padding* tersebut dengan jumlah *word* sebanyak 64 yang masing-masing panjangnya 32 bit dan komputasi ronde sebanyak 64 ronde, yang mana hasilnya didapatkan dengan melakukan penjumlahan *digest* dikalikan dengan mod 2^{32}. Setelah tidak ada blok pesan lagi yang diproses, maka hasil dari fungsi kompresi dijadikan *digest* yang dicari dengan panjang 256 bit [8][9].

2.4 *Virtual Private Network* (VPN)

Secara simpel, *Virtual Private Network* (VPN) didefinisikan sebagai emulasi *Public Switched Telephone Network* (PSTN) yang didedikasikan terhadap jaringan privat [10]. jaringan privat ini dibangun diatas jaringan lainnya yang dianggap tidak aman. Konsep VPN ini dirumuskan oleh D. Wood, V.Stoss, L.Chan-Lizardo dkk dari AT&T and Philips Telecommunications UK Ltd.

Dengan adanya VPN, komputer satu dengan komputer lainnya dapat terhubung ke jaringan privat dengan pesan yang telah dienkripsi antara *peer* dengan *server* VPN kemudian dikirim ke *peer* lainnya dengan pesan yang sudah dienkripsi. VPN biasanya memanfaatkan penggunaan kriptografi sebagai mekanisme pengamanan data yang dikirim dan diterima di dalam jaringan privat.

Model keamanan VPN menerapkan kerahasiaan data dengan penggunaan kriptografi, autentikasi untuk mengakses jaringan privat (bisa dilakukan dengan password maupun dengan *digital certificates*), dan keutuhan data dengan menggunakan *digital signature*.

Banyak macam-macam dari protokol VPN yang telah dikembangkan dan salah satunya adalah dengan menggunakan *Transport Layer Security/Secure Socket Layer* (TLS/SSL). Tujuan dari TLS/SSL protokol ini adalah untuk menciptakan privasi dan keutuhan data antara aplikasi yang berjalan di dalam komunikasi dua komputer yang terhubung melalui protokol ini.

Data yang dikirim melalui protokol TLS/SSL dienkripsi dengan kunci simetrik yang memiliki kunci unik dan dibuat berdasarkan *shared secret* yang dimiliki dua komputer yang terhubung. Salah satu program yang menggunakan VPN dengan protokol TLS/SSL ini adalah OpenVPN.

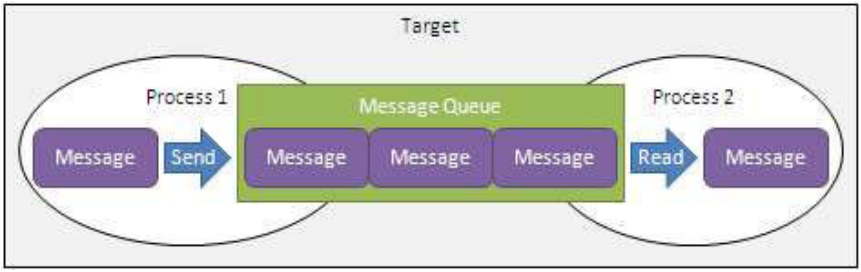
Gambar 2.2 Arsitektur Jaringan TLS/SSL VPN

2.5 *Message Queue*

Message queue merupakan salah satu paradigma dari komunikasi antar proses (*interprocess communication*) yang digunakan dalam teknik pengembangan perangkat lunak. *Message queue* menyediakan komunikasi protokol secara asinkron, maksudnya pengirim dan penerima data melalui *message queue* tidak perlu berinteraksi (mengirim dan menerima) pesan dalam waktu yang bersamaan.

Prinsip kerja dari *message queue* adalah membuat list antrian pesan yang dikirim dari satu proses ke proses lain. Pesan yang dikirim diberikan ID yang nantinya akan dipanggil dan di proses apabila pesan tersebut mendapat giliran untuk dipanggil. Cara kerja mirip dengan *First In First Out* (FIFO), namun *message queue* lebih mendukung fungsionalitas yang lain seperti *drop* data yang ada di antrian, mendahulukan data yang lebih cepat di proses, dan macam-macam fungsionalitas lain yang bisa dimanfaatkan. *Library message queue* yang sudah dikembangkan

antara lain ϕMQ (ZeroMQ), RabbitMQ, ActiveMQ dan masih banyak lainnya.



Gambar 2.3 *Message Queue* secara umum

2.6 Basis Data NoSQL

NoSQL (Not only SQL) adalah istilah umum yang merujuk pada penyimpanan data yang tidak mengikuti aturan dari basis data relasional contohnya MySQL. Konsep yang diterapkan dalam basis data NoSQL ini merupakan pengembangan dari model basis data yang sudah ada yaitu antara lain model hirarki, model jaringan, model multidimensional, dan model yang berorientasi objek.

NoSQL ini merupakan metode penyimpanan yang cocok untuk menyimpan data dalam skala besar dan terus berkembang karena meski begitu, metode ini tetap mempunyai kinerja dan kecepatan yang baik. Selain itu, NoSQL memiliki sifat skema fleksibel yang memungkinkan pengembang aplikasi tidak mengikuti pola yang sama di setiap kelasnya. NoSQL juga tidak menggunakan *operational join* yang berdampak pada kecepatan pengambilan data. Kelebihan lainnya yaitu memungkinkan *data update* terjadi di berbagai *server* sekaligus dengan cepat.

Jenis penyimpanan data pada NoSQL secara umum terbagi menjadi beberapa yaitu *Column-Oriented*, *Document Store*, *Key-value*, dan *Graph*. Contoh aplikasi basis data NoSQL ini salah satunya adalah MongoDB.

2.7 Teknologi *Blockchain*

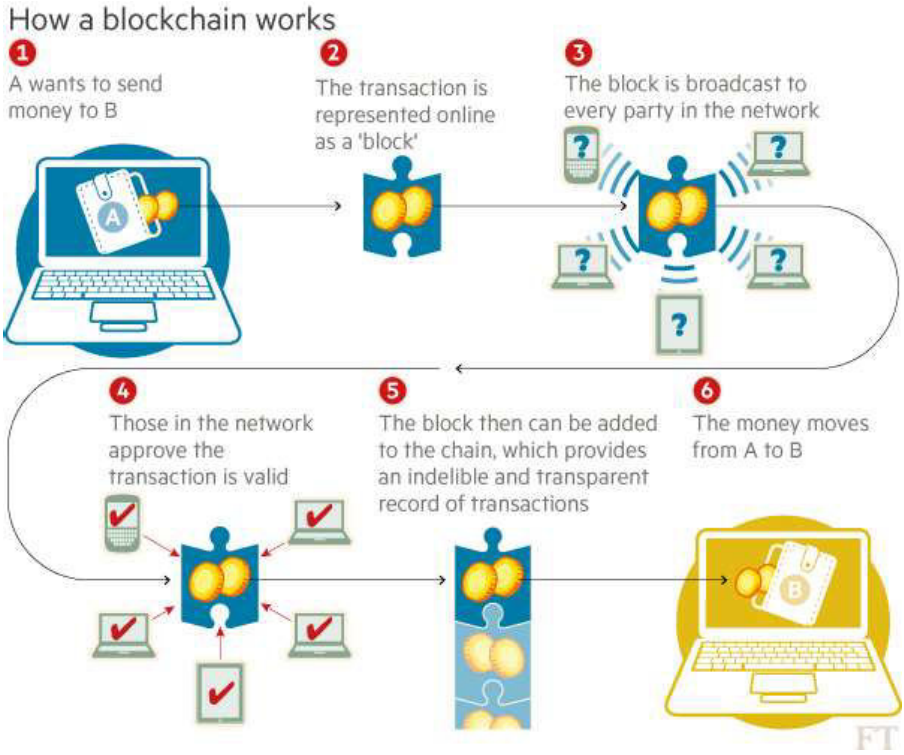
Teknologi *Blockchain* adalah teknologi yang digunakan dalam mewujudkan *peer-to-peer electronic cash system, Bitcoin*. *Blockchain* digunakan dalam *Bitcoin* agar transaksi yang dilakukan oleh suatu pihak yang membayar dapat dipercaya oleh pihak yang menerima pembayaran tanpa harus melalui *trusted third party* yang mengawasi jalannya transaksi namun memanfaatkan konsep *consensus system* [3].

Teknologi *blockchain* ini memanfaatkan *hash value* dari transaksi atau proses yang dilakukan pada node sebelumnya, *digital signature*, dan *distributed network* untuk dapat menjaga integritas data yang ada sehingga data yang disimpan ke dalam *blockchain* menjadi *tamper proof* dan terpercaya meski tidak melalui *trusted third party* dalam menjalankan prosesnya. *Blockchain* bertindak sebagai *public ledger* [5] dengan prinsip *single shared truth* dimana hal ini dibutuhkan di dalam sistem *E-Vote.*

Sebuah *Blockchain* adalah database transaksi bersama oleh semua *node* yang berpartisipasi dalam sistem pada protokol *Bitcoin*. Sebuah salinan lengkap dari *Blockchain* mengandung setiap transaksi yang pernah dilakukan dalam mata uang *Bitcoin*. Dengan informasi ini, seseorang dapat mengetahui berapa banyak nilai milik masing-masing alamat pada setiap *node* dalam *history*.

Untuk bisa memodifikasi data yang ada pada *Blockchain* ini, penyerang harus melakukan *backtrace* data dari awal *node* tempat memulai proses *Blockchain* ini kemudian mengulang proses tersebut di semua *node* yang terlibat dalam sistem. Untuk bisa meyakinkan sistem bahwa modifikasi data yang dilakukan seolah-olah adalah data yang benar, seorang penyerang harus mengubah data di semua *node* yang terhubung dalam suatu *chain* pada sistem agar sistem mengakui kebenarannya. Hal ini yang membuat data di *Blockchain* bersifat *tamper proof* dan memiliki

integritas tinggi karena melakukan *hacking* di banyak *node* dianggap impraktikal.



Gambar 2.4 *Blockchain* dalam *Bitcoin*

Mengenai algoritma pada teknologi *Blockchain* ini dijelaskan lebih lanjut pada Gambar 2.5. Algoritma yang dijelaskan ini digunakan pada sistem *Bitcoin* atau *Crypto currency* lainnya yang merujuk pada *paper* mengenai *peer-to-peer electronic cash system* [3].

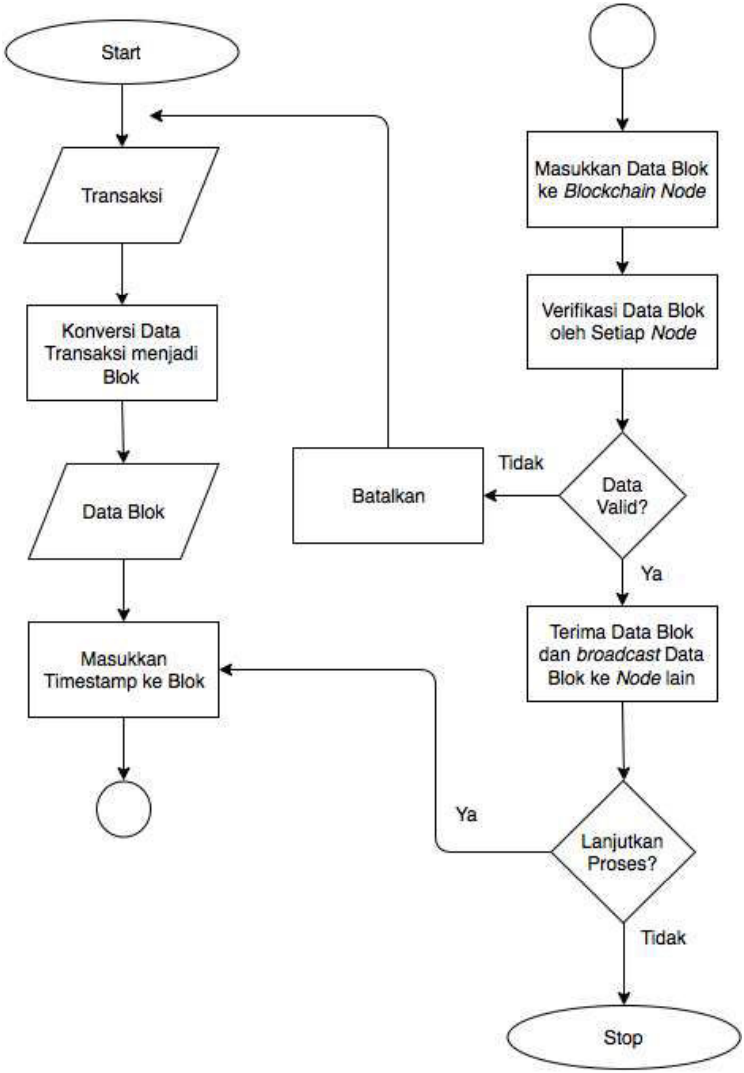
2.8 *Multi Party Computation*

Multi Party Computation adalah metode komputasi untuk menghitung suatu fungsi yang dilakukan oleh beberapa *node* yang memiliki input masing-masing dengan menjaga input tersebut tetap bersifat *private* atau rahasia namun hasilnya bisa diketahui oleh semua pihak yang terlibat [2].

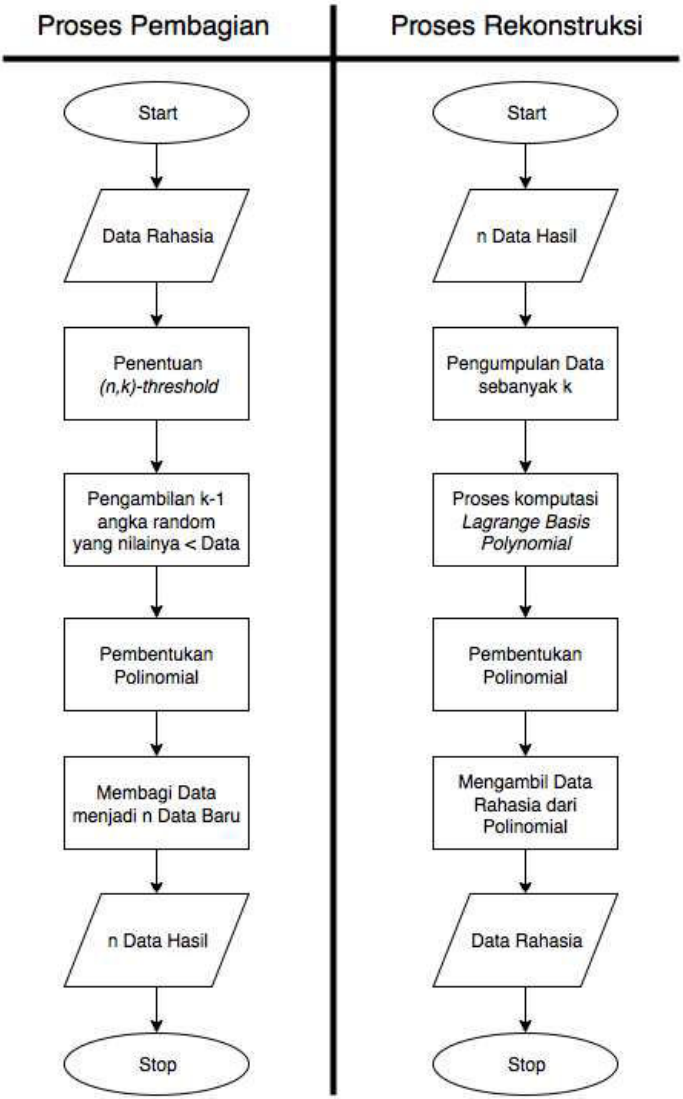
Multi Party Computation ini memanfaatkan algoritma yang dikembangkan oleh Adi Shamir dikenal dengan *Shamir's Secret Sharing* [1]. Algoritma ini memanfaatkan *polynomial interpolation* (*lagrange interpolation*) untuk dapat berbagi rahasia dengan orang lain. Alur dari algoritma ini adalah data rahasia dibagi menjadi n bagian. Ditentukan *k* yang kurang dari n untuk dapat mengetahui data rahasia yang ada. Skema ini disebut skema *(n,k)-threshold* dalam algoritma *Shamir's Secret Sharing*.

Di dalam metode *Multi Party Computation,* algoritma *Shamir's Secret Sharing* digunakan untuk menyembunyikan data input yang diberikan dengan cara membaginya menjadi n data baru. Meski begitu sistem bisa mengetahui dan memasukkan input apabila diperoleh sejumlah k data baru tadi. Hal ini lebih aman daripada menyimpan *raw data* pada sistem.

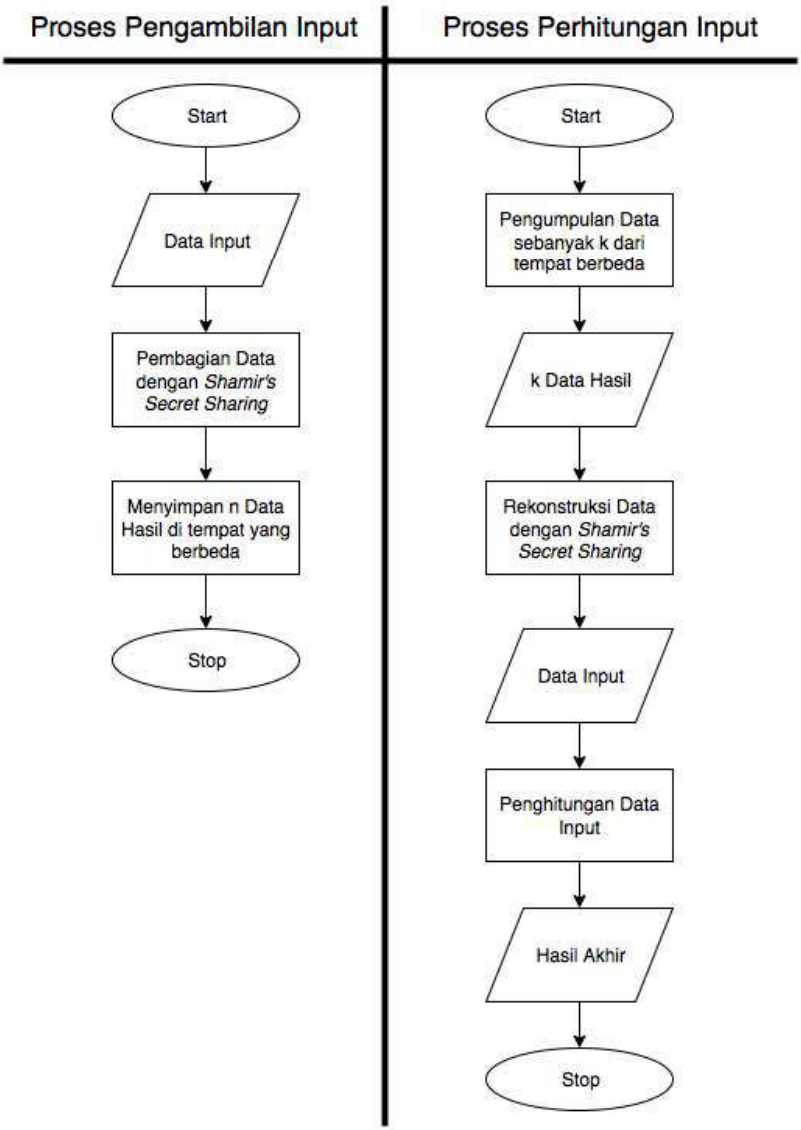
Polinomial yang dihasilkan dari algoritma *Shamir's Secret Sharing* ini bernilai acak sedangkan rahasia yang ingin disimpan dijadikan nilai konstanta dalam polinomial tersebut. Misal untuk membentuk suatu polinomial $y = ax^2 + bx + c$, nilai *a dan b* merupakan nilai acak, sedangkan nilai *c* merupakan nilai rahasia yang ingin disimpan. Kemudian data *share* merupakan data koordinat yang dibentuk dari polinomial ini dengan menggunakan permisalan nilai *x*. Data *share* berupa data koordinat dari hasil polinomial yang dibentuk, jadi nantinya data *share* berupa koordinat [*x,y*]. Algoritma *Shamir's Secret Share* dijelaskan secara umum pada Gambar 2.6 dan *Multi Party Computation* dijelaskan di Gambar 2.7.



Gambar 2.5 Diagram alur proses *Blockchain* secara umum



Gambar 2.6 Diagram alur proses *Shamir's Secret Sharing*



Gambar 2.7 Diagram alur proses *Multi Party Computation*

BAB III
DESAIN DAN PERANCANGAN

Bab ini membahas mengenai perancangan dan pembuatan sistem perangkat lunak. Perancangan yang dijelaskan pada bab ini meliputi perancangan alur sistem, perancangan arsitektur sistem, perancangan basis data, perancangan data, perancangan protokol *backend*, perancangan *frontend* & API, dan perancangan antarmuka.

3.1 Perancangan Alur Sistem secara Umum

Pada Tugas Akhir ini dibangun sistem *backend* penyimpanan data dalam *E-Vote* dengan menggunakan perpaduan hasil modifikasi teknologi *Blockchain* pada *Bitcoin* dan *Multi Party Computation*. Selain itu, dibangun juga mekanisme pengamanan data sebelum data masuk dan disimpan ke sistem *backend*, API *server,* dan contoh aplikasi *frontend* sebagai sarana uji coba terhadap sistem *backend* yang telah dibangun. Untuk skenario yang diujikan, dirancang alur sistem yang menyerupai pemilihan umum yang biasa dilakukan tiap tahunnya di Indonesia namun dengan beberapa proses yang diikuti dengan penggunaan fitur *E-Vote.*

Gambar 3.1 menunjukkan alur sistem secara umum. Tahapan-tahapan yang terjadi selama proses pemungutan suara adalah sebagai berikut.

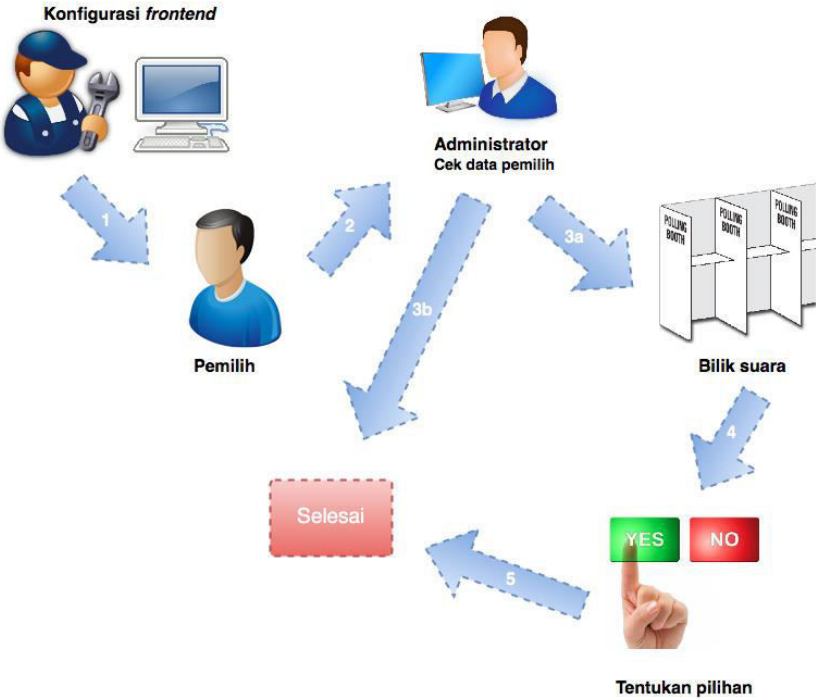
1. Pengawas sistem atau *administrator*, petugas, dan saksi di Tempat Pemungutan Suara (TPS) melakukan persiapan aplikasi di mesin *voting*.
2. Pemilih datang di TPS dengan membawa Kartu Tanda Penduduk (KTP). Pemilih menuju *administrator* untuk mengecek data KTP milik pemilih.
3. a) Jika data pemilih terdapat dalam basis data dan belum pernah memberikan suara sebelumnya, maka pemilih diizinkan untuk memilih, kemudian sistem akan

memberikan kode acak yang nantinya bisa digunakan untuk memilih. Kode ini berlaku sebagai *token*. Metode autentikasi pemilih ini bisa digantikan dengan metode yang lebih aman jika diinginkan, seperti dengan scan sidik jari, menggunakan modul *Near Field Communication* (NFC), dan lain-lain. Kemudian pemilih menuju ke bilik suara.

b) Jika data pemilih tidak tercatat dalam basis data sebagai pemilih atau pemilih sudah memberikan suara sebelumnya, dilakukan pengecekan data terlebih dahulu. Apabila telah dikonfirmasi bahwa pemilih tidak terdaftar atau sudah memilih, maka pemilih tersebut tidak boleh memilih lagi.

4. Pemilih memasukkan Nomor Induk Kependudukan (NIK) dan *token* yang sebelumnya telah diterima dari sistem ke aplikasi yang berjalan di mesin *voting* kemudian memilih pasangan calon sesuai keinginan. Mesin *voting* ini memiliki konfigurasi yang unik dan pasangan kunci asimetrik yang unik pula, sehingga suara hanya dapat masuk melalui mesin *voting* yang sudah terdaftar di basis data sebelumnya
5. Seusai memilih, *administrator* bisa mengecek bahwa pemilih berhasil memberikan suaranya di antarmuka sistem yang disediakan. Antarmuka yang disediakan bisa berupa halaman *web* atau bisa pula berupa program yang sudah dibuat sebelumnya dan memanfaatkan API untuk berhubungan dengan *server* pusat. Hal ini tergantung dari implementasi yang dibuat oleh pengembang sistem.

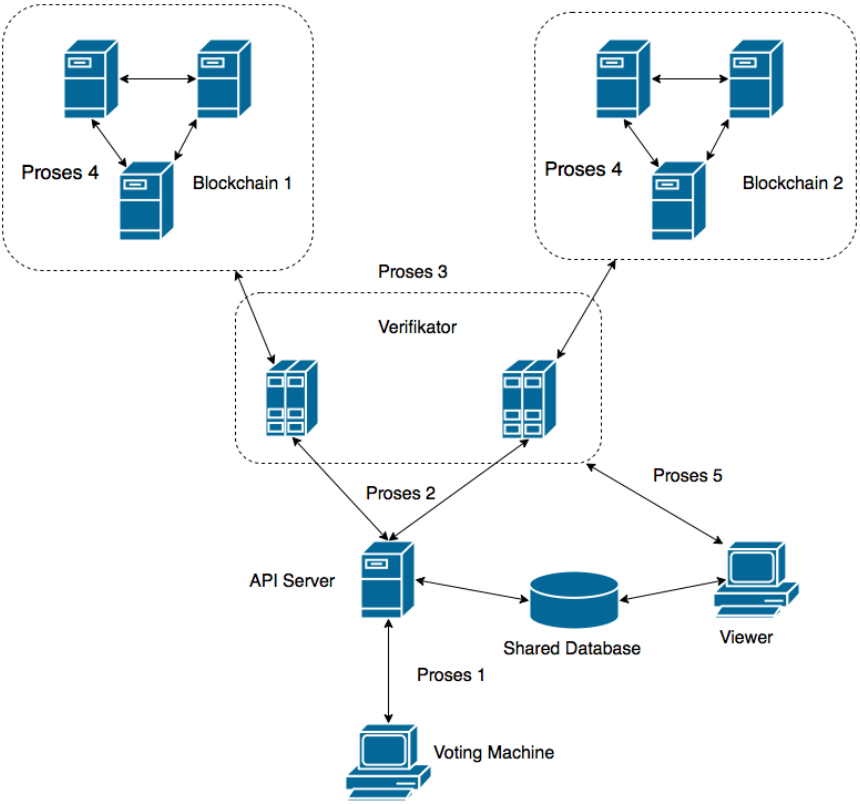
Mesin *voting* diletakkan di bilik suara yang disediakan sedangkan untuk pengecekan data pemilih, disediakan pula komputer untuk *administrator* mengecek data pemilih yang terdaftar dalam sistem. Komputer ini terhubung ke jaringan VPN yang sudah disediakan sebelumnya.



Gambar 3.1 Alur pemungutan suara secara umum dengan *E-Vote*

3.2 Perancangan Arsitektur Sistem

Karena sistem dalam rancangan Tugas Akhir ini membutuhkan beberapa komputer untuk dapat melakukan tugasnya, perlu dirancang gambaran arsitektur sistem yang akan dibangun. Sistem ini menerapkan konsep sistem terdistribusi (*distributed system*) yaitu beberapa komputer melakukan suatu pekerjaan dan saling terhubung satu sama lain untuk dapat melakukan pekerjaan sesuai kebutuhan yang diperlukan dari sistem tersebut. Gambar 3.2 menggambarkan arsitektur yang telah dirancang secara umum untuk memenuhi kebutuhan sistem.



Gambar 3.2 Arsitektur Sistem *E-Vote* yang dibangun

Sistem tersebut berjalan diatas jaringan VPN sehingga semua komunikasi sudah terenkripsi dan berikut merupakan penjelasan tiap *node* dari Gambar 3.2.

- *Voting Machine* merupakan *node* yang berhubungan langsung dengan pemilih. Di mesin ini nantinya pemilih memasukkan suara melalui antarmuka yang disediakan. Data yang masuk dikirimkan melalui jaringan ke API *server*.

- API *server* merupakan *node* yang berhubungan dengan mesin *voting* dan mesin verifikator. Tugas dari API *server* yaitu menyediakan data *ballot* untuk mesin *voting*, melakukan verifikasi mesin *voting*, melakukan validasi data yang masuk dari mesin *voting*, dan meneruskan data ke verifikator.
- Verifikator merupakan *node* yang bertugas melakukan *filter* data yang masuk ke *blockchain*, sebagai *broadcaster* pesan di *blockchain*, dan pembuat *genesis block* yaitu blok data yang belum pernah terdaftar dalam *blockchain* sebelumnya.
- *Node Blockchain* merupakan *node* yang tergabung dalam jaringan *Blockchain* yang melakukan verifikasi data blok terhadap data yang masuk dan di *broadcast* oleh verifikator. Tiap *node blockchain* memiliki basis data local tersendiri sebagai alat bantu *tracking* data blok.
- *Viewer* atau *Observer* merupakan *node* yang bertugas sebagai antarmuka data blok yang disimpan di dalam *blockchain* dan juga sebagai antarmuka *administrator* untuk menjalankan fungsi *checking* pemilih dan menghasilkan *token*. *Viewer* ini berjalan sebagai *Web Server Service*.

Penjelasan proses yang dilakukan didalam sistem secara urut adalah sebagai berikut.

1. Mesin *voting* melakukan proses autentikasi dan input data ke API *server*. Autentikasi dilakukan dengan *signing and verify procedure* menggunakan metode enkripsi dengan pasangan kunci asimetrik.
2. API *server* memastikan data berasal dari mesin *voting* yang sudah terdaftar dan suara berasal dari pemilih yang berhak dan belum menggunakan hak suaranya. API *server* memecah data pilihan pemilih menjadi *shares* dengan metode *Multi Party Computation* kemudian

mengirim *share* yang berbeda ke tiap jaringan *Blockchain* melalui verifikator.

3. Verifikator memastikan data yang masuk benar dari API *server* dan kemudian memasukkan data baru ke dalam blok. Apabila blok ini belum dilakukan proses *Blockchain,* maka pada *interval* tertentu, verifikator akan melakukan *broadcast* data blok ke semua node yang terhubung dalam jaringan *Blockchain*.
4. *Node* di *Blockchain* melakukan verifikasi data blok yang masuk melalui pengecekan *header hash*, *merkle root*, dan *digital signature* pengirim. Kemudian dilakukanlah proses yang dikenal dengan istilah *mining* dalam *Bitcoin*, yaitu membuat data blok baru dengan kriteria memenuhi *target difficulty* yang di set sebelumnya. Yaitu dengan cara melakukan *increment nounce* sampai memenuhi *target* atau *difficulty*. Hal ini yang dijadikan *proof of work* dalam *Blockchain* sehingga untuk membuat suatu rantai *hash* memerlukan waktu yang relatif lama, apalagi jika dilakukan sebanyak *node* yang ada di dalam sistem.
5. *Viewer* atau *Observer* bertugas untuk monitoring data yang ada di *backend* dan melihat hasil suara secara langsung tanpa mengetahui data tersebut memiliki ID apa atau terletak di bagian mana blok. Sistem *Blockchain* menganggap blok dengan chain yang paling panjang merupakan blok paling aman, apabila blok tersebut telah dimodifikasi atau rusak, maka sistem akan meminta data blok lain yang disimpan di *node* lain dalam chain sehingga menemukan data yang benar dan sesuai dengan proses *Blockchain*.

3.3 Perancangan Basis Data

Pada rancangan sistem dalam Tugas Akhir ini, basis data yang digunakan adalah basis data NoSQL. Dalam sistem yang dibangun, terdapat basis data berbagi *(shared database*) yang

digunakan oleh beberapa komputer untuk menyediakan data yang berkaitan dengan pemungutan suara, misal data pemilih, data mesin *voting* dan lainnya, serta basis data lokal (*local database*) untuk keperluan *tracking* data blok pada tiap komputer yang berada dalam jaringan *Blockchain* di sistem *backend* tempat menyimpan hasil suara.

3.3.1 Perancangan Skema *Collection Shared Database*

Perancangan sistem ini membutuhkan basis data untuk menyimpan seluruh data berkaitan dengan pemungutan suara. Sebelum pembuatan basis data dilaksanakan, dibuatlah permodelan skema *collection* secara garis besar. *Collection* memiliki peran sama dengan *table* dalam basis data relasional. Berikut adalah skema masing-masing basis data yang ada.

- *Collection citizen*
  Collection ini berfungsi untuk menyimpan data warga Negara yang terdaftar sebagai pemilih tetap.

Tabel 3.1 Penjelasan Skema pada *Collection Citizen*

Nama Atribut / Field	Tipe Data	Keterangan
NIK	String	Nomor ID pemilih
Photo	String	Foto pemilih dalam base64 format
Name	String	Nama pemilih
Born_date	Date	Tanggal lahir
Born_place	String	Tempat lahir
Marital_status	String	Status pernikahan
Region	String	Daerah asal pemilih
City	String	Kota domisili pemilih
Session	String	*Hash* dari string

Nama Atribut / Field	Tipe Data	Keterangan
		autentikasi acak dari sistem untuk mendapatkan key yang diberikan secara acak pula. *Hash* menggunakan modul Bcrypt.
Auth_string	String	*Hash* dari string autentikasi acak dari sistem. *Hash* menggunakan modul Bcrypt.

- *Collection vmclient*
  Collection ini berfungsi untuk menyimpan data mesin *voting* yang terdaftar sebagai mesin diakui berasal dari pihak penyelenggara.

Tabel 3.2 Penjelasan Skema pada *Collection vmclient*

Nama Atribut / Field	Tipe Data	Keterangan
Identifier	String	Nomor ID pemilih
Name	String	Nama mesin *voting*
Authority	String	Nama penanggung jawab mesin *voting*
Password	String	Passsword untuk autentikasi mesin *voting*
Region	String	Daerah tempat mesin *voting*

Nama Atribut / Field	Tipe Data	Keterangan
City	String	Kota tempat mesin *voting*.
Place_number	String	Kode pos tempat mesin *voting*
Public_key	String	Kunci public dari mesin *voting*.
Auth_string	String	String acak sebagai autentikasi mesin *voting*
Session	String	*Hash* dari string autentikasi acak dari sistem. Hash menggunakan modul Bcrypt.

- *Collection ballot*
  Collection ini berfungsi sebagai menyimpan data yang berlaku sebagai surat suara elektronik yang terdiri dari data calon kandidat.

Tabel 3.3 Penjelasan Skema pada *Collection Ballot*

Nama Atribut / Field	Tipe Data	Keterangan
Event_id	String (8 karakter)	Event ID yang sedang berlangsung
Title	String	Nama mesin *voting*
Candidate_number	String	Nama penanggung jawab mesin *voting*
Candidate_name	String	Passsword untuk autentikasi mesin *voting*
Candidate_picture	String	Daerah tempat

Nama Atribut / Field	Tipe Data	Keterangan
		mesin *voting*
Additional_name	String	Kota tempat mesin *voting*.
Additional_picture	String	Kode pos tempat mesin *voting*
Description	String	Kunci public dari mesin *voting.*
Vote_code	String	ID suara tiap calon yang unik.
Issued	Date	Tanggal dibuat.

- *Collection* verifikator
  Collection ini berfungsi sebagai menyimpan data verifikator yang ada sebagai wakil tiap jaringan *Blockchain* yang ada.

Tabel 3.4 Penjelasan Skema pada *Collection* Verifikator

Nama Atribut / Field	Tipe Data	Keterangan
Identifier	String	ID dari verifikator
Name	String	Nama verifikator
Auth_string	String	String autentikasi acak yang dimiliki verifikator dan disepakati oleh sistem.
Public_key	String	Passsword untuk autentikasi mesin *voting*
Ip_address	String	Daerah tempat

Nama Atribut / Field	Tipe Data	Keterangan
		mesin *voting*
Port	String	Kota tempat mesin *voting*.
Additional_picture	String	Kode pos tempat mesin *voting*
Description	String	Kunci public dari mesin *voting*.
Vote_code	String	ID suara tiap calon yang unik.
Issued	Date	Tanggal dibuat.

- *Collection voter*
  Collection ini berfungsi sebagai menyimpan data pemilih yang sudah memilih atau memberikan suara sebelumnya.

Tabel 3.5 Penjelasan Skema pada *Collection Voter*

Nama Atribut / Field	Tipe Data	Keterangan
NIK	String	ID dari pemilih
Vote_machine_id	String	ID dari mesin *voting* tempat *user* memilih.
Session	String	*Hash* dari string autentikasi acak yang diberikan kepada mesin *voting* saat *user* memilih. *Hash* menggunakan modul Bcrypt.

Nama Atribut / Field	Tipe Data	Keterangan
Timestamp	Date	Waktu pemilih memasukkan suara.
Sign	String	*Digital signature,* berisi *hash* dari NIK pemilih yang dienkripsi oleh kunci privat mesin *voting*.

3.3.2 Perancangan Skema *Collection Local Database*

Di jaringan *Blockchain* sistem ini, digunakan basis data lokal sebagai alat bantu *tracking* seluruh data blok hasil pemungutan suara yang disimpan di *node* yang bersangkutan. Sebelum pembuatan basis data dilaksanakan, dibuatlah permodelan skema *collection* secara garis besar untuk basis data lokal tiap *node* di jaringan *Blockchain*.

1) Basis data di verifikator.
   - *Collection Block*
     Collection ini berfungsi sebagai menyimpan data blok (*genesis block,* blok yang pertama kali dibuat) yang disimpan untuk mempermudah *tracking*.

Tabel 3.6 Penjelasan Skema pada *Collection Block* di Verifikator

Nama Atribut / Field	Tipe Data	Keterangan
Name	String	Nama dari *file* blok.
Header_hash	String	*Hash value* dari *header blok*

Nama Atribut / Field	Tipe Data	Keterangan
Prev_block_name	String	Nama dari *file* blok sebelumnya dari blok ini.
Event_id	String (8 Karakter)	ID Event yang sedang berlangsung.
Total_data	Number	Jumlah data yang ada di blok.
Chain_order	Array, terdiri dari Object dengan parameter : id & hash	Menyimpan urutan dari *Blockchain* blok yang bersangkutan.
Verified	Number	Sebagai penanda bahwa blok tersebut valid setelah diuji.

- *Collection Miner*
  Collection ini berfungsi sebagai menyimpan data miner yang terlibat dalam jaringan *Blockchain* disimpan untuk mempermudah *tracking*.

Tabel 3.7 Penjelasan Skema pada *Collection Miner* di Verifikator

Nama Atribut / Field	Tipe Data	Keterangan
Identifier	String	ID dari *miner*, berupa *hash* SHA256 dari kunci publik milik *miner*

Nama Atribut / Field	Tipe Data	Keterangan
Public_key	String	Kunci public milik *miner* yang pernah terlibat dalam proses *Blockchain*.

2) Basis data di *node Blockchain*

- *Collection Block*
  Collection ini berfungsi sama seperti *collection block* di verifikator namun terdapat beberapa perbedaan struktur. Berfungsi sebagai menyimpan data blok (*chained block,* blok yang diproses dalam *Blockchain*) yang disimpan untuk mempermudah *tracking*

Tabel 3.8 Penjelasan Skema pada *Collection Block* di *node Blockchain*

Nama Atribut / Field	Tipe Data	Keterangan
Name	String	Nama dari *file* blok.
Header_hash	String	*Hash value* dari *header blok*
Prev_block_name	String	Nama dari *file* blok sebelumnya dari blok ini.
Prev_header_hash		*Hash* dari *header* blok sebelumnya.
Event_id	String	Waktu pemilih memasukkan suara.
Total_data	Number	Jumlah data yang ada di blok.

Nama Atribut / Field	Tipe Data	Keterangan
Genesis_block_hash	Array, terdiri dari Object dengan parameter : id & hash	Menyimpan urutan dari *Blockchain* blok yang bersangkutan.

3.4 Perancangan Data (Struktur Blok)

Data hasil suara yang masuk ke dalam sistem jaringan *Blockchain*, disimpan dalam bentuk blok dengan format *binary* dan ekstensi .dat. Karena data dalam bentuk *binary*, diperlukan spesifikasi atau standar untuk dapat membacanya. Dalam sistem ini, dirancang struktur blok yang serupa dengan struktur blok yang digunakan pada sistem *Blockchain* di *Bitcoin* dengan beberapa modifikasi sesuai keperluan sistem ini, yaitu menyimpan data hasil suara sehingga susah untuk diubah. Gambaran spesifikasi rancangan struktur blok yang digunakan pada mekanisme *Blockchain* di sistem ini dijelaskan pada Gambar 3.3.

Blok dari *Event_id* sampai Total data merupakan *header block* sedangkan sisanya merupakan data hasil berupa *share* yang masuk ke dalam sistem. Tiap bagian dari blok memiliki fungsi dan kegunaan masing-masing dalam proses *Blockchain*. Spesifikasi blok mengambil referensi dari rancangan blok pada *Bitcoin* [3] dengan beberapa modifikasi di bagian-bagian tertentu agar sesuai dengan kebutuhan untuk menyimpan data hasil *voting* yang sebelumnya sudah dilakukan proses pemecahan data dengan algoritma *Shamir's Secret Sharing*.

Berikut merupakan penjelasan tiap-tiap bagian blok dan fungsinya di dalam proses *Blockchain*.

Event_id (8 Karakter)	
Panjang blok (4 bytes, Uint32)	
Hash blok sebelumnya (32 bytes, 64 *Hex* karakter)	
Merkle *root* (32 bytes, 64 *Hex* karakter)	
Timestamp (4 bytes, Uint32)	
Difficulty (32 bytes, 64 *Hex* karakter)	
Nounce (4 bytes, Uint32)	
Total Data (4 bytes, Uint32)	
Data hasil suara yang sudah dipecah (*share*)	
Panjang ID *share* (4 bytes)	ID *share*
Panjang Data *share* (4 bytes)	*share*

Gambar 3.3 Struktur Blok yang digunakan

- *Event_Id* adalah data mengenai ID *event* yang sedang berlangsung. Terdiri dari 8 karakter (bisa diubah namun akan mengubah panjang struktur *header* blok nantinya).
- Panjang blok menyimpan data mengenai panjang data blok dalam satuan *byte,* disimpan dalam bentuk *Unsigned integer* 32 *Big Endian.*
- *Hash* blok sebelumnya menyimpan nilai *hash* dari blok sebelumnya di representasikan dalam bentuk

hexadecimal string sepanjang 64 *hex* karakter atau 32 *bytes*.

- *Merkle root* menyimpan nilai *hash* dari semua data yang ada dalam blok dengan metode *merkle* [3] yaitu memasangkan nilai *hash* dari dua data menjadi satu dan dilakukan dengan cara rekursif sehingga semua data menjadi satu nilai *hash*.
- *Timestamp* menyimpan waktu nilai kapan blok tersebut dibuat atau diperbarui (untuk *genesis block* saja). *Timestamp* ini berupa angka nilai *epoch counter* pada komputer dalam satuan detik, disimpan dalam bentuk *Unsigned integer* 32 *Big Endian.*
- *Difficulty* menyimpan nilai *hash* yang ditentukan oleh sistem sebelumnya yang berfungsi untuk mengecek *proof of work* dalam suatu blok [3].
- *Nounce* menyimpan nilai *counter* yang berfungsi menyesuaikan nilai *hash* blok memenuhi *difficulty* yang sudah ditentukan dalam sistem.
- Total data menyimpan jumlah suara yang disimpan dalam sistem berfungsi untuk mengecek data dan panduan iterasi data yang dibutuhkan dalam membaca data pada blok.
- Data *share*, tiap data yang disimpan berbentuk *share*, dan tiap data ini memiliki panjang ID *share* disimpan dalam bentuk *Unsigned Integer 32 Big Endian*, ID *share* dengan panjang sesuai nilai panjang ID *share*, panjang data *share* disimpan dalam bentuk *Unsigned Integer 32 Big Endian*, dan data *share* itu sendiri yang panjangnya sesuai nilai panjang data sebelumnya.

3.5 Perancangan Protokol *Backend*

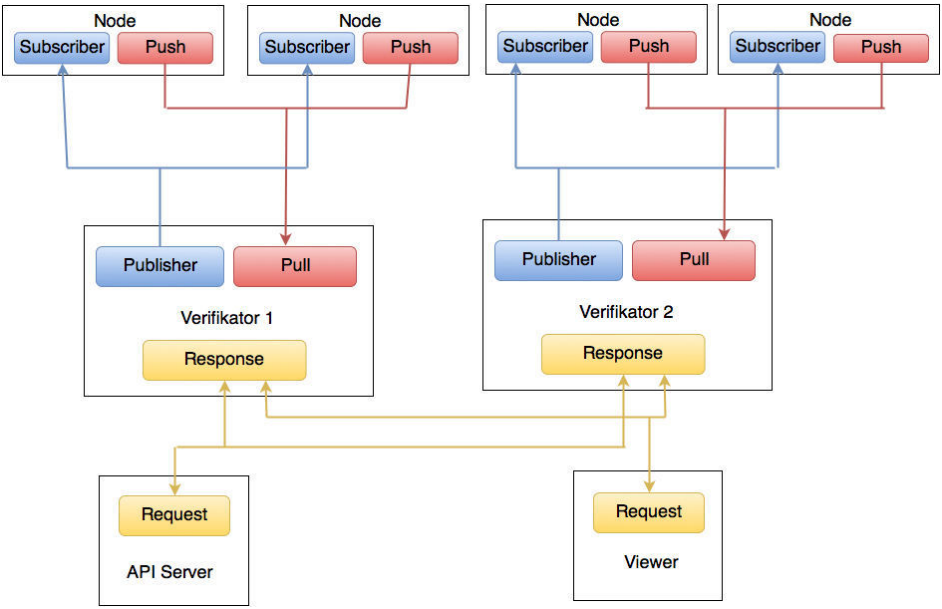
Untuk protokol yang digunakan di *backend*, digunakan *message queue* (*library* yang digunakan adalah ϕMQ) untuk *node*

bisa saling berinteraksi. Di *backend* terdapat tiga tipe komunikasi antar *node*. Berikut penjelasannya.

- Komunikasi antara API *server* dengan verifikator.
  Komunikasi ini dilakukan untuk memasukkan data dari mesin *voting* ke dalam sistem melalui perantara API *server*. Paradigma *message queue* yang digunakan untuk berkomunikasi adalah jenis *Request-Response*, dimana terjadi *blocking* input sampai *node* yang melakukan *request* menghentikan *request* atau *node* yang dimintai *request* menjawab dengan *response*. Hal ini dibutuhkan untuk memastikan data yang dikirim oleh pemilih berhasil masuk ke sistem atau gagal.
- Komunikasi antara verifikator dengan *node* di jaringan *Blockchain*.
  Komunikasi ini dilakukan untuk melakukan proses yang berkaitan dengan *Blockchain* dan mengecek data blok. Paradigma *message queue* yang diterapkan adalah *Publisher-Subscriber* sebagai proses *signaling* dari verifikator ke semua *node* yang terhubung dalam jaringan *Blockchain*. Verifikator sebagai *publisher* dan *node* lainnya sebagai *subscriber*. Selain paradigma *Publisher-Subscriber*, digunakan pula paradigma *Push-Pull* dalam komunikasi ini untuk mengirimkan data hasil kerja dari *node* di *Blockchain* ke verifikator. *Node* berlaku sebagai *push node* dan verifikator sebagai *pull node.*
- Komunikasi antara *Viewer* dengan verifikator.
  Komunikasi ini dilakukan untuk melihat data hasil di dalam sistem. Paradigma *message queue* yang digunakan untuk berkomunikasi adalah jenis *Request-Response*, dimana terjadi *blocking* input sampai *node* yang melakukan *request* menghentikan *request* atau *node* yang dimintai *request* menjawab dengan *response*, sama seperti komunikasi antara API *server* dengan verifikator. Hal ini dibutuhkan untuk memastikan data

yang diterima dari verifikator berhasil didapat atau tidak.

Data yang dikirim dalam *message queue* menggunakan format JSON (*JavaScript Object Notation*). Gambar 3.3 menjelaskan bagaimana antar node terhubung dengan menggunakan *message queue*.



Gambar 3.4 Protokol *Message Queue* di Sistem *Backend*

Di dalam protokol ini, terdapat sejumlah perintah-perintah yang dilakukan untuk melakukan suatu pekerjaan tertentu misal memasukkan data, mengambil data dan sejenisnya. Tabel 3.10 berikut menjelaskan daftar perintah-perintah yang ada dalam sistem yang telah dibuat.

Tabel 3.9 Daftar Perintah di Sistem *Backend*.

Perintah	*Node* yang terlibat	Keterangan
new_data	API *Server* – Verifikator – *Node* di *Blockchain*	Perintah untuk memasukkan data baru ke dalam sistem
request_genesis	*Viewer* - Verifikator	Perintah untuk meminta data mengenai blok *genesis* yang ada pada verifikator
request_data	*Viewer* - Verifikator	Perintah untuk meminta data blok paling baru dan valid yang ada pada sistem
request_info	*Viewer* – Verifikator – *Node* di *Blockchain*	Perintah untuk meminta data informasi mengenai sebuah blok dengan nilai *hash header* sebagai ID
request_block	*Viewer* - Verifikator	Perintah untuk meminta data blok tertentu dengan nilai *hash header* sebagai ID yang nantinya digunakan untuk melakukan rekonstruksi data hasil.
request_download	*Viewer* – Verifikator – *Node* di *Blockchain*	Perintah untuk melakukan download data blok

Perintah	*Node* yang terlibat	Keterangan
		tertentu dengan nilai *hash header* sebagai ID
start_mining	Verifikator – *Node* di *Blockchain*	Perintah untuk melakukan proses *mining* pada suatu *header* blok dari verifikator ke semua *node* yang ada di *Blockchain*.
stop_mining	Verifikator – *Node* di *Blockchain*	Perintah untuk berhenti melakukan proses *mining* dikarenakan sebuah *node* berhasil memecahkan *difficulty* yang ditetapkan
finish_mining	*Node* di *Blockchain* - Verifikator	Perintah yang dikeluarkan oleh *node* kepada verifikator apabila node tersebut berhasil memecahkan *difficulty* yang ditetapkan
verify_block	Verifikator – *Node* di *Blockchain*	Perintah yang diberikan dari verifikator ke *Node* di *Blockchain* untuk verifikasi data blok yang disimpan pada verifikator.

Perintah	*Node* yang terlibat	Keterangan
response_block	*Node* di *Blockchain* - Verifikator	Perintah yang dikeluarkan oleh *node* kepada verifikator apabila blok dengan nilai *hash header* yang diterima disimpan oleh *node*.
response_download	*Node* di *Blockchain* - Verifikator	Perintah yang dikeluarkan apabila sebuah blok terdapat pada *node* yang dimaksud

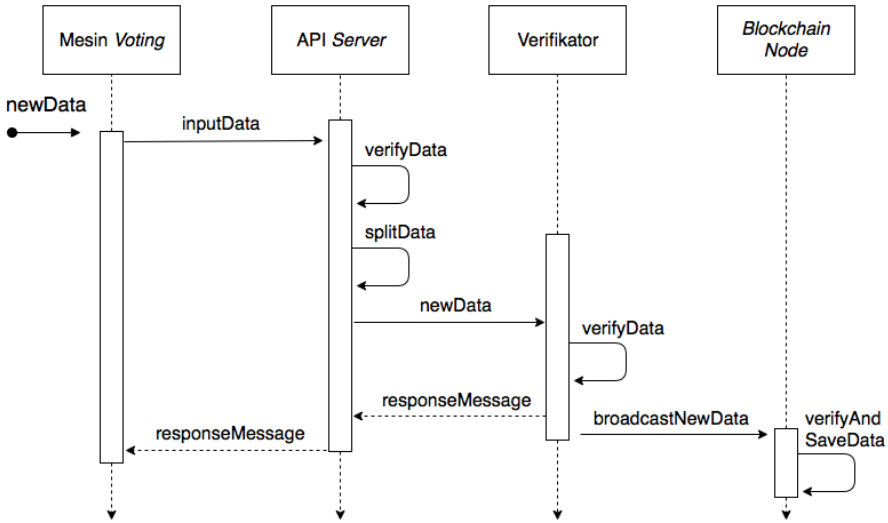
Secara garis besar, terdapat tiga proses yang dapat dijalankan pada sistem yaitu memasukkan data baru, melakukan proses *mining*, mengambil informasi blok atau mengecek validasi blok yang bisa digunakan untuk melakukan rekonstruksi data. Masing-masing proses digambarkan dengan *diagram sequence* pada Gambar 3.5 yang menjelaskan alur proses memasukkan data baru, Gambar 3.6 yang menjelaskan alur proses *mining*, dan Gambar 3.7 yang menjelaskan alur proses mengambil data blok.

Penjelasan dari Gambar 3.5 mengenai alur proses memasukkan data baru adalah sebagai berikut.

- inputData : Mesin *voting* mengirimkan data pilihan dari pemilih ke API *server*. Data dikirim melalui protokol HTTP/HTTPS dengan parameter sebagai berikut.
  1) id : berisi *identifier* dari mesin *voting*.
  2) nik : berisi data NIK dari pemilih.
  3) *session* : kode sesi dari mesin *voting* yang didapat sebelumnya dari proses autentikasi mesin *voting*.
  4) *ballot_sign* : nilai *hash* dari *ballot* yang didapat dan ditandatangani dengan kunci privat milik mesin

voting sebagai bukti bahwa data ini berasal dari mesin *voting* yang terdaftar sebelumnya.

5) *event_id : identifier* dari *event* pemilihan yang terjadi sekarang.
6) *vote_session* : kode sesi yang didapat dari proses autentikasi pemilih.
7) *vote_sign* : berisi nilai *hash* dari *event_id*, *vote_session*, dan kode calon yang dipilih oleh pemilih.
8) *proof_sign* : nilai *hash* dari NIK pemilih yang ditandatangani dengan kunci privat milik mesin *voting*.



Gambar 3.5 Alur Protokol memasukkan Data Baru

- verifyData : API *server* melakukan pengecekan data yang dikirim oleh mesin *voting*. Pengecekan yang dilakukan yaitu mengecek *digital signature* yang

dikirim oleh mesin *voting* (parameter dengan nama *sign*) dan mengambil kode pemilih dari *vote_sign* dengan cara *bruteforce hash* memasukkan kode calon dari data yang ada pada *vote_sign* untuk bisa mendapatkan kode calon yang dipilih. Sedangkan verifikasi di verifikator dan *node Blockchain* adalah dengan mengecek nilai *hash* dari data yang diterima apakah sama dengan *signature* yang didapat.

- splitData : API *server* memecah data kode calon pilihan dengan algoritma *Shamir's Secret Share*, dan memberi ID yang sama pada data hasil pecahan untuk nantinya bisa direkontruksi. Data hasil pecahan ini memiliki format *base64*.
- newData : API *server* mengirim data melalui protokol *message queue* ke verifikator menggunakan perintah *new_data* yang sudah disebutkan sebelumnya. Parameter data yang dikirim adalah sebagai berikut.
  1) cmd : perintah yang dikirim, dalam hal ini adalah perintah *new_data*.
  2) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
  3) *share_id* : ID share yang diberikan secara acak pada data *share* sehingga nantinya bisa dipasangkan untuk rekonstruksi data dengan format *base64*.
  4) *vote_data* : data hasil pecahan berupa *share* yang dimasukkan dalam blok dengan format *base64*.
  5) *voter_count* : jumlah data pemilih yang sudah memberikan suaranya dengan format *integer*.
  6) *sign :* berisi nilai *hash* dari semua parameter yang ada ditambah dengan autentikasi string (*auth_string*) dari verifikator yang telah disepakati sebelumnya kemudian ditandatangani dengan kunci privat API *server*.
- *broadcastNewData* : verifikator melakukan *broadcast* data yang diterima dari API *server* ke semua *node* yang

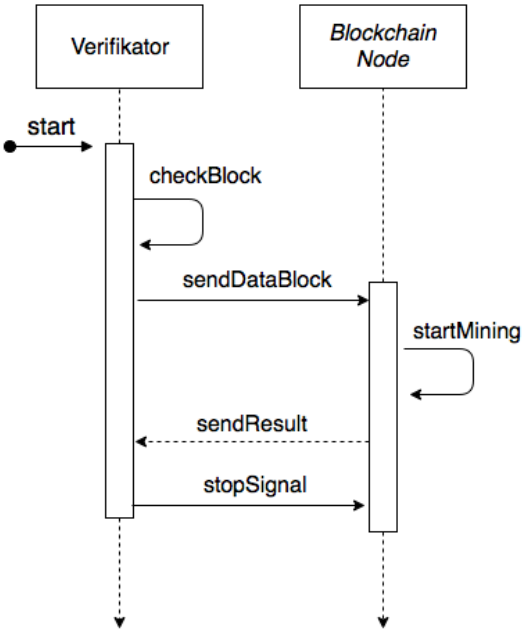
terhubung dalam *Blockchain* melalui protokol *message queue*. Format data yang dikirim yaitu JSON dan terdiri dari atribut sebagai berikut.

1) cmd : perintah yang dikirim, dalam hal ini adalah perintah *new_data*.
2) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
3) *share_id* : ID share yang diberikan secara acak pada data *share* sehingga nantinya bisa dipasangkan untuk rekonstruksi data dengan format *base64*.
4) *vote_data* : data hasil pecahan berupa *share* yang dimasukkan dalam blok dengan format *base64*.
5) *sign :* berisi nilai *hash* dari semua parameter yang dikirim oleh verifikator dan ditandantangani dengan kunci privat verifikator.

- *responseMessage* : balasan kepada perintah yang telah diberikan, balasan ini bisa berupa pesan "ok/success" ataupun pesan "error".

Sedangkan penjelasan dari Gambar 3.6 mengenai alur protokol pada proses *mining* adalah sebagai berikut.

- *checkBlock* : verifikator melakukan cek pada *genesis block* terbaru yang dibuat dan belum dilakukan proses *mining*. Proses pengecekan dilakukan pada *hash header* dan data yang ada blok dengan mencocokkan nilai *hash* dari *merkle root*.
- *sendDataBlock* : verifikator mengirimkan data *header* dari *genesis block* yang digunakan dalam proses *mining*. Data blok yang berupa data *binary* di konversi menjadi format *base64* lalu kemudian data dikirim oleh verifikator. Data yang dikirim ini memiliki format JSON dan dikirim melalui protokol *message* queue. Atribut dari format data yang dikirim adalah sebagai berikut.

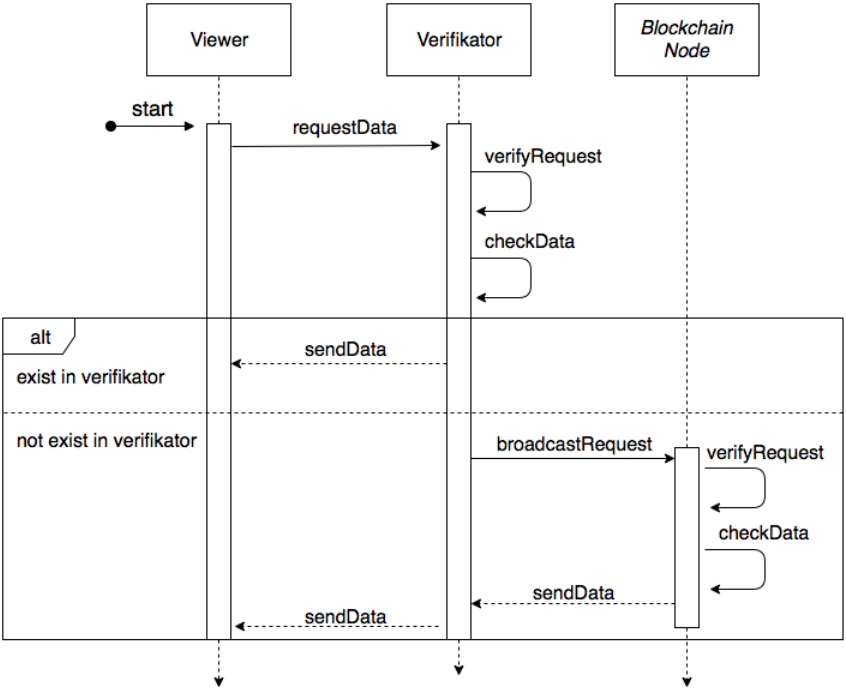


Gambar 3.6 Alur Protokol Proses *Mining*

1) cmd : perintah yang dikirim, dalam hal ini adalah perintah *start_mining*.
2) *name* : nama dari *genesis block* yang digunakan dalam proses *mining* untuk mempermudah pencarian *genesis block* nantinya.
3) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
4) *block* : berisi data *header* dari *block* yang sedang dalam proses *mining*.
5) *genesis_block* : berisi nilai *hash* dari *header genesis block* yang digunakan dalam proses *mining*.
6) *sign :* berisi nilai *hash* dari semua parameter yang dikirim oleh verifikator dan ditandantangani dengan kunci privat verifikator.

- *startMining* : membuat blok baru dengan melakukan proses yang disebut *proof of work* terlebih dahulu dengan melakukan iterasi terhadap nilai *nounce* pada *header* kemudian mencari nilai *hash* dari *header* yang memenuhi *difficulty* yang sebelumnya telah disepakati oleh sistem.
- *sendResult* : mengembalikan hasil dari proses *mining* ke verifikator untuk selanjutnya dilakukan proses lebih lanjut. Data yang dikembalikan memiliki atribut sebagai berikut.
  1) cmd : perintah yang dikirim, dalam hal ini adalah perintah *start_mining*.
  2) *name* : nama dari *genesis block* yang digunakan dalam proses *mining* untuk mempermudah pencarian *genesis block* nantinya.
  3) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
  4) *block* : berisi data *header* dari *block* yang sedang dalam proses *mining*.
  5) *genesis_block* : berisi nilai *hash* dari *header genesis block* yang digunakan dalam proses *mining*.
  6) *public_key* : berisi kunci publik yang dimiliki *node* yang digunakan untuk menandatangani data yang dikirim.
  7) *sign :* berisi nilai *hash* dari semua parameter yang dikirim oleh *node* dan ditandantangani dengan kunci privat milik *node* tersebut.
- *stopSignal* : verifikator melakukan *broadcast* sinyal melalui protokol *message* queue untuk menghentikan aktivitas *mining* yang dilakukan karena salah satu *node* berhasil menyelesaikan proses *mining* dan mengembalikan hasilnya ke verifikator dan telah dilakukan cek bahwa data yang didapat memenuhi *difficulty* yang ditetapkan. Pesan yang dikirim memiliki format JSON dan atribut sebagai berikut.

1) cmd : perintah yang dikirim, dalam hal ini adalah perintah *stop_mining*.
2) *genesis_block* : berisi nilai *hash* dari *header genesis block* yang digunakan dalam proses *mining*.
3) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
4) *sign* : berisi nilai *hash* dari semua parameter yang dikirim oleh verifikator dan ditandantangani dengan kunci privat verifikator.



Gambar 3.7 Alur Protokol Proses *Request* Data

Terakhir yaitu mengenai penjelasan dari Gambar 3.7 mengenai proses *request* data adalah sebagai berikut.

- *requestData* : *Viewer* melakukan *request* untuk meminta data dari sistem melalui protokol *message queue*. Format data yang dikirim berupa JSON dan memiliki atribut sebagai berikut.
  1) cmd : perintah yang dikirim, dalam hal ini perintah yang dilakukan yaitu perintah dengan tipe *request* (misal *request_genesis*, *request_block*, dan semacamnya).
  2) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
  3) *header_hash* : atribut ini dibutuhkan untuk perintah *request_block, request_info,* dan *download_data*. Atribut ini berisi nilai *hash* dari *header* blok yang dicari.
  4) *sign* : berisi nilai *hash* dari semua parameter yang dikirim oleh *viewer* ditambah dengan autentikasi string (*auth_string*) dari *viewer* yang telah disepakati sebelumnya kemudian ditandantangani dengan kunci privat viewer.
- *verifyRequest* : verifikator melakukan pengecekan data yang dikirim oleh *viewer*. Pengecekan yang dilakukan yaitu mengecek *digital signature* yang dikirim oleh *viewer* (parameter dengan nama *sign*).
- *checkData* : verifikator melakukan pengecekan data blok terbaru yang *valid* atau sesuai dengan *request header_hash* yang dimaksud. Apabila data terdapat pada verifikator, verifikator langsung mengirim data blok tersebut. Namun apabila data tersebut tidak ada pada verifikator, dilakukan *broadcastRequest*.
- *broadcastRequest* : apabila data yang dicari tidak ditemukan pada verifikator, verifikator akan meneruskan perintah *request* ke *node* lain yang terhubung pada jaringan *Blockchain*. Format data *request* yang dikirim sama namun dengan *sign* menggunakan kunci privat milik verifikator.

- *sendData* : verifikator atau *node* di *Blockchain* mengirimkan data yang diminta kembali ke *viewer* yang melakukan *request*. Format data yang diterima bisa berupa download link ke *fileserver* yang dijalankan di tiap *node* atau data blok yang dimaksud.

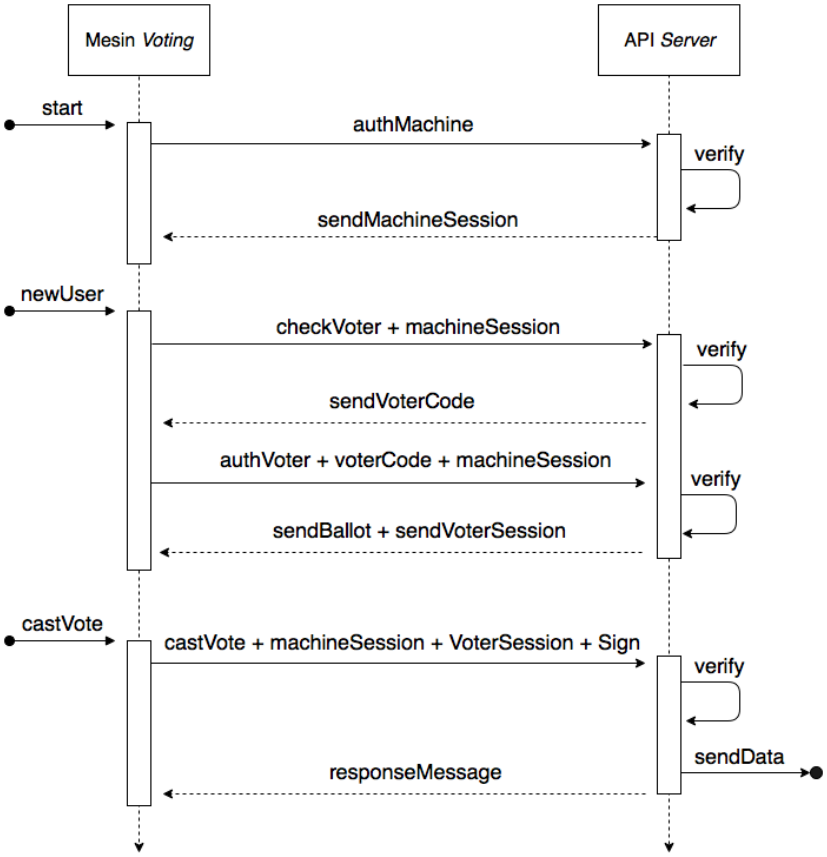
3.6 Perancangan *Frontend* dan API

Untuk memastikan sistem *backend* yang dirancang dapat berjalan dengan benar, dirancang pula mekanisme sistem *frontend* untuk memastikan bahwa sistem *backend* yang telah dirancang berjalan dengan benar. Dalam tugas akhir ini, dibuat contoh mekanisme *frontend* yang terdiri dari dua komponen yaitu mesin *voting* yang berhubungan langsung dengan pemilih serta API *server* yang terhubung dengan mesin *voting* dan memiliki tugas untuk melakukan verifikasi data yang diberikan mesin *voting* juga memberikan informasi mengenai *event* pemilihan.

Komunikasi antara mesin *voting* dengan API *server* secara umum terbagi menjadi tiga tahapan yaitu proses autentikasi mesin *voting*, verifikasi data pemilih, dan memasukkan data pilihan dari pemilih. Pada gambar 3.8 dijelaskan alur komunikasi antara mesin *voting* dengan API *server*.

Penjelasan mengenai tiap tahapan komunikasi yang dijelaskan pada gambar 3.8 adalah sebagai berikut.

- Proses autentikasi mesin *voting* (*authMachine*) : Proses ini dilakukan tiap mesin *voting* pertama kali dijalankan atau ketika koneksi ke *server* mengalami gangguan. Proses ini dimulai dengan penanggung jawab mesin *voting* melakukan konfigurasi terhadap mesin *voting* dengan memasukkan data konfigurasi dan pasangan kunci yang dimiliki oleh mesin *voting* yang sebelumnya telah terdaftar pada API *server*. Kemudian, mesin *voting* mengirimkan data untuk keperluan autentikasi. Data dikirim melalui protokol HTTP/HTTPS POST dan berikut parameter data yang dikirim.



Gambar 3.8 Alur Komunikasi Mesin *Voting* dengan API *Server*

1) *id* : berisi data *identifier* dari mesin *voting* yang bersifat unik di tiap mesin dan telah terdaftar di dalam basis data yang digunakan untuk autentikasi.

2) *password* : berisi data password yang hanya diketahui oleh penanggung jawab mesin *voting* dan sudah tersimpan di dalam basis data terlebih dahulu.
3) *sign* : berisi data *signature* dari data *id*, *password*, dan *auth_string* (string autentikasi yang dimiliki oleh mesin *voting*) untuk memastikan identitas mesin *voting*.

- Proses autentikasi pemilih (*checkVoter* & *authVoter*) : Proses ini dilakukan tiap pemilih ingin memberikan suaranya dalam pemilihan. Pemilih melakukan pengecekan identitas terlebih dahulu ke panitia yang bertugas di TPS yang bersangkutan dengan memberikan kartu identitas (KTP). Penanggung jawab mengecek data pemilih dari sistem yang telah disediakan kemudian akan mendapat kode yang nantinya akan diberi ke pemilih (*checkVoter*). Data yang dikirim saat melakukan proses untuk mendapatkan kode adalah data NIK dan *session* yang dibuat secara acak oleh sistem melalui sistem yang disediakan (berupa *web application*). Setelah pemilih mendapat kode dari penanggung jawab, pemilih memasukkan data NIK dan kode yang diberikan untuk verifikasi di mesin *voting* (*authVoter*). Data dikirim oleh mesin *voting* melalui protokol HTTP/HTTPS POST dan berikut parameter data yang dikirim.
  1) *id* : berisi data *identifier* dari mesin *voting* yang bersifat unik di tiap mesin dan telah terdaftar di dalam basis data yang digunakan untuk autentikasi.
  2) *method* : berisi metode yang dipakai untuk metode autentikasi, hal ini nantinya bisa diubah dengan menggunakan metode autentikasi yang aman.
  3) *session* : berisi kode sesi acak yang diberikan kepada mesin *voting* setelah melakukan proses autentikasi.
  4) *nik* : berisi nomor NIK dari pemilih yang akan menggunakan hak suaranya.

5) *code* : berisi data *key voterCode* yang didapat dari proses pengecekan identitas oleh penanggung jawab di TPS yang bersangkutan (*checkVoter*).
6) *sign* : berisi data *signature* parameter data *id* mesin *voting*, *auth_string* mesin *voting*, NIK, dan *code* untuk memastikan data yang dikirim benar.

Dari proses autentikasi ini, mesin *voting* akan mendapatkan data *ballot* dan *vote_session*. Data *ballot* berlaku sebagai surat suara dalam pemilihan sedangkan *vote_session* adalah sesi yang diberikan kepada pemilih sebagai bukti autentikasi telah diverifikasi. Data *ballot* yang dibuat dalam sistem ini memiliki format JSON dan atribut seperti yang dijelaskan pada tabel 3.3.

- Proses memasukkan suara (*castVote*) : Proses ini dilakukan tiap pemilih yang memasukkan suara ke dalam sistem. Data pilihan suatu calon diwakilkan dengan ID calon yang terdapat pada *ballot*. Untuk memastikan bahwa pemilih melihat data *ballot* yang asli, nantinya data *ballot* yang ditampilkan dibuat *signature* nya dan dikirim lagi ke API *server* untuk di cek. Data *vote* ini dikirim oleh mesin *voting* melalui protokol HTTP/HTTPS POST dan berikut parameter data yang dikirim.
  1) *id* : berisi data *identifier* dari mesin *voting* yang bersifat unik di tiap mesin dan telah terdaftar di dalam basis data yang digunakan untuk autentikasi.
  2) *nik* : berisi nomor NIK dari pemilih yang akan menggunakan hak suaranya.
  3) *method* : berisi metode yang dipakai untuk metode autentikasi, hal ini nantinya bisa diubah dengan menggunakan metode autentikasi yang aman.
  4) *session* : berisi kode sesi acak yang diberikan kepada mesin *voting* setelah melakukan proses autentikasi.
  5) *ballot_sign* : berisi *signature* yang dihasilkan oleh kunci privat mesin *voting* dari ballot yang diterima

sebagai bukti bahwa data *ballot* yang dilihat oleh pemilih sama dengan *ballot* yang diberikan oleh API *server*.

6) *event_id* : *identifier* dari *event* pemilihan yang terjadi sekarang.
7) *vote_session* : berisi kode sesi yang diberikan dari proses autentikasi pemilih sebelumnya.
8) *vote_sign* : berisi *signature* dari data *event_id*, *vote_session*, dan ID calon pilihan yang nantinya akan dicek oleh sistem. Jadi data ID calon pilihan yang dipilih *user* tidak dikirim dengan format *plain text*, namun dengan teknik mencocokkan *signature* yang diberikan ini kemudian sistem akan melakukan *brute_force* ID calon pilihan sampai menemukan hasil. Apabila tidak ada ID calon yang cocok, maka data tersebut dianggap tidak *valid*.
9) *proof_sign* : berisi *signature* dari data NIK pemilih yang dihasilkan dari kunci privat mesin *voting*.

Setelah proses ini selesai, mesin *voting* akan menerima pesan dari API *server* apakah pesan yang dikirim berhasil diterima atau tidak. Pada Tabel 3.10 dijelaskan mengenai daftar REST API yang ada pada API *server*.

Tabel 3.10 Daftar REST API di API *Server*

REST API	Parameter Input	Response Data	Keterangan
/authenticate	• *id* • *password* • *sign*	• *session*	API untuk proses autentikasi mesin *voting* (*authMachine*)
/verify	• *id* • *method* • *session*	• *ballot* • *vote_session*	API untuk proses autentikasi

REST API	Parameter Input	Response Data	Keterangan
	• *nik* • *code* • *sign*	• *code*	pemilih (*authVoter*)
/cast-vote	• *id* • *nik* • *method* • *session* • *ballot_sign* • *event_id* • *vote_session* • *vote_sign* • *proof_sign*	• ok / error	API untuk proses autentikasi data pilihan yang masuk ke sistem

3.7 Perancangan Antarmuka

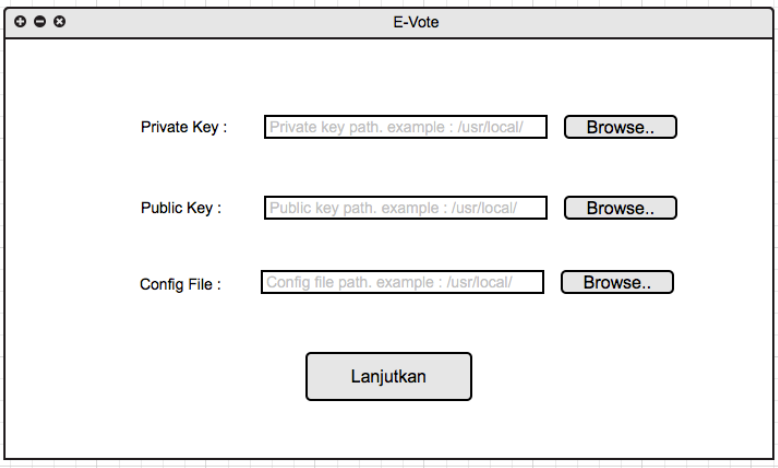
Antarmuka tampilan yang dirancang terdiri dari antarmuka mesin *voting* dan antarmuka yang menghubungkan dengan sistem *Backend* untuk mempermudah interaksi antara *administrator* dengan sistem.

3.7.1.1 Perancangan Antarmuka Mesin *Voting*

Untuk perancangan tampilan antarmuka di mesin *voting*, terdapat empat tampilan yaitu jendela konfigurasi data mesin *voting*, jendela konfigurasi koneksi, jendela verifikasi pemilih, dan jendela utama.

3.7.1.1.1 Jendela Konfigurasi Data

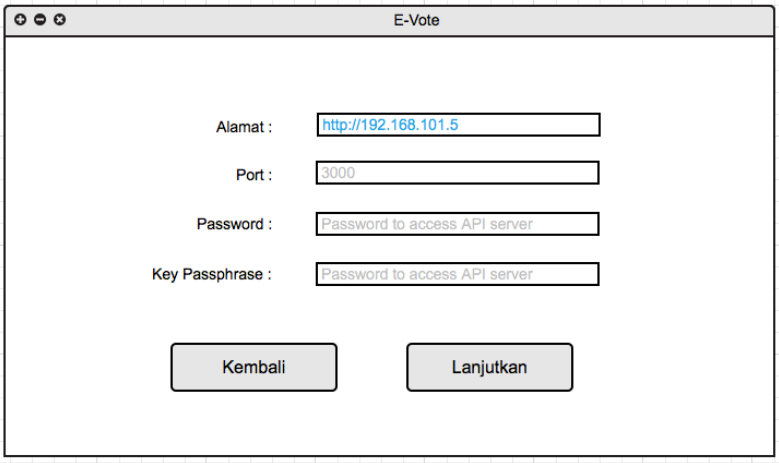
Jendela ini digunakan untuk memilih data konfigurasi mesin *voting* yang digunakan. Terdiri dari data pasangan kunci publik & kunci privat, dan *file* konfigurasi yang telah disediakan. Gambar 3.9 merupakan gambaran secara umum mengenai bentuk tampilan jendela konfigurasi data yang akan dirancang.



Gambar 3.9 Rancangan Jendela Konfigurasi Data

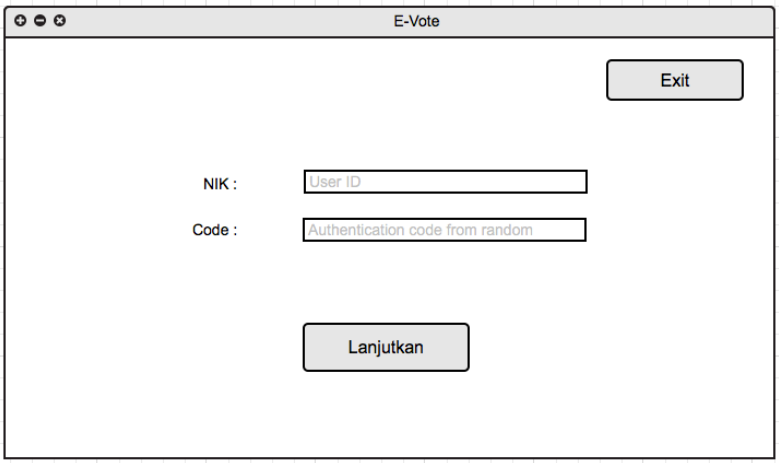
3.7.1.1.2 Jendela Konfigurasi Koneksi

Jendela ini digunakan untuk mengatur konfigurasi koneksi mesin *voting* ke API *server* dengan memasukkan sejumlah parameter untuk dapat mengakses API *server*. *Administrator* mengisi parameter di halaman ini sebelum mesin *voting* digunakan. Gambar 3.10 merupakan gambaran secara umum mengenai bentuk tampilan jendela konfigurasi koneksi yang akan dirancang.



Gambar 3.10 Rancangan Jendela Konfigurasi Koneksi

3.7.1.1.3 Jendela Verifikasi Pemilih

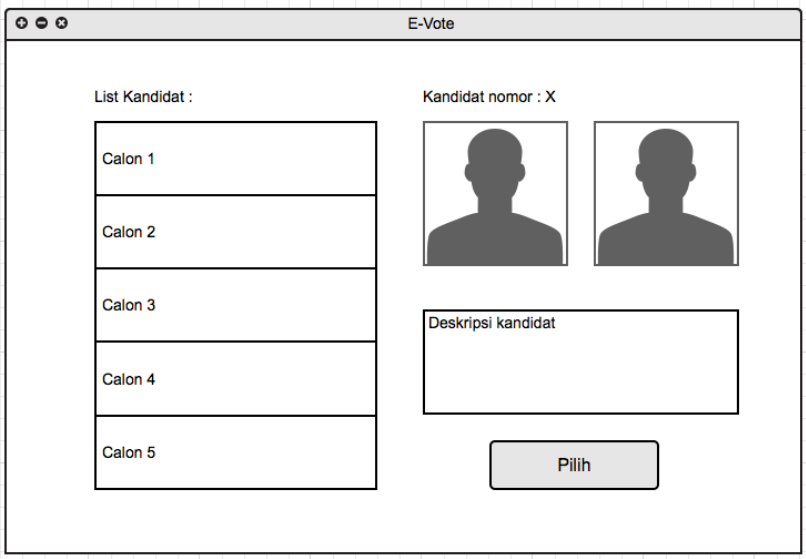


Gambar 3.11 Rancangan Jendela Verifikasi Pemilih

Jendela ini digunakan untuk melakukan autentikasi data pemilih pada sistem. Gambar 3.11 merupakan gambaran secara umum mengenai bentuk tampilan jendela verifikasi pemilih yang akan dirancang.

3.7.1.1.4 Jendela Utama

Jendela ini digunakan oleh pemilih untuk memilih calon yang terdaftar pada *ballot* yang disediakan oleh API *server*. *Ballot* yang sebelumnya diterima oleh mesin *voting* dalam format JSON di interpretasi ke dalam tampilan seperti pada Gambar 3.12 yang merupakan gambaran secara umum mengenai bentuk tampilan jendela utama yang akan dirancang. Tampilan jendela utama ini bergantung pada format *ballot* yang disediakan oleh API *server* sehingga apabila format *ballot* berubah, tampilan jendela utama pun harus menyesuaikan dengan *ballot* yang disediakan.



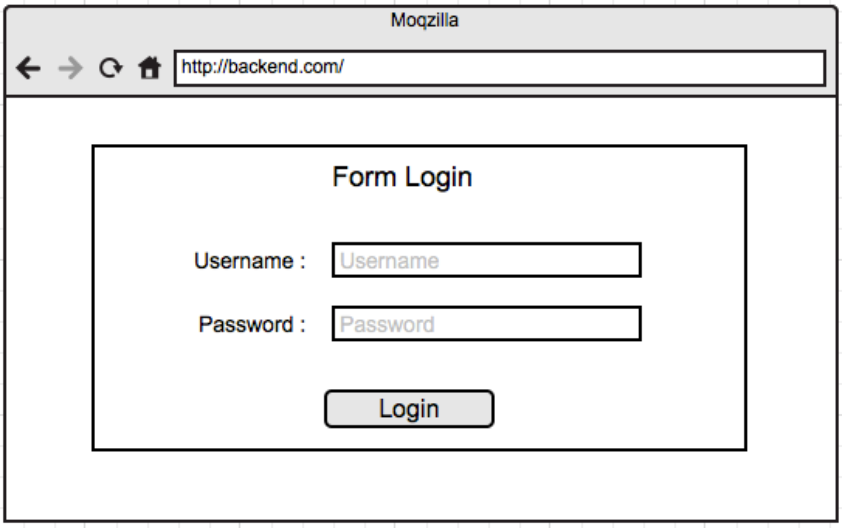
Gambar 3.12 Rancangan Jendela Utama

3.7.1.2 Perancangan Antarmuka Sistem *Backend*

Untuk perancangan tampilan antarmuka di sistem *backend* digunakan *web application* agar mudah diakses oleh *administrator* di semua TPS yang menjalankan pemilihan umum. Secara umum, sistem *backend* memiliki tiga fitur utama yaitu melihat data yang ada di *shared database*, memberikan autorisasi kepada pemilih untuk dapat memberikan suara dengan mengambil kode yang dibuat oleh sistem, dan mengambil data mengenai hasil suara atau blok data yang terdapat pada sistem *Backend*. Berikut adalah rancangan dari tampilan sistem *backend* yang akan dibuat.

3.7.1.2.1 Halaman *Login*

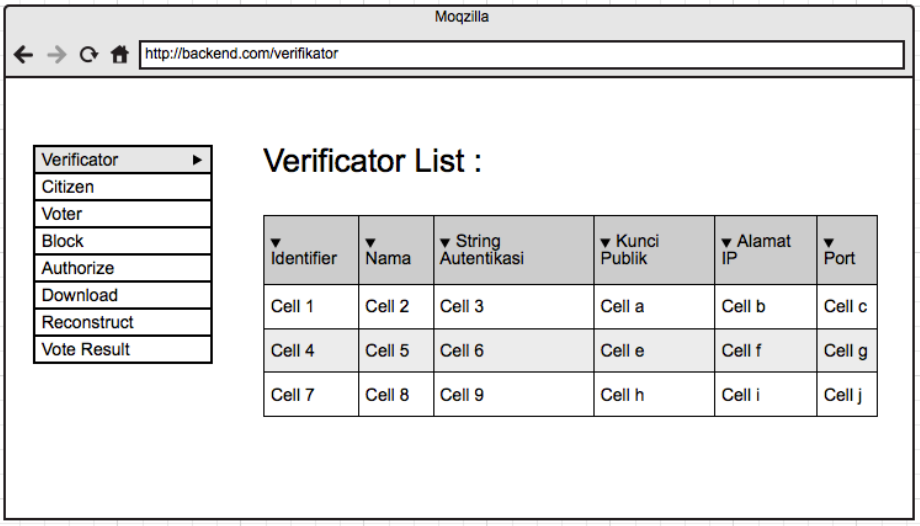
Halaman ini merupakan halaman awal sistem *backend* untuk melakukan autentikasi kepada *administrator* yang berhak. Gambar 3.13 menggambarkan halaman ini secara umum.



Gambar 3.13 Rancangan Halaman *Login*

3.7.1.2.2 Halaman *List* Verifikator

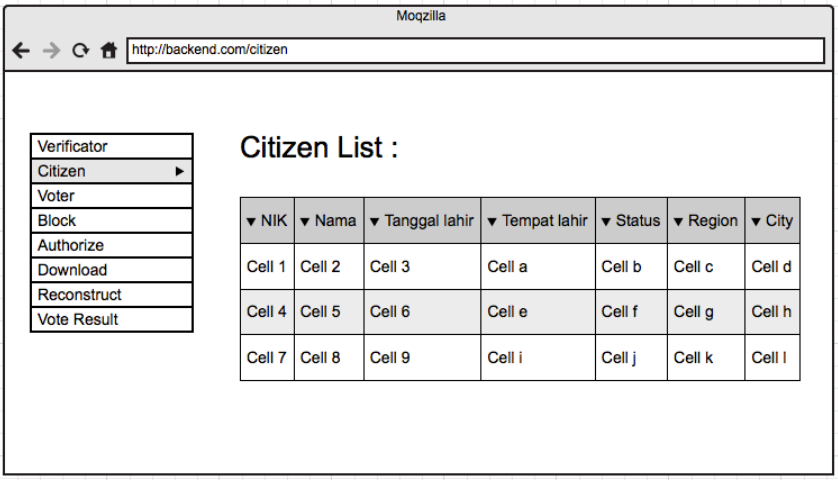
Halaman ini merupakan halaman yang menampilkan data verifikator yang terhubung dalam sistem *backend*. Halaman ini hanya untuk keperluan monitoring saja. Gambar 3.14 menggambarkan halaman ini secara umum.



Gambar 3.14 Rancangan Halaman *List* Verifikator

3.7.1.2.3 Halaman *List Citizen*

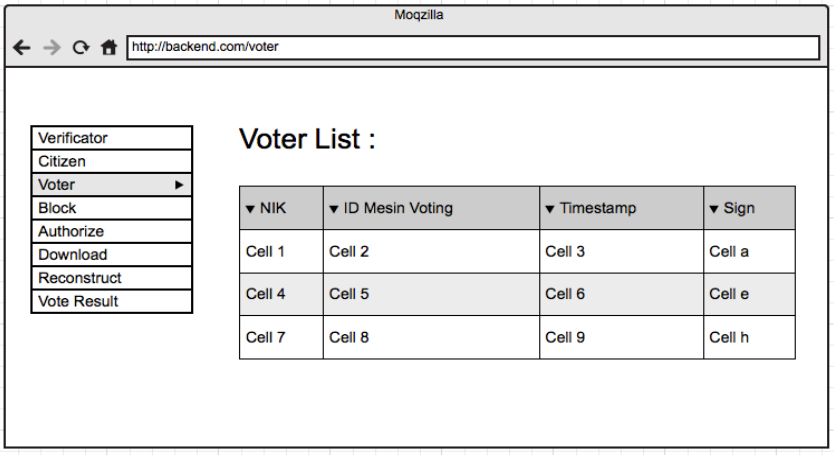
Halaman ini merupakan halaman yang menampilkan data *citizen* atau warga negara yang terdaftar sebagai pemilh tetap dalam sistem *backend*. Daftar Pemilih Tetap (DPT) harus sudah terdaftar sebelum pemilihan berlangsung dan data tersebut sudah ada di dalam *shared database* dalam sistem ini. Halaman ini hanya untuk keperluan monitoring saja. Gambar 3.15 menggambarkan halaman ini secara umum.



Gambar 3.15 Rancangan Halaman *List Citizen*

3.7.1.2.4 Halaman *List Voter*

Halaman ini merupakan halaman yang menampilkan data pemilih yang telah memberikan hak suara.

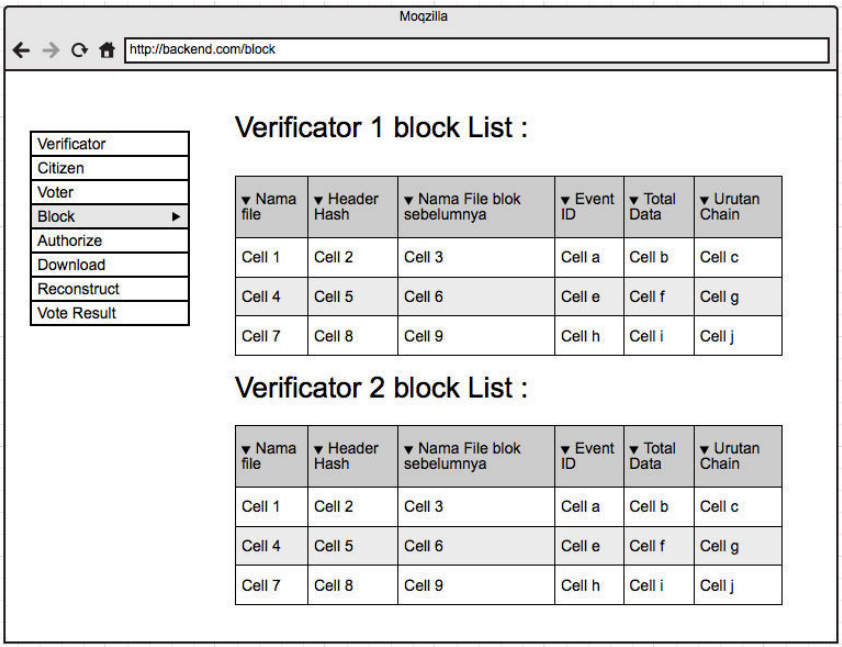


Gambar 3.16 Rancangan Halaman *List Voter*

Halaman ini digunakan *administrator* untuk mengecek apakah data pemilih berhasil masuk atau tidak.

3.7.1.2.5 Halaman *List Block*

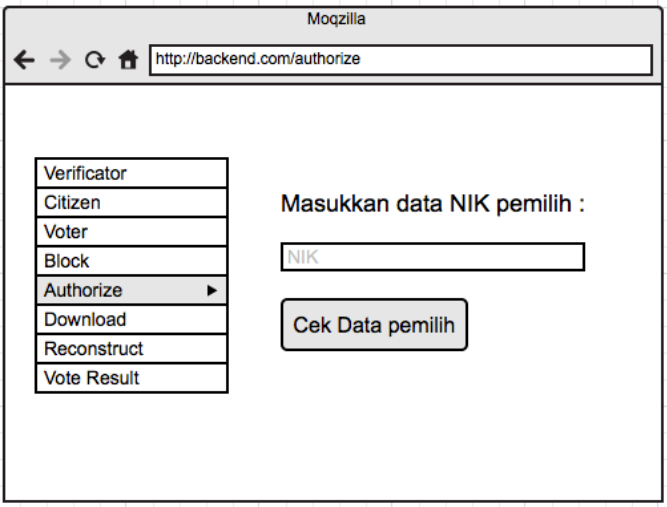
Halaman ini merupakan halaman yang menampilkan data blok yang terdapat di sistem *backend*.



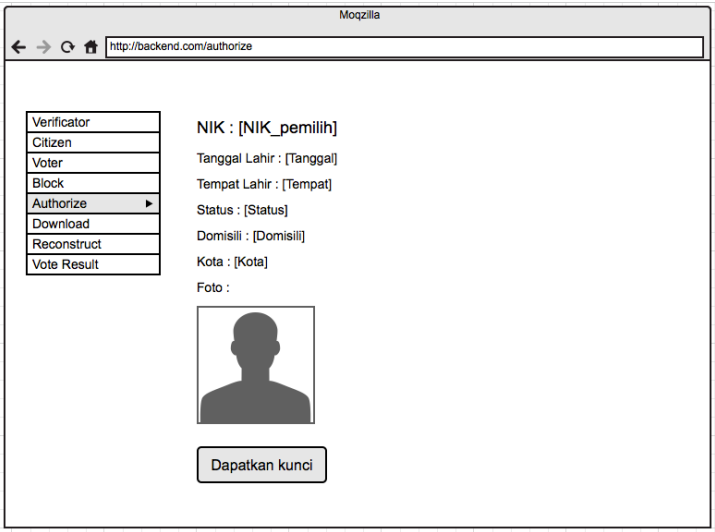
Gambar 3.17 Rancangan Halaman *List Block*

Pada Gambar 3.17 digambarkan secara umum mengenai tampilan halaman ini. Tiap verifikator memiliki *list* data blok yang berbeda. Blok yang ditampilkan disini adalah data *genesis_block* yang terdapat pada tiap verifikator yaitu blok pertama yang dibuat dan merupakan blok awal dari sebuah proses *Blockchain*.

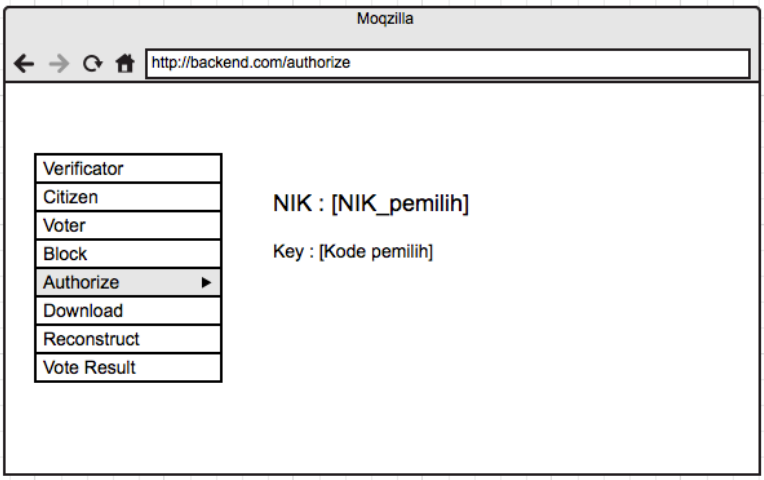
3.7.1.2.6 Halaman *Authorize*



Gambar 3.18 Rancangan Halaman *Authorize* 1



Gambar 3.19 Rancangan Halaman *Authorize* 2

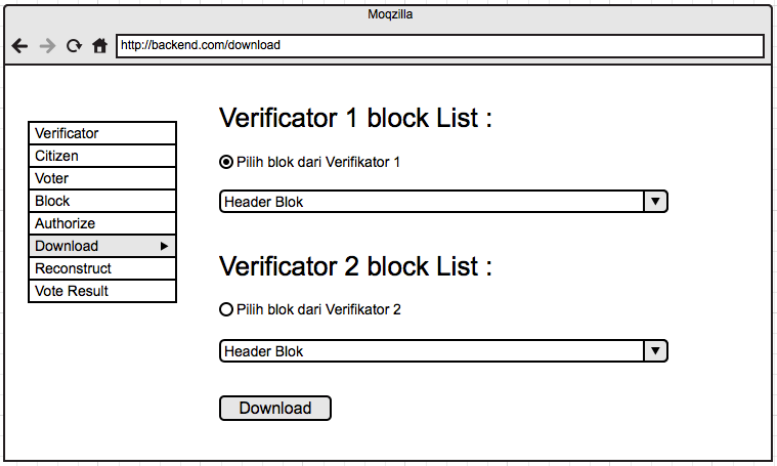


Gambar 3.20 Rancangan Halaman *Authorize* 3

Halaman ini berfungsi untuk memberikan autorisasi kepada pemilih untuk memberikan hak suaranya dengan melakukan pengecekan identitas terlebih dahulu oleh *administrator* yang bertugas. Terdapat tiga langkah di dalam proses autorisasi ini yaitu memasukkan NIK, pengecekan data, dan pembuatan kode. Gambar 3.18, Gambar 3.19 dan Gambar 3.20 menggambarkan rancangan halaman ini secara urut.

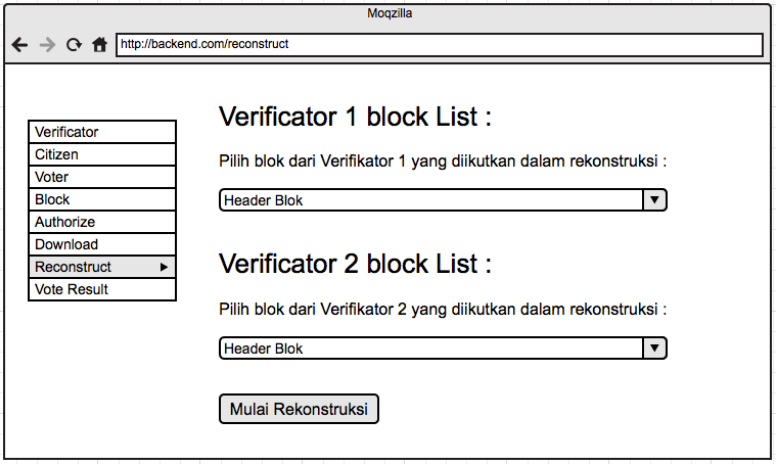
3.7.1.2.7 Halaman *Download*

Halaman ini berfungsi untuk memberikan opsi kepada *administrator* untuk mengunduh data blok yang diinginkan (hal ini merupakan fitur opsional) yang nantinya bisa direkonstruksi secara lokal apabila data blok yang ada di sistem memiliki kapasitas yang amat besar dan tidak memungkinkan untuk melakukan rekonstruksi langsung lewat antarmuka yang disediakan. Gambar 3.21 menggambarkan tampilan halaman ini secara umum.



Gambar 3.21 Rancangan Halaman *Download*

3.7.1.2.8 Halaman *Reconstruct*



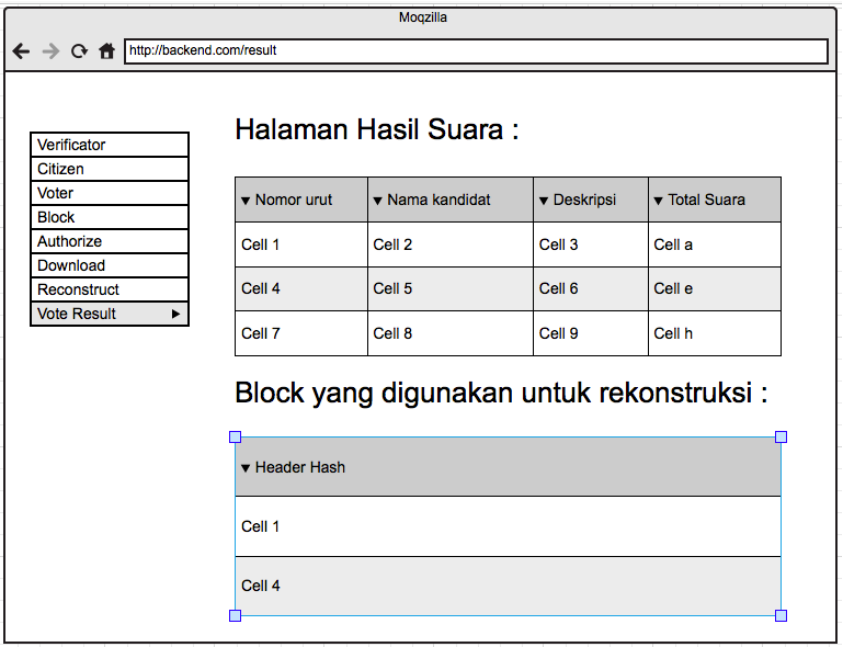
Gambar 3.22 Rancangan Halaman *Reconstruct*

Halaman ini berfungsi untuk memberikan opsi kepada *administrator* untuk memilih data blok yang diinginkan untuk

direkonstruksi dan data hasil suara ditampilkan di tampilan sistem *backend* pada halaman *vote result*. Gambar 3.22 menggambarkan tampilan halaman ini secara umum.

3.7.1.2.9 Halaman *Vote Result*

Halaman ini berfungsi untuk menampilkan data hasil suara yang berhasil direkonstruksi dari sejumlah blok di verifikator yang berbeda baik dengan memilih blok yang diinginkan maupun dengan otomatis mencari blok dengan data terbaru dan status blok yang valid setelah dilakukan beberapa tes untuk menguji validitas blok. Gambar 3.23 menggambarkan tampilan halaman ini secara umum.



Gambar 3.23 Rancangan Halaman *Vote Result*

BAB IV
IMPLEMENTASI

Bab ini berisi penjelasan mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun sistem yang telah dirancang sebelumnya.

4.1 Lingkungan Implementasi

Implementasi sistem *E-Vote* yang dibangun terdiri dari dua macam, lingkungan sistem *frontend*, dan lingkungan sistem *backend*. Spesifikasi perangkat keras dan perangkat lunak di sistem *frontend* ditunjukkan pada Tabel 4.1 sedangkan untuk *backend,* seperti ditunjukkan pada Tabel 4.2.

Tabel 4.1 Lingkungan Implementasi Sistem *Frontend*

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	2.7 GHz Intel Core i7
	Memori	4 GB 1333 MHz DDR3
Perangkat Lunak	Sistem Operasi	OS X El Capitan versi 10.11.2
	Perangkat Pengembang	Qt Framework

Tabel 4.2 Lingkungan Implementasi Sistem *Backend*

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Harddisk	20 GB
	Memori	512 MB & Swap 1 GB

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Lunak	Sistem Operasi	Ubuntu 14.04
	Perangkat Pengembang	ExpressJS, MongoDB

Sistem *backend* menggunakan *Virtual Private Server* (VPS) yang dikelola oleh *Digital Ocean* dengan lokasi fisik *server* berada di Singapura. *Virtual Private Server* ini terhubung satu dengan yang lain dengan menggunakan jaringan VPN. Satu *Virtual Private Server* bertindak sebagai VPN *server* dalam sistem ini dan lainnya bertindak sebagai *node* yang terhubung ke jaringan VPN.

4.2 Implementasi

Pada sub bab implementasi ini menjelaskan mengenai pembangunan sistem *E-Vote* secara detail dan menampilkan kode sumber yang digunakan.

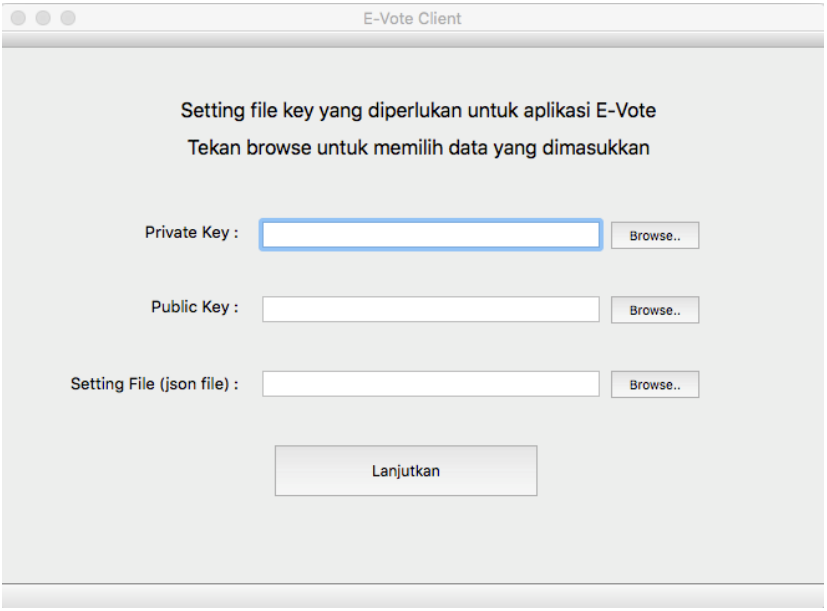
4.2.1 Implementasi Sistem *Frontend*

Terdapat dua komponen yang membangun kinerja dari sistem *frontend* yang dirancang pada sistem ini yaitu mesin *voting* dan API *server*.

4.2.1.1 Implementasi Antarmuka Mesin *Voting*

4.2.1.1.1 Jendela Konfigurasi Data

Pada Gambar 4.1 menunjukkan hasil implementasi dari aplikasi yang telah dibangun. Jendela ini merupakan jendela awal ketika aplikasi ini dijalankan di mesin *voting*.



Gambar 4.1 Implementasi Jendela Konfigurasi Data

4.2.1.1.1.1 Implementasi Tombol *Browse*

Tombol *Browse* ini berfungsi untuk memanggil jendela *file system* yang berguna untuk memilih data yang diinginkan dari komputer untuk diambil *path directory* nya dan didaftarkan pada aplikasi ini. Ini untuk memudahkan *administrator* dan mengurangi *human* error ketika memilih data konfigurasi yang diperlukan untuk mesin *voting*.

Untuk tombol *browse* di input *private key* dan *public key*, format data yang diperlukan yaitu format data .pem dan .der. Sedangkan untuk *setting file* diperlukan format data .json. Apabila format tidak sesuai, aplikasi akan menolak data tersebut. Implementasi kode bisa dilihat pada lampiran A.1.

```
SET desiredFormat
READ path
file ← getFile(path)
IF
    RETURN file
ELSE
    RETURN NULL
ENDIF
```

Pseudocode 4.1 Tombol *Browse*

4.2.1.1.1.2 Implementasi Tombol Lanjutkan

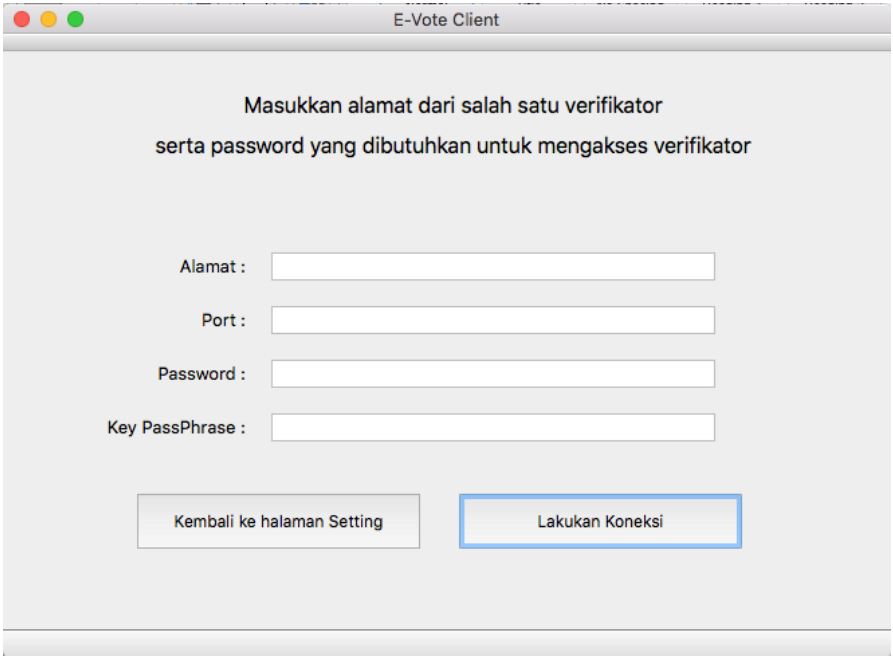
Tombol ini hanya untuk melanjutkan ke menu sebelumnya namun sebelum itu dilakukan pengecekan data yang dimasukkan. Implementasi kode bisa dilihat pada lampiran A.2.

```
READ privateKeyPath
READ publicKeyPath
READ settingFilePath
privateKey ← getPrivateKey(privateKeyPath)
publicKey ← getPublicKey(publicKeyPath)
settingFile ← getSettingFile(settingFilePath)
IFprivateKey and publicKey and settingFile ≠ NULL
    RETURN true
ELSE
    RETURNfalse
ENDIF
```

Pseudocode 4.2 Tombol Lanjutkan

4.2.1.1.2 Jendela Konfigurasi Koneksi

Pada Gambar 4.2 menunjukkan hasil implementasi dari jendela konfigurasi koneksi.



Gambar 4.2 Implementasi Jendela Konfigurasi Koneksi

Jendela ini merupakan antarmuka untuk memasukkan konfigurasi koneksi ke API *server*.

4.2.1.1.2.1 Implementasi Tombol Kembali

Tombol ini digunakan untuk kembali ke jendela konfigurasi data sebelumnya. Implementasi kode bisa dilihat pada lampiran A.3. Tombol kembali ini hanya terdapat pada pada halaman konfigurasi koneksi saja untuk bisa berpindah ke halaman konfigurasi data.

4.2.1.1.2.2 Implementasi Tombol Koneksi

Tombol ini berfungsi untuk menjalankan fungsi koneksi ke API *server* dengan data yang diberikan pada halaman konfigurasi. Implementasi kode bisa dilihat pada lampiran A.4.

```
READ settingFile
id ← settingFile.id
authString ← settingFile.authString
READ privateKey
READ address
READ port
READ password
READ passphrase
data ← concat(id, password, authString)
signature ← sign(data, privateKey, passphrase)
SET parameters(id, password, signature)
SET url(address, port, parameters)
reply ← connect(url)
IF reply = error
    RETURN error
ELSE
    IF reply HAS session AND serverKey
        SET session
        SET serverKey
        RETURN true
    ELSE
        RETURN false
    ENDIF
ENDIF
```

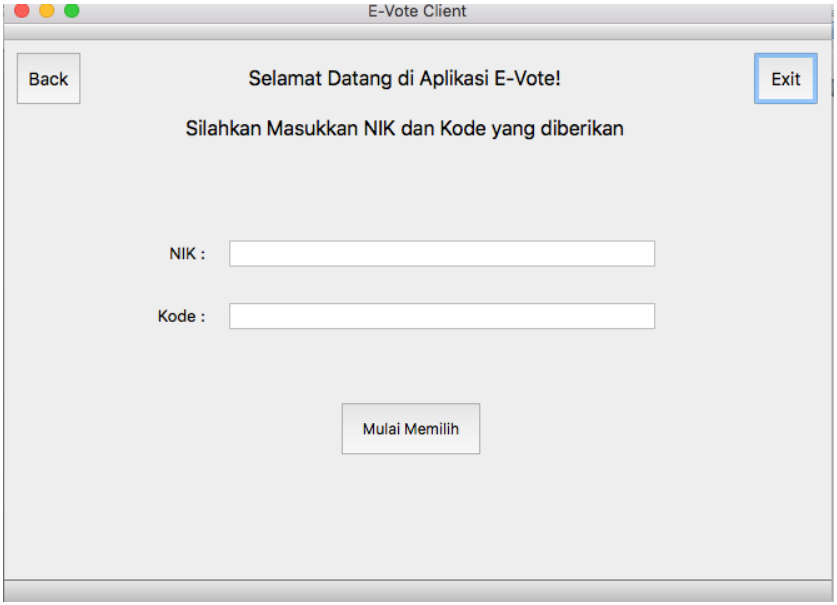
Pseudocode 4.3 Tombol Koneksi

Jendela selanjutnya dari jendela konfigurasi koneksi adalah jendela verifikasi pemilih sedangkan jendela sebelumnya adalah jendela konfigurasi data.

4.2.1.1.3 Jendela Verifikasi Pemilih

Pada Gambar 4.3 ditunjukkan hasil implementasi dari tampilan jendela verifikasi pemilih. Jendela ini berfungsi sebagai antarmuka untuk memasukkan data pemilih yang nantinya akan di verifikasi oleh API *server*. Disini terdapat dua buah input teks yaitu NIK dan kode. Kode yang harus dimasukkan disini adalah kode yang didapat dari proses autentikasi pemilih yang dilakukan sebelum pemilih memberikan suara di mesin *voting*, yaitu kode 12 karakter acak yang dibuat oleh sistem dengan sepengetahuan dari *administrator* sistem.

Apabila pemilih sudah memberikan suara sebelumnya atau tidak terdaftar dalam DPT, maka aplikasi akan menolak pemilih untuk masuk ke jendela utama untuk memberikan suara.



Gambar 4.3 Implementasi Jendela Verifikasi Pemilih

4.2.1.1.3.1 Implementasi Tombol Mulai

Tombol ini berfungsi untuk memulai verifikasi data pemilih dengan mengirim data yang dimasukkan ke API *server*. Implementasi kode bisa dilihat pada lampiran A.5.

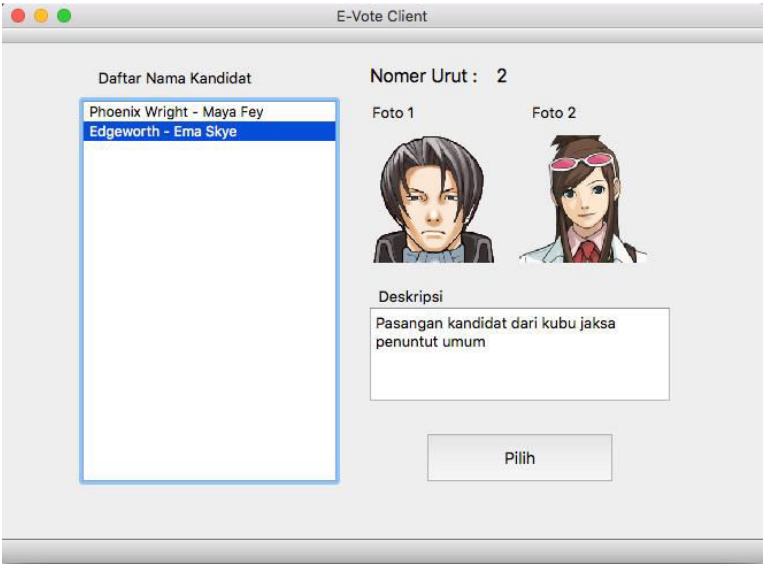
```
READ settingFile
id ← settingFile.id
authString ← settingFile.authString
READ privateKey
READ address
READ port
READ password
READ passphrase
READ nik
READ code
READ session
SET method
data ← concat(id, authString, nik, code)
signature ← sign(data, privateKey, passphrase)
SET parameters(id, method, session)
SET url(address, port, parameters)
reply ← connect(url)
IF reply = error
    RETURN error
ELSE
    IF reply HAS session AND serverKey
        SET session
        SET serverKey
        RETURN true
    ELSE
        RETURN false
    ENDIF
ENDIF
```

Pseudocode 4.4 Tombol Mulai Memilih

4.2.1.1.4 Jendela Utama

Pada Gambar 4.4 menunjukkan hasil implementasi dari jendela utama. Jendela ini merupakan antarmuka untuk pemilih memilih kandidat yang dipilihnya. Di jendela ini terdapat *list* nama kandidat di sebelah kiri, dan data mengenai kandidat di sebelah kanan. Pemilih awalnya memilih dengan cara klik *list* nama kandidat yang ada di sebelah kiri, kemudian data kandidat akan muncul sesuai dengan nama yang dipilih oleh pemilih.

Data *ballot* yang didapat dari API *server* diolah oleh aplikasi mesin *voting* sehingga menghasilkan tampilan seperti itu. Tampilan jendela utama harus menyesuaikan dengan data *ballot* yang disediakan sehingga apabila format *ballot* yang disediakan oleh API *server* berubah, maka untuk dapat mengolah data *ballot* tersebut, mesin *voting* harus menyesuaikan tampilan sesuai format yang disediakan.



Gambar 4.4 Implementasi Jendela Utama

4.2.1.1.4.1 Implementasi *List* Kandidat

List ini berisi daftar kandidat yang terdaftar pada *event* pemilihan calon saat ini dan pemilih bisa memilih kandidat mana yang dipilih dengan melakukan klik nama kandidat. Otomatis informasi mengenai kandidat akan berubah sesuai pilihan. Kode implementasi dapat dilihat pada lampiran A.6.

```
READ ballot
IF ballot.length > 0
    ui ← ballot[0].data
    voteCode ← ballot[0].candidate_id
    render(ui)
ELSE
    RETURN error
ENDIF

IF ui = clicked
    index ← getIndex()
    ui ← ballot[index].data
    voteCode ← ballot[index].candidate_id
    render(ui)
ENDIF
```

Pseudocode 4.5 *List* Kandidat

4.2.1.1.4.2 Implementasi Tombol Pilih

Tombol ini berfungsi untuk mengirimkan data pemilih ke API *server* yang kemudian akan dimasukkan ke dalam sistem. Kode implementasi bisa dilihat pada lampiran A.7. Tombol ini berfungsi mengirimkan data kandidat pilihan yang dipilih oleh pemilih ke API *server*. Nilai *hash* dari ballot yang digunakan untuk memilih pun juga dikirim ke API *server* sebagai bukti bahwa *ballot* yang diterima merupakan *ballot* yang benar.

```
READ settingFile
id ← settingFile.id
authString ← settingFile.authString
READ privateKey
READ address
READ port
READ password
READ code
READ passphrase
READ nik
READ session
READ serverKey
READ voterSession
READ voteCode
eventID ← ballot.event_id
dataVote ← concat(eventID, voterSession, voteCode)
voteSign ← sign(dataVote, privateKey, passphrase)
ballotSign ← sign(ballot, privateKey, passphrase)
proofSign ← sign(nik, privateKey, passphrase)
SET parameters(id, nik, method, session, ballotSign,
        eventID, voterSession, voteSign, proofSign)
SET url(address, port, parameters)
reply ← connect(url)
IF reply = error
    RETURN error
ELSE
    IF reply HAS success
        RETURN true
    ELSE
        RETURN false
    ENDIF
ENDIF
```

Pseudocode 4.6 Tombol Pilih

4.2.1.2 Implementasi API *Server*

4.2.1.2.1 API Autentikasi Mesin *Voting*

REST API berada di URL /authenticate dan berfungsi untuk menerima dan verifikasi data autentikasi dari mesin *voting*. Kode implementasi dapat dilihat pada lampiran A.8.

```
READ id
READ password
READ sign
IF id IN database
    match ← compare(hash, password)
    IF match
        valid ← verify(sign)
        IF valid
            session ← base64Encode(randomBytes)
            serverKey ← readFile(publicKey)
            RETURN (session, serverKey)
        ELSE
            RETURN error
        ENDIF
    ELSE
        RETURN error
    ENDIF
ELSE
    RETURN error
ENDIF
```

Pseudocode 4.7 API Autentikasi Mesin Voting

4.2.1.2.2 API Verifikasi Pemilih

REST API ini berada pada URL /verify dan berfungsi untuk menerima dan verifikasi data pemilih yang dimasukkan lewat mesin *voting*. Dari data ini diketahui apakah pemilh

terdaftar sebagai pemilih yang *valid* atau tidak. Kode implementasi dapat dilihat pada lampiran A.9.

```
READ id
READ session
READ nik
READ code
READ sign
IF id IN database
    match ← compare(hash, session)
    IF match
        IF nik IN database
            authorized ← authenticate(nik, code)
            IF authorized
                done ← check(nik)
                IF done
                    RETURN error
                ELSE
                    valid ← verify(sign)
                    IF valid
                        DO Process
                    ELSE
                        RETURN error
                    ENDIF
                ENDIF
            ELSE
                RETURN error
            ENDIF
        ELSE
            RETURN error
        ENDIF
    ELSE
        RETURN error
    ENDIF
ELSE
```

```
        RETURN error
ENDIF

Process:
        READ ballot
        now ← getTime()
        voterSession ← encodebase64(randomBytes)
        hashSession ← bcrypt(voterSession)
        saveToDatabase(nik, id, hashSession, time)
RETURN (ballot, voterSession, ballotSign)
```

Pseudocode 4.8 API Verifikasi Pemilih

4.2.1.2.3 API Masukkan Data Pilihan

REST API ini berada pada URL /cast-vote dan berfungsi untuk menerima dan verifikasi data suara yang dimasukkan lewat mesin *voting* oleh pemilih yang berhak. Data ini yang nantinya akan diteruskan ke *backend*. Fungsi *receive* adalah fungsi untuk menerima data sedangkan fungsi *startProcess* adalah fungsi untuk memulai mengirim data pilihan pemilih. Kode implementasi bisa dilihat pada lampiran A.10.

```
READ id
READ session
READ nik
READ eventID
READ voteSign
READ proofSign
READ ballotSign
READ voterSession
IF id IN database
        match ← compare(hash, session)
        IF match
                IF nik IN database
                        authorized ← authenticate(nik, code)
```

```
                IF authorized
                    exists ← check(nik)
                    IF exists
                        valid ← verify(ballotSign)
                        matchVote ← (hash,
                                    voterSession)
                        IF valid AND matchVote
                            DO Process
                        ELSE
                            RETURN error
                        ENDIF
                    ELSE
                    ENDIF
                ELSE
                    RETURN error
                ENDIF
            ELSE
                RETURN error
            ENDIF
        ELSE
            RETURN error
        ENDIF
    ELSE
        RETURN error
    ENDIF

    Process:
        READ ballotData FROM database
        READ verificator FROM database
        FOR i → ballotData.length
            found ← verify(voteSign,
                        ballotData.candidate_id)
            IF found
                shares ← SPLIT ballotData.candidate_id
                FOREACH verificator
```

```
                cmd ← new_data
                sign ← sign(cmd, eventID, shares)
                data ← JSON(cmd, eventID, shares)
                send(data)
                RETURN true
        ELSE
                RETURN error
        ENDIF
```

Pseudocode 4.9 API Masukkan Data Pilihan

4.2.2 Implementasi Sistem *Backend*

Terdapat dua komponen yang membangun kinerja dari sistem *Backend* yang dirancang pada sistem ini yaitu jaringan *Blockchain*, dan *viewer.*

4.2.2.1 Implementasi Program di *Blockchain*

Program di verifikator berlaku sebagai *manager* di jaringan *Blockchain* yang bertugas untuk melakukan manajemen terhadap data yang masuk dan dikirim dalam jaringan *Blockchain*. Sedangkan *node* berlaku sebagai *worker* yang bertugas melakukan proses *Blockchain*. Berikut merupakan fungsi-fungsi utama yang dijalankan di verifikator.

4.2.2.1.1 Fungsi Mengecek Data Masuk

Fungsi ini dijalankan ketika verifikator mendapat data suara baru dari API *server* yang dimasukkan oleh pemilih. Implementasi kode bisa dilihat pada lampiran A.11.

```
READ sign
READ cmd
READ data
valid ← verify(sign)
```

```
IF valid
    CASE based on cmd
        CASE = new_data
            insert(data)
            broadcast(data)
        CASE = request_data
            found ← search(data)
            IF found
                RETURN found
            ELSE
                reply ← broadcast(data)
                RETURN reply
            ENDIF
    END CASE
ELSE
    RETURN error
ENDIF
```

Pseudocode 4.10 Mengecek Data Masuk

4.2.2.1.2 Fungsi Melakukan Proses *Mining*

Fungsi ini dilakukan untuk mencari nilai *hash* dari *header* suatu blok agar memenuhi *difficulty* yang ditetapkan oleh sistem. Fungsi ini merupakan fungsi utama dari dari proses *Blockchain*, berfuungsi sebagai *proof of work* suatu *node* sebelum membuat data blok yang mempersulit penyerang sistem untuk membuat blok palsu yang serupa dengan blok asli sebelum diterima oleh sistem.

Dalam sistem ini, proses *mining* dijalankan dalam *interval* tertentu agar tidak terlalu memberatkan sistem dibanding jika dilakukan tiap ada data masuk. Implementasi kode bisa dilihat pada lampiran A.12.

```
READ header
INIT target
```

```
INIT nounce ← randomInt()
READ data
root ← merkle(data)
SET MaxUint32
FOR nounce → MaxUint32
    proof ← sha256(concat(header + nounce))
    valid ← check(root)
    IF proof < target AND valid
        header ← concat(header + nounce)
        saveBlock(header + data)
        broadcast(header)
        BREAK
    ENDIF
END FOR
```

Pseudocode 4.11 Proses *Mining*

4.2.2.1.3 Fungsi Membaca Data Blok

Untuk membaca blok yang merupakan data *binary*, diperlukan spesifikasi khusus agar tidak terjadi kesalahan saat membaca data. Implementasi kode bisa dilihat pada lampiran A.13.

```
READ block
header ← READ 0 → 120 bytes FROM block
eventID ← READ 0 → 8 bytes FROM block
length ← READ 8 → 12 bytes FROM block
previousHash ← READ 12 → 44 bytes FROM block
merkleRoot ← READ 44 → 76 bytes FROM block
timestamp ← READ 76 → 80 bytes FROM block
difficulty ← READ 80 → 112 bytes FROM block
nounce ← READ 112 → 116 bytes FROM block
countData ← READ 116 → 120 bytes FROM block
voteData ← READ 120 → length bytes FROM block
```

Pseudocode 4.12 Membaca Data Blok

4.2.2.1.4 Fungsi Membuat *Genesis Block*

Genesis block adalah data blok pertama yang dibuat oleh suatu *node* sebelum dilakukannya proses *Blockchain*. Implementasi kode bisa dilihat pada lampiran A.14.

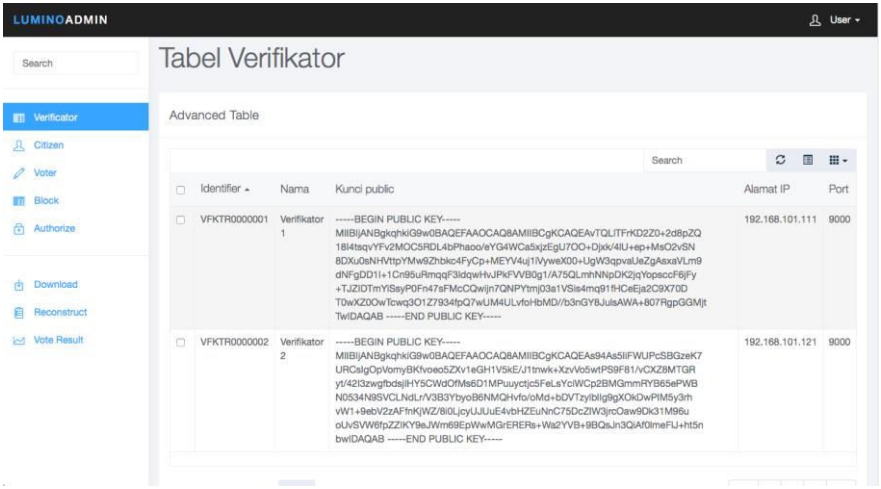
```
READ shareId
READ dataShare
READ block
header ← READ 0 → 120 bytes FROM block
voteData ← READ 120 → length bytes FROM block
finalData ← concat(shareId.length, shareId,
                dataShare.length, dataShare)

READ header
INIT target
INIT nounce ← randomInt()
READ data
root ← merkle(data)
SET MaxUint32
FOR nounce → MaxUint32
    proof ← sha256(concat(header + nounce))
    valid ← check(root)
    IF proof < target AND valid
        header ← concat(header + nounce)
        saveBlock(header + data)
        RETURN success
        BREAK
    ELSE
        RETURN error
    ENDIF
END FOR
```

Pseudocode 4.13 Membuat *Genesis Block*

4.2.2.2 Implementasi Antarmuka di *Viewer*

4.2.2.2.1 Halaman Tampilan Data

Halaman ini berfungsi untuk melihat data yang disimpan pada *shared database* seperti data warga Negara yang telah memilih, data verifikator, dan data blok. Pada Gambar 4.5 merupakan salah satu tampilan halaman data yaitu data verifikator yang terdaftar pada sistem *backend*. Implementasi kode bisa dilihat pada lampiran A.15.



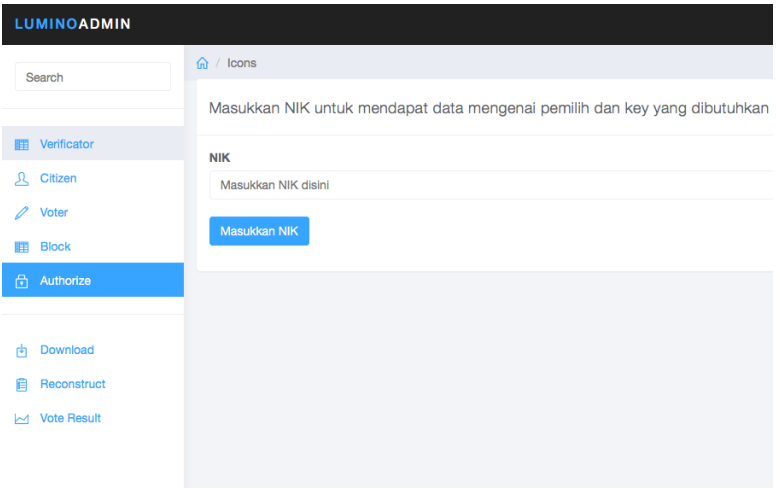
Gambar 4.5 Implementasi Halaman Tampilan Data

```
READ data FROM database
IF data > 0
        render(ui(data))
ELSE
        render(ui(NULL))
ENDIF
```

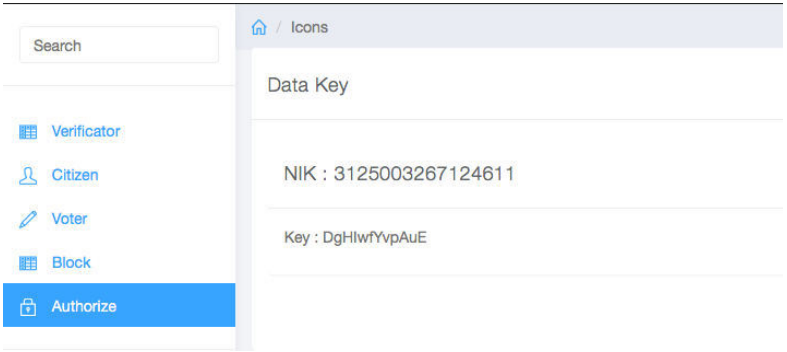
Pseudocode 4.14 Halaman Tampilan Data

4.2.2.2.2 Halaman Autorisasi (*Authorize*)

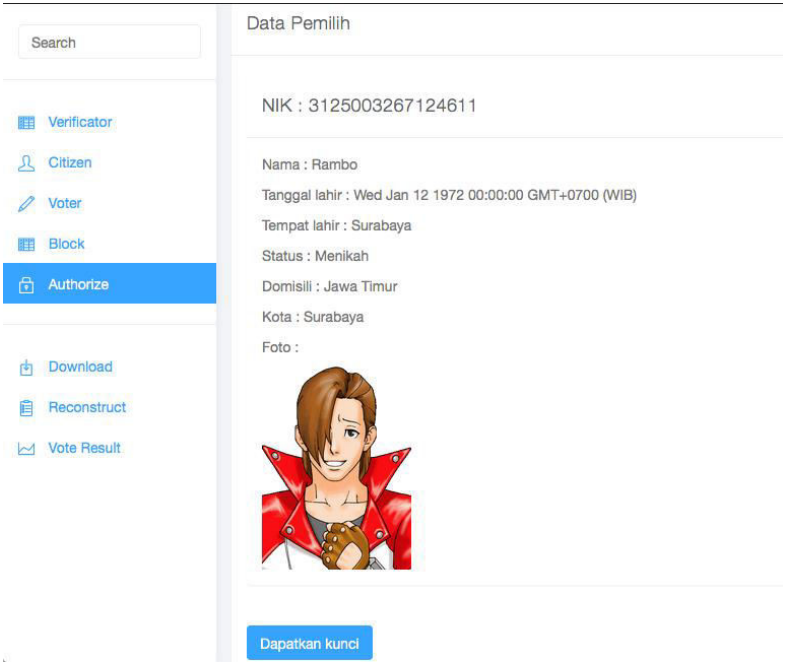
Halaman ini berfungsi untuk menghasilkan kunci acak yang dibuat saat data pemilih dan identitas pemilih sudah dicek oleh panitia penyelenggara pemilu. Untuk melakukan autorisasi ini, terdapat dua langkah yaitu mencocokkan data identitas pemilih dengan data yang ada pada sistem, dan membuat kode acak sebagai kode autorisasi pemilih.



Gambar 4.6 Implementasi Halaman Awal Autorisasi



Gambar 4.7 Implementasi Halaman Tampilan Kode



Gambar 4.8 Implementasi Halaman Informasi

Gambar 4.6 memperlihatkan halaman awal untuk mencari NIK pemilih, Gambar 4.8 menunjukkan halaman informasi pemilih, sedangkan Gambar 4.7 menunjukkan kode yang dihasilkan. Implementasi kode bisa dilihat pada lampiran A.16 dan A.17.

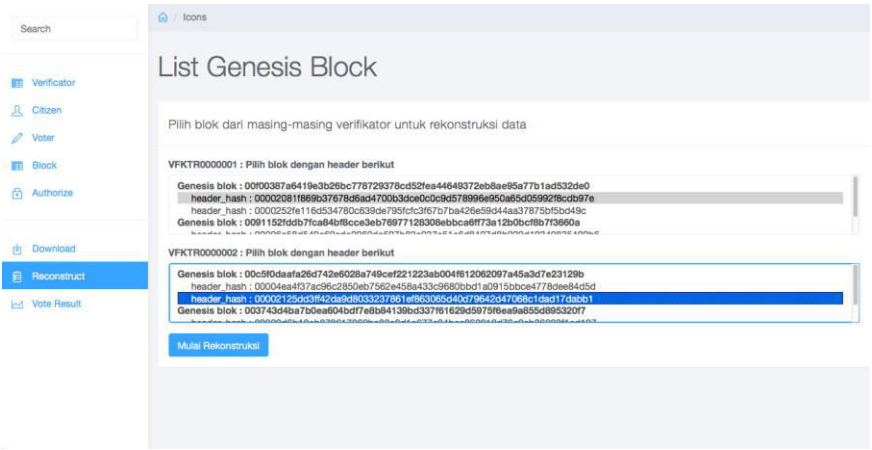
```
READ nik
IF nik IN database
    authorized ← authenticate(nik)
    IF authorized
        done ← check(nik)
        IF done
```

```
                RETURN error
            ELSE
                DO Process
            ENDIF
        ELSE
            RETURN error
        ENDIF
ELSE
        RETURN error
ENDIF

Process:
        voterCode ← encodebase64(randomBytes)
        hashCode  ← bcrypt(voterCode)
        saveToDatabase(hashCode)
        RETURN (voterCode)
```

Pseudocode 4.15 Autorisasi Data Pemilih

4.2.2.2.3 Halaman Rekonstruksi Blok

Halaman ini berfungsi untuk memberikan opsi kepada *administrator* sistem untuk memilih data blok yang dibuat untuk rekonstruksi data hasil kemudian dapat melihat data hasil suara yang masuk dari blok yang dipilih. Gambar 4.9 menunjukkan tampilan halaman rekonstruksi blok. Pada halaman rekonstruksi ini yang ditampilkan merupakan *list* data blok yang tersimpan pada verifikator. Tiap bloknya ditandai dengan *header hash* sebagai *identifier*. Dari semua blok ini, masing-masing berasal dari *genesis block* yaitu blok awal yang digunakan dari proses *Blockchain.* Pengelompokan *list* ini berdasarkan dari blok yang memiliki *genesis block* yang sama dan diurutkan dari atas ke bawah berdasarkan blok yang paling baru dibuat. Semakin panjang proses *Blockhain* yang dilakukan, semakin banyak data blok yang berada pada satu *genesis block* yang ditampilkan. Implementasi kode bisa dilihat pada lampiran A.18.



Gambar 4.9 Implementasi Halaman Rekonstruksi Blok

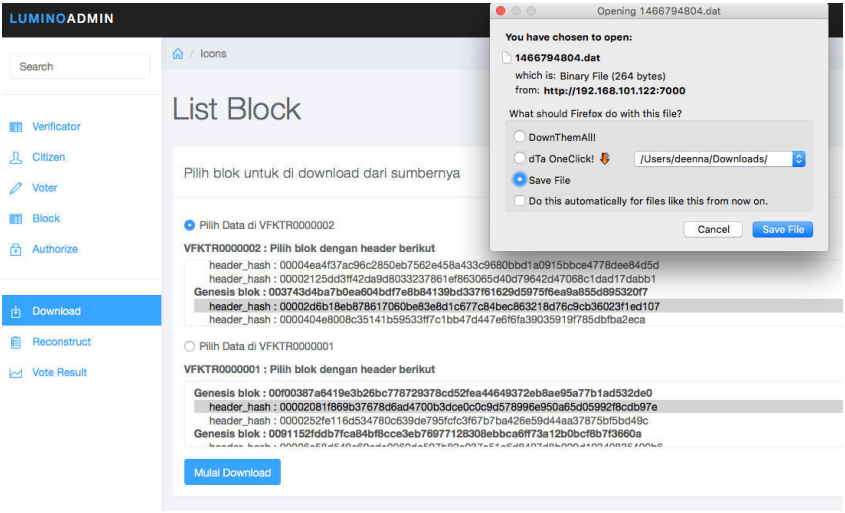
```
READ verificator FROM database
FOREACH verificator
    verifcator.sendRequest(info)
    reply ← response(data)
    IF reply = error
        render(ui(NULL))
    ELSE
        render(ui(reply))
    ENDIF
END FOREACH
```

Pseudocode 4.16 *Request List* Data Blok

4.2.2.2.4 Halaman *Download* Blok

Halaman ini berfungsi untuk memberi opsi bagi *administrator* untuk mengunduh data blok yang diinginkan

apabila data blok yang disimpan sangatlah besar dan tidak memungkinkan untuk melihat hasil suara secara langsung. Blok yang diunduh tadi selanjutnya bisa direkonstruksi secara lokal. Gambar 4.10 memperlihatkan tampilan halaman ini. Implementasi kode bisa dilihat pada lampiran A.19.



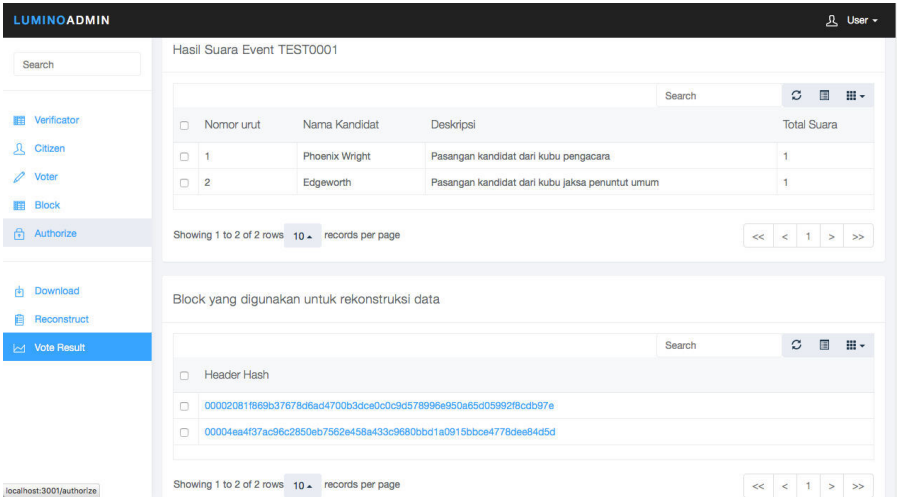
Gambar 4.10 Implementasi Halaman *Download* Blok

```
READ verificator FROM database
FOREACH verificator
    verifcator.sendRequest(info)
    reply ← response(addr, port, name)
    IF reply = error OR reply = no data
        NEXT
    ELSE
        download(addr, port, name)
    ENDIF
END FOREACH
```

Pseudocode 4.17 *Download* Data Blok

4.2.2.2.5 Halaman Hasil Suara

Halaman ini menunjukkan hasil suara yang telah didapat dengan melakukan rekosntruksi data blok yang dipilih atau data blok terbaru yang memiliki status *valid* (telah melalui beberapa tahap pengecekan). Halaman hasil suara ini ditentukan oleh *event_id* yang dipilih sebelumnya. Gambar 4.11 Menunjukkan implementasi tampilan halaman hasil suara ini dimana tabel diatas merupakan hasil suara sedangkan tabel dibawah ada blok yang digunakan untuk rekonstruksi data. Implementasi kode bisa dilihat pada lampiran A.20.



Gambar 4.11 Implementasi Halaman Hasil Suara

```
READ verificator FROM database
FOREACH verificator
        verifcator.sendRequest(info)
        reply ← response(block)
        IF reply = error OR reply = no data
                NEXT
```

```
        ELSE
                valid ← checkBlock(block)
                IF valid
                        data ← getData(block)
                        cache ← data
                ELSE
                        NEXT
                ENDIF
        ENDIF
END FOREACH

shares ← pair(shareId, data)
result ← reconstruct(shares)
RETURN result
```

Pseudocode 4.18 Halaman Hasil Suara

BAB V
HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi pada sistem *E-Vote* yang telah dibangun. Hasil uji coba didapatkan dari implementasi pada bab 4 dengan skenario yang berbeda. Bab ini berisikan pembahasan mengenai lingkungan pengujian dan skenario uji coba yang terdiri dari pengujian fungsionalitas, pengujian keamanan, dan *profiling* program yang telah dibuat.

5.1 Lingkungan Pengujian

Lingkungan pengujian pada sistem *E-Vote* yang dibangun terdiri dari dua macam, lingkungan sistem *frontend*, dan lingkungan sistem *backend*. Spesifikasi perangkat keras dan perangkat lunak di sistem *frontend* ditunjukkan pada 5.1 sedangkan untuk *backend,* seperti ditunjukkan pada Tabel 5.2.

Sistem *backend* menggunakan *Virtual Private Server* (VPS) yang dikelola oleh *Digital Ocean* dengan lokasi fisik *server* berada di Singapura. *Virtual Private Server* ini terhubung satu dengan yang lain dengan menggunakan jaringan VPN. Satu *Virtual Private Server* bertindak sebagai VPN *server* dalam sistem ini dan lainnya bertindak sebagai *node* yang terhubung ke jaringan VPN.

Tabel 5.1 Spesifikasi Lingkungan Pengujian *Frontend*

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	2.7 GHz Intel Core i7
	Memori	4 GB 1333 MHz DDR3

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Lunak	Sistem Operasi	OS X El Capitan versi 10.11.2
	Perangkat Pengembang	Qt Framework

Tabel 5.2 Spesifikasi Lingkungan Pengujian *Backend*

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Harddisk	20 GB
	Memori	512 MB & Swap 1 GB
Perangkat Lunak	Sistem Operasi	Ubuntu 14.04
	Perangkat Pengembang	ExpressJS, MongoDB

5.2 Skenario Uji Coba

Pada bagian ini akan dijelaskan mengenai skenario uji coba yang telah dilakukan. Ada tiga jenis uji coba yang telah dilakukan yaitu:

1) Uji coba fungsionalitas
   Jenis uji coba ini berfungsi untuk menguji fungsionalitas dari sistem yang telah dibuat. Uji coba yang telah dilakukan yaitu:
   - Menguji konfigurasi data dan koneksi aplikasi mesin *voting* saat terhubung ke API *server*.
   - Mengecek data pemilih.
   - Pengecekan data yang terkirim.

- Mencoba rekonstruksi data blok.
- Mengunduh data blok.

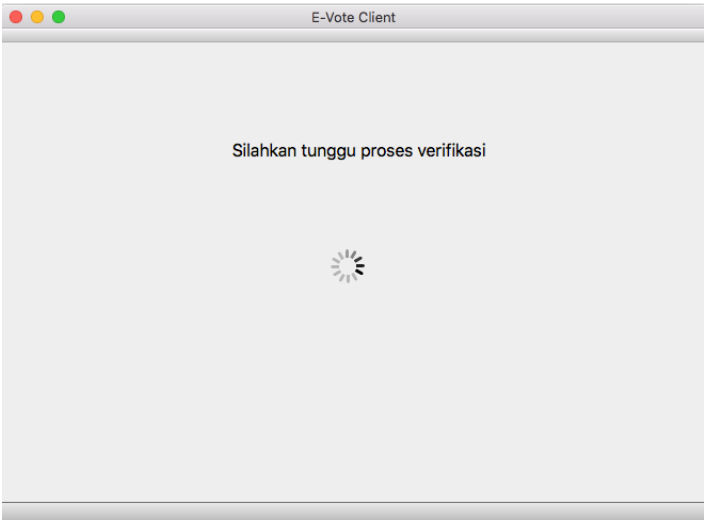
2) Uji coba keamanan
   Jenis uji coba ini berfungsi untuk menguji keamanan dari sistem yang telah dibangun dengan fokus utama keamanan data dan kerahasiaan suara pemilih. Uji coba yang telah dilakukan yaitu:
   - Menguji API.
   - Mengubah data blok.
3) Uji coba monitoring performa (*Profiling*)
   Jenis uji coba ini berfungsi untuk melihat seberapa banyak kekuatan komputasi yang dibutuhkan dalam melakukan proses dalam sistem. Hal yang akan dipantau antara lain:
   - Memori yang dibutuhkan dalam menjalankan proses.
   - Beban kerja CPU.
   - Pengaruh antara *difficulty* dengan waktu proses *mining*.

5.2.1 Pengujian Fungsionalitas

5.2.1.1 Pengujian Konfigurasi Mesin *Voting*

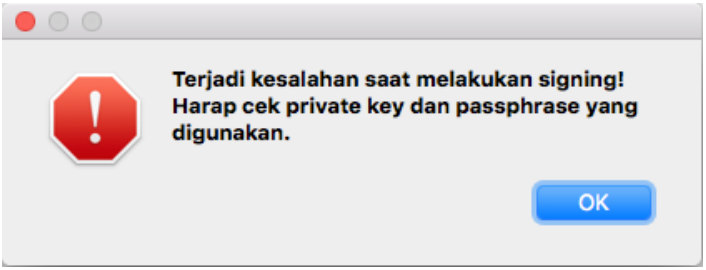
Dalam Pengujian ini, dilakukan uji coba mesin *voting* apabila konfigurasi data dan koneksi yang dibutuhkan oleh aplikasi salah atau tidak sesuai dengan protokol yang telah ditetapkan.

Hal pertama yang diuji coba yaitu apabila data konfigurasi dipasang dengan benar dan koneksi ke API *server* tersambung. Aplikasi mesin *voting* akan menampilkan pesan sesuai dengan kondisi yang dihadapi oleh mesin *voting*. Gambar 5.1 menunjukkan tampilan *loading* apabila konfigurasi dipasang dengan benar dan menunggu jawaban dari API *server*.

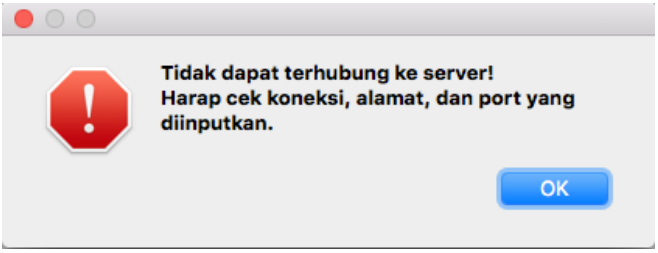


Gambar 5.1 Jendela *Loading* Autentikasi Mesin *Voting*

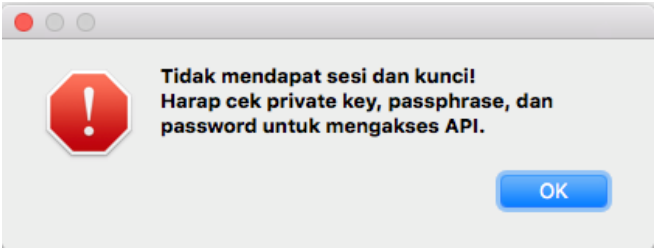
Aplikasi akan menampilkan pesan kesalahan (*error*) yang sesuai dengan kendala yang dihadapi mesin *voting*. Gambar 5.2 menunjukkan kunci privat yang digunakan mesin *voting* bukan *file* kunci privat RSA, Gambar 5.3 Menunjukkan mesin tidak bisa terhubung dengan API *server*, Gambar 5.4 menunjukkan kesalahan pada *password* untuk mengakses API atau *signature* yang salah, dan Gambar 5.5 merupakan tampilan jika berhasil.



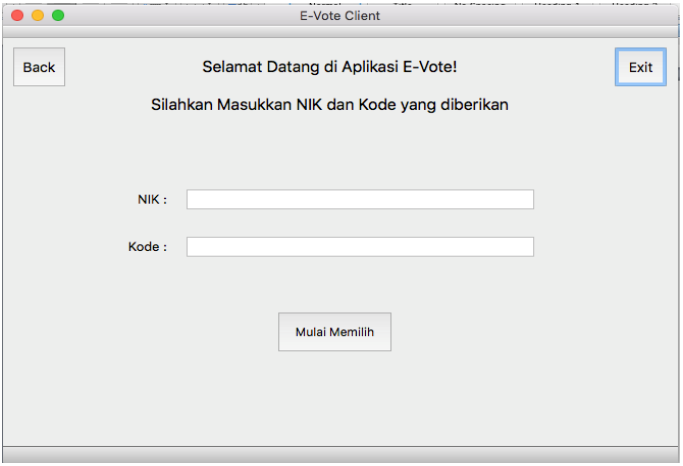
Gambar 5.2 Pesan Kesalahan pada *File* Kunci Privat



Gambar 5.3 Pesan Kesalahan Koneksi ke API *Server*



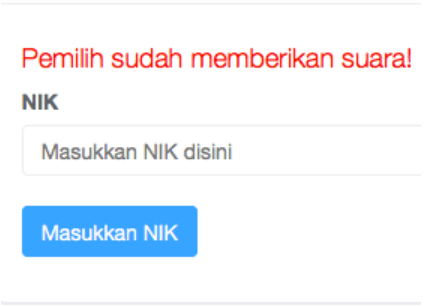
Gambar 5.4 Pesan Kesalahan *Siganture* atau *Password*



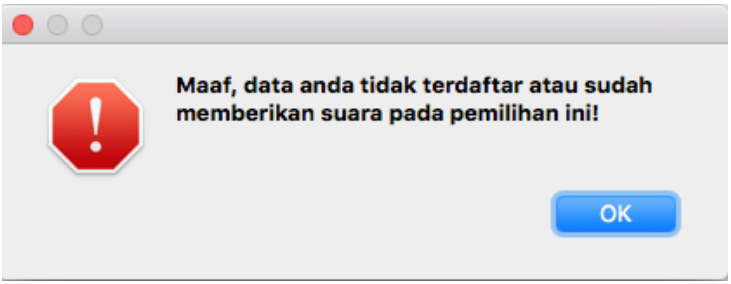
Gambar 5.5 Tampilan Setelah Mesin *Voting* dikonfigurasi

5.2.1.2 Pengecekan Data Pemilih

Dalam Pengujian ini, dilakukan uji coba apabila pemilih telah memberikan suara sebelumnya atau pasangan NIK dan kode yang diinputkan tidak terdaftar dalam sistem, maka aplikasi tidak akan memperbolehkan pemilih untuk melanjutkan proses pemilihan. Gambar 5.6 menunjukkan tampilan pesan di *viewer* kepada *administrator* bahwa pemilih telah memberikan suara atau tidak terdaftar pada sistem, sedangkan Gambar 5.7 menunjukkan pesan kesalahan di mesin *voting* apabila data pemilih tidak sesuai dengan yang terdaftar pada sistem.



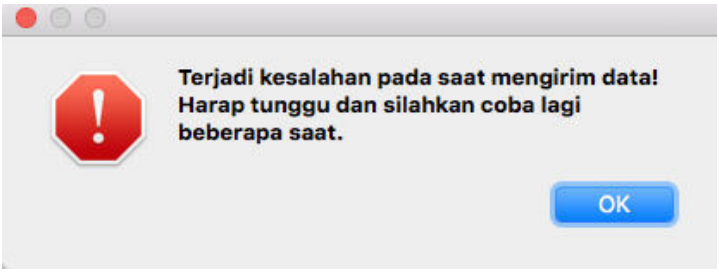
Gambar 5.6 Pesan Kesalahan di *Viewer* saat Autorisasi



Gambar 5.7 Pesan Kesalahan di Mesin *Voting*

5.2.1.3 Pengecekan Data yang Terkirim

Dalam Pengujian ini, dilakukan uji coba apabila sistem siap untuk menerima data yang dikirim oleh pemilih. Apabila sistem belum siap menerima data, maka akan ditampilkan pesan kesalahan seperti pada Gambar 5.8. Jika data yang dikirimkan masuk, maka data pemilih akan terlihat di *viewer* seperti yang terlihat pada Gambar 5.9. Kolom *sgin* pada Gambar 5.9 menunjukkan *signature* bahwa pemilih memilih dari mesin *voting* dengan kunci privat yang cocok dengan *signature* tersebut.



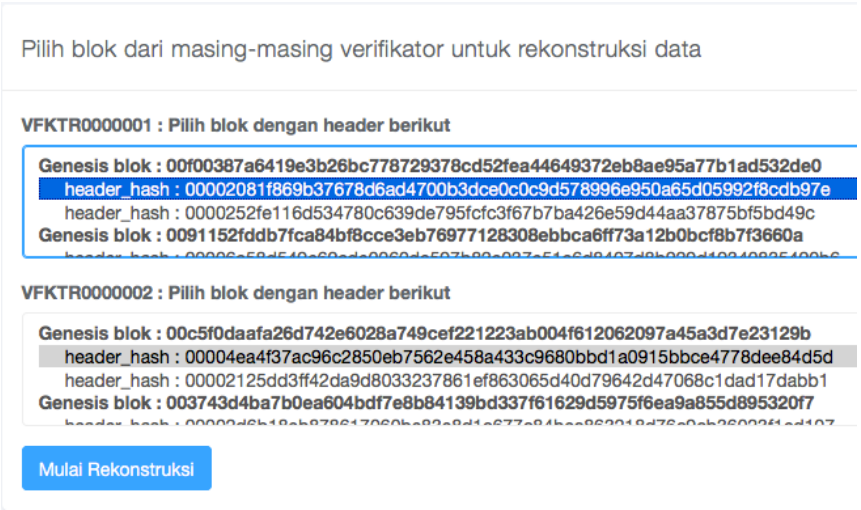
Gambar 5.8 Pesan Kesalahan di Sistem *Backend*

NIK	ID Mesin Voting	Timestamp	Sign
3125003267124642	VMJTSBY00001	Sat Jun 25 2016 01:59:14 GMT+0700 (WIB)	ATvYIuFBZCzN+duYD27PsXJDGSOKUWJ35qpl//DBKA /ai3tCf3S7KVwXn2rW9MuNGVi2OzZxxmqrHJBLxfSO5t /XQuMVpb2mYgdyYXHRLpbILaKkytAaXVZtl7S3ElNgXn
3125001573188651	VMJTSBY00001	Sat Jun 25 2016 02:35:22 GMT+0700 (WIB)	WOMJq6mz0yzkzbLxL+HrL+Qe8N8J+IMhIWm8J/jK1Dv /nV3EDlaV6YqG0pT+iRb2B6sJ0tiAUWhXc1JbDg23AYkl

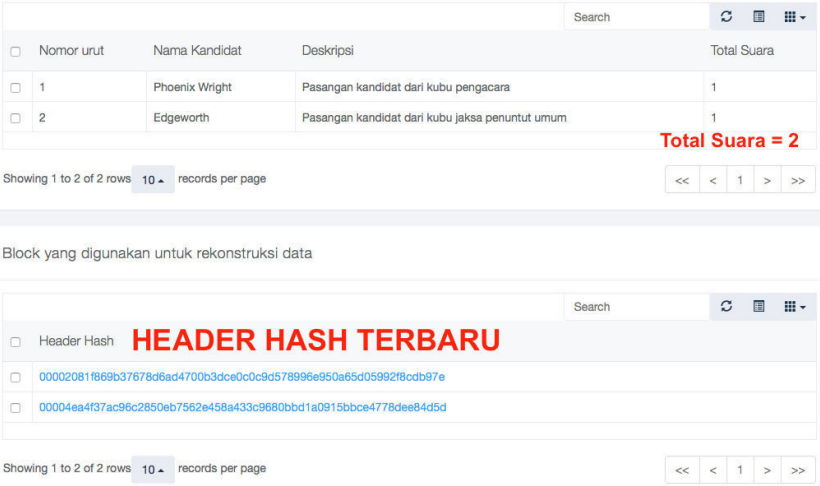
Gambar 5.9 *List* Pemilih yang telah Memberikan Suara

5.2.1.4 Mencoba Rekonstruksi Data

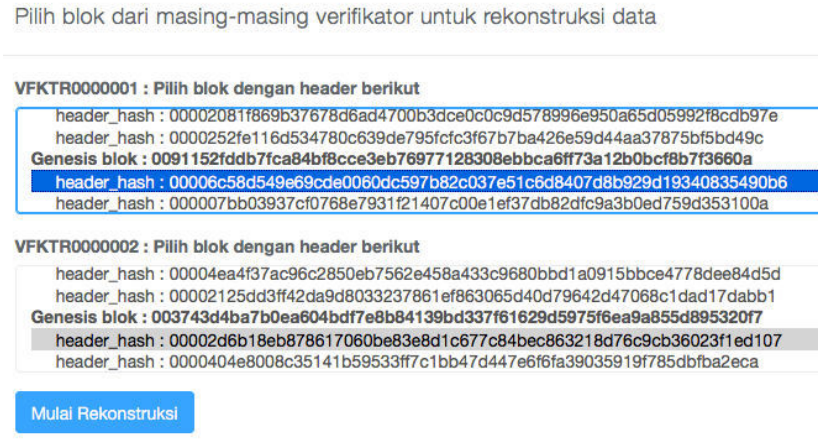
Dalam Pengujian ini, dilakukan uji coba untuk melakukan rekonstruksi data dari blok yang diinginkan sehingga bisa menampilkan data dari blok yang dipilih. Pada Gambar 5.10 dan 5.11 menunjukkan hasil data dari proses *Blockchain* terbaru. Sedangkan pada Gambar 5.12 dan 5.13 menunjukkan hasil data dari proses *Blockchain* sebelumnya. Dari hal ini terlihat, data hasil suara yang disimpan dengan metode ini memiliki semacam *versioning data* dimana kita memiliki data *backup* hasil dari proses *Blockchain* sebelumnya. Jadi apabila satu blok rusak, maka masih ada blok lainnya yang tersimpan dan apabila ada data blok yang diubah, maka blok lainnya bisa melakukan uji *validitas* terhadap blok yang diubah. Apabila data blok yang baru digabungkan dengan data blok yang lama, maka data suara yang bisa di rekonstruksi sebanyak jumlah data yang dimiliki oleh data blok lama.



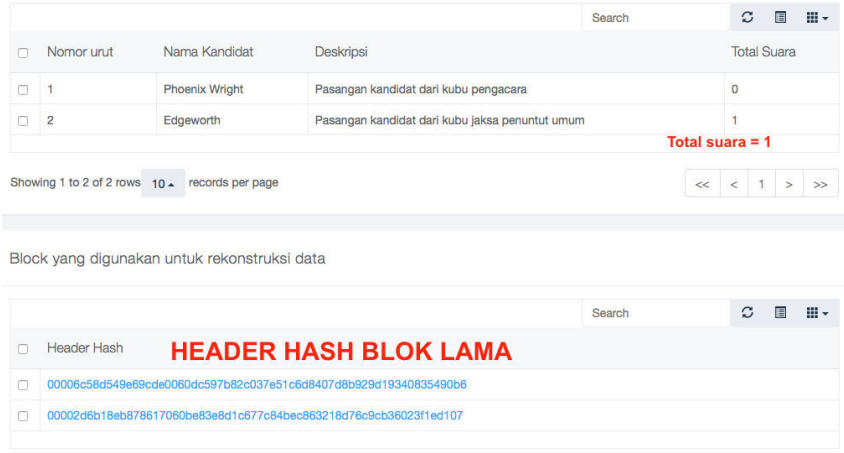
Gambar 5.10 Memilih Blok Paling Baru



Gambar 5.11 Hasil Rekonstruksi Blok Baru



Gambar 5.12 Memilih Blok lama

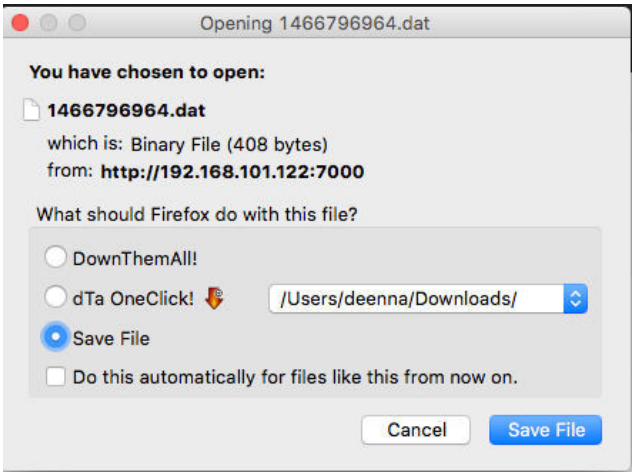


Gambar 5.13 Hasil Rekonstruksi Blok Lama

5.2.1.5 Pengecekan Data Pemilih

Ada kalanya ukuran blok hasil suara yang disimpan memiliki kapasitas yang sangat besar sehingga tidak memungkinkan untuk melakukan rekonstruksi data secara langsung melalui protokol HTTP karena kendala *timeout*.

Gambar 5.14 Hasil Rekonstruksi Blok Lama



Gambar 5.15 Jendela Opsi *Download*

Maka, sistem memberi opsi untuk mengunduh data blok terlebih dahulu yang nantinya bisa diproses lebih lanjut secara lokal. Gambar 5.14 dan 5.15 menunjukkan uji coba fitur *download* ini.

5.2.2 Pengujian Keamanan

5.2.2.1 Menguji API

Dalam uji coba ini, dilakukan pengujian terhadap API yang digunakan. Adapun celah keamanan yang sering dijumpai yaitu celah SQL / NoSQL *injection*. Disini kita akan uji coba mengenai celah keamanan ini.

Pada Gambar 5.16 dan 5.17 digunakan cURL untuk menguji coba kemungkinan terdapat celah keamanan *injection* ini. Referensi untuk melakukan uji coba diambil dari *website* OWASP mengenai NoSQL *injection* [11].

```
Ikhanks-MacBook-Pro:~ ikhank$ curl http://localhost:3000/voteclient/authenticate
 --data "id='$where: function() {verifikator.dropAll()}' "
{"error":"vote machine client not found"}Ikhanks-MacBook-Pro:~ ikhank$
```

Gambar 5.16 Injeksi dengan Memasukkan Fungsi Gagal

```
Ikhanks-MacBook-Pro:~ ikhank$ curl http://localhost:3000/voteclient/authenticate
 --data 'id=(){}""'
{"error":"vote machine client not found"}Ikhanks-MacBook-Pro:~ ikhank$
```

Gambar 5.17 Injeksi dengan Memasukkan Karakter Khusus Gagal

Basis data tidak terkena masalah dan API tetap menjalankan fungsinya, ini membuktikan bahwa Injeksi NoSQL gagal dilakukan terhadap API. Kemudian kita coba mengirim data yang menyerupai mesin *voting*. Terlihat pada Gambar 5.18 hal ini tidak bisa dilakukan dan balasan dari *server* kosong karena dibutuhkan *signature* dari data yang dikirim.

```
Ikhanks-MacBook-Pro:~ ikhank$ curl http://localhost:3000/voteclient/authenticate
 --data 'id=VMJTSBY00001&password=vmclient01&sign=zxczxczzxc'
curl: (52) Empty reply from server
Ikhanks-MacBook-Pro:~ ikhank$
```

Gambar 5.18 Dibutuhkan *Signature* dari Data

5.2.2.2 Mengubah Data Blok

Dalam uji coba ini, dilakukan pengujian *validitas* data blok yang disimpan dalam sistem *Blockchain*. Pada Gambar 5.19 dicoba untuk mengubah data di suatu blok. Dari Gambar tersebut, dapat disimpulkan bahwa mengubah isi data suatu blok dapat dideteksi oleh sistem.

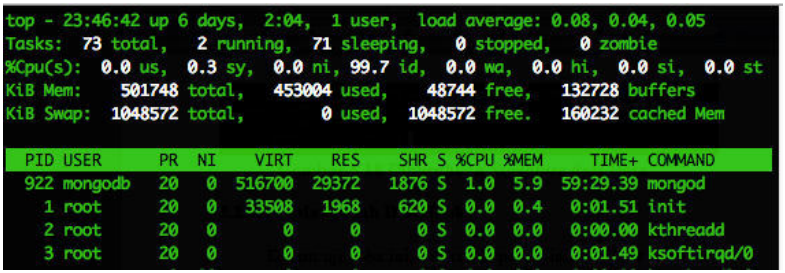
Gambar 5.19 Uji Coba Mengubah Data Blok

Proses nomor 1, mengubah data blok dengan nama 1466794804.dat dengan nilai *hash header* seperti yang tertera pada nomor 2. Lalu kita coba ubah data dari data awal pada nomor 3 menjadi data seperti terlihat pada nomor 4. Kemudian hasilnya bisa kita lihat pada proses 5 pada Gambar 5.19.

5.2.3 *Profiling*

5.2.3.1 Memori yang dibutuhkan dalam Proses

Dari hasil uji coba ini, diperlihatkan Memori yang terpakai waktu menjalankan sistem ini. Disini penulis telah menjalankan sistem yang dirancang di VPS selama 2 hari non stop dan hasilnya terlihat pada Gambar 5.20 dan 5.21.



Gambar 5.20 Monitoring Penggunaan Memori

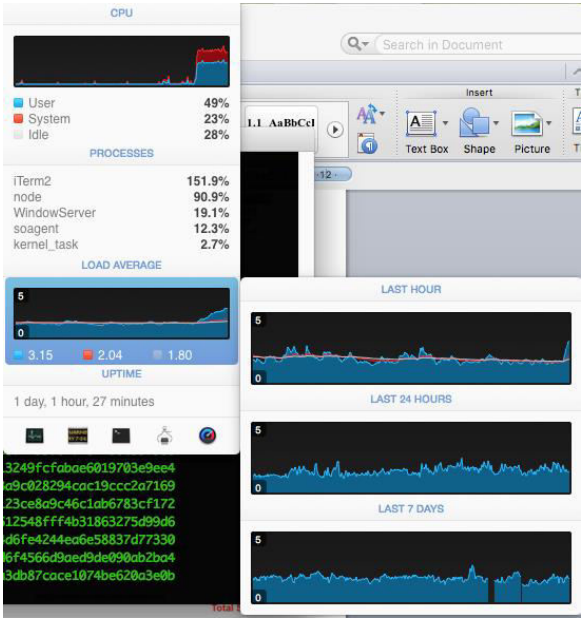
Gambar 5.21 Statistik Penggunaan Memori

Disini terlihat pada Gambar 5.21 bahwa memori yang digunakan adalah sebanyak 479 MB dan penggunaan memori *Swap* 57 MB, total penggunaan memori adalah 536 MB. Sebagian besar penggunaan memori digunakan oleh penggunaan MongoDB sebagai alat *tracking* data blok yang memerlukan

sumberdaya yang cukup intensif. Hal ini terlihat dari Gambar 5.20. Dari uji coba ini dapat disimpulkan bahwa kapasitas memori yang dipakai minimal adalah diatas 600 MB, atau lebih lumrah dengan 1 GB memori.

5.2.3.2 Beban Kerja CPU

Dari hasil uji coba, ketika melakukan proses *mining*, dibutuhkan kira-kira 60-80% dari kekuatan komputasi komputer yang dimiliki. Hal ini membuktikan bahwa proses *mining* sebagai *proof of work* cukup efektif agar penyerang yang ingin mengubah data blok saat proses *mining* memerlukan daya komputasi yang cukup besar. Hal ini dipaparkan pula oleh penemu *Bitcoin* dalam *paper*nya [3]. Gambar 5.22 berikut memperlihatkan grafik penggunaan CPU saat proses *mining* dari aplikasi iStat di OS X.



Gambar 5.22 Penggunaan CPU

5.2.3.1 Pengaruh *Difficulty* terhadap Waktu *Mining*

SHA256 memiliki karakteristik unik pada hasil nilai *hash* yang dihasilkan apabila disimpan dalam *big endian*. Hal ini yaitu nilai *hash* yang dihasilkan akan memiliki angka 0 didepan. Hal ini yang digunakan sebagai *proof of work* dalam proses *mining* pada *Blockchain* sehingga diperlukan hal ini untuk membuat data blok baru di dalam sistem. *Proof of work* ini idealnya dijalankan dalam waktu yang agak lama. Tidak terlalu lama juga tidak terlalu cepat dilakukan. Hal ini diperlukan untuk memenuhi keseimbangan sistem yang dibuat. Berikut adalah hasil data uji coba proses *mining* dengan *target* tertentu dan waktu yang dibutuhkan. Dijalankan pada komputer dengan spesifikasi seperti pada Tabel 5.1.

Tabel 5.3 Hasil Uji Coba *Target* dengan Waktu Proses

Target	Waktu
Hash < 00fffff	1 Detik
Hash < 000fffff	3 Detik
Hash < 0000fffff	10 Detik
Hash < 00000ffff	5 - 10 menit
Hash < 000000fffff	30 menit++
Hash < 00000000fffff	Tidak terhitung, *Max Difficulty,* SHA256 *limit*

Dari data dan hasil uji coba tersebut, dirumuskan *target* sistem yang ideal dengan waktu sekitar 1-5 menit yaitu antara nilai 0000fff dengan 00000fff dan ditemukanlah nilai "00007a0000000000000000000000000000000000000000000000000000000000" sebagai *target* yang ideal pada *Blockchain* sistem *E-Vote* ini.

5.3 Evaluasi Umum Skenario Uji Coba

Dari sejumlah uji skenario fungsionalitas yang telah dilakukan, kontrol-kontrol aplikasi yang dibuat baik dari aplikasi di mesin *voting* maupun program di sistem *backend* semuanya dapat berjalan dengan baik. API *server* yang telah dibangun terbukti aman dari serangan yang umum seperti SQL/NoSQL injection, dan *Impersonation attack* dengan adanya *digital signature*.

Dari sejumlah test keamanan, sistem yang dirancang terbukti dapat mencegah penyerang untuk menyerang sistem. Selain itu, data di sistem *backend* yang disimpan dengan mekanisme *Blockchain* terbukti *tamper-proof* karena adanya mekanisme *proof of work* saat akan membuat data yang dalam hal ini bisa mempersulit membuat data baru bagi komputer dengan sumber daya komputasi yang normal. Juga dengan adanya tes *validitas* lainnya dengan mengecek *merkle root* dan *header hash* membuat data yang disimpan tergolong impraktikal untuk diubah. Apabila terdapat satu blok diubah, maka sistem tidak akan memasukkan blok tersebut dalam perhitungan suara dan akan mengambil sumber data blok lain yang tersedia di *Blockchain*. Selain itu, kita dapat mengecek *validitas* kebenaran suatu blok dengan melakukan *tracing* terhadap rantai hash yang ada pada sistem. Jadi apabila penyerang mencoba untuk mengubah data pada suatu blok, kita bisa mengetahui kebenarannya dari rantai blok sebelumnya.

Meskipun seorang penyerang berhasil menguasai satu jaringan *Blockchain*, jaringan *Blockchain* yang lain tidak akan menyepakati perubahan data yang dilakukan karena data disimpan dalam bentuk *share* yaitu data yang sudah dipecah dengan menggunakan proses *Multi Party Computation*. Dari hal ini, perubahan terhadap penyimpanan data di sistem *backend* bersifat impraktikal, mungkin untuk dilakukan, tapi memerlukan usaha, daya komputasi besar, *administrator* internal, dan waktu

yang cukup lama untuk dapat mengubah data yang telah disimpan dalam sistem ini.

Dari sisi *profiling*, sistem ini memerlukan daya komputasi yang besar dan alokasi memori yang cukup besar, namun hal ini membuat sistem lebih aman dari penyerang yang berusaha mengubah data pada sistem juga adanya semacam *versioning* data sehingga apabila data yang satu rusak atau hilang, terdapat data lain yang berlaku sebagai *backup*.

LAMPIRAN

A.1 Kode Sumber Tombol Browse

```
//Choose Private Key Browse Button
void MenuWindow::choosePrivateKeyPath()
{
   QString path; //Path variable
   QString filename; //Filename variable
   //Open dialog window
   path = QFileDialog::getOpenFileName(
        this,
        "Choose a file to open",
        QString::null,
        QString::null);
   //Get filename
   filename = QFileInfo(path).fileName();
   QStringList list = filename.split(".");
   QString format = list[list.size() - 1];

   //Check format data
   if(list.size() > 1 && (format.toLower() == "pem" ||
      format.toLower() == "der"))
   {
      QString splittedName = "";
      for(int i = 0; i < list.size() -1 ; i++)
      {
         splittedName += list[i];
      }
      //Set Path
      ui->privateKeyPath->setText(path);
      this->myConnection->setPrivateKey(path);
   }
}
```

```
//Choose Setting file Browse Button
void MenuWindow::chooseSettingFilePath()
{
   QString path; //Path variable

   QString filename; //Filename variable

   //Open dialog window
   path = QFileDialog::getOpenFileName(
         this,
         "Choose a file to open",
         QString::null,
         QString::null);

   //Get filename
   filename = QFileInfo(path).fileName();
   QStringList list = filename.split(".");
   QString format = list[list.size() - 1];

   //Check format data
   if(list.size() > 1 && (format.toLower() == "json"))
   {
      QString splittedName = "";

      for(int i = 0; i < list.size() -1 ; i++)
      {
         splittedName += list[i];
      }
      //Set Path
      ui->settingFilePath->setText(path);
      this->myConnection->setVoteMachineId(path);
   }
}
```

```
//Choose Public key Browse Button
void MenuWindow::choosePublicKeyPath()
{
   QString path; //Path variable

   QString filename; //Filename variable

   //Open dialog window
   path = QFileDialog::getOpenFileName(
        this,
        "Choose a file to open",
        QString::null,
        QString::null);

   //Get filename
   filename = QFileInfo(path).fileName();
   QStringList list = filename.split(".");
   QString format = list[list.size() - 1];

   //Check format data
   if(list.size() > 1 && (format.toLower() == "pem" ||
       format.toLower() == "der"))
   {
      QString splittedName = "";

      for(int i = 0; i < list.size() -1 ; i++)
      {
         splittedName += list[i];
      }
      //Set Path
      ui->publicKeyPath->setText(path);
      this->myConnection->setPublicKey(path);
   }
}
```

A.2 Kode Sumber Tombol Lanjutkan

```
//Next button from data config page to connection config page
void MenuWindow::goToHome()
{
   //Get variable from data config before
   QString privkey = ui->privateKeyPath->text();
   QString pubkey = ui->publicKeyPath->text();
   QString setfile = ui->settingFilePath->text();

   //Checking if data is empty
   if(privkey.isEmpty())
   {
      return;
   }

   if(pubkey.isEmpty())
   {
      return;
   }

   if(setfile.isEmpty())
   {
      return;
   }
   //go to home page (connection config page)
   ui->stackedWidget->setCurrentWidget(ui->homePage);
}
```

A.3 Kode Sumber Tombol Kembali

```
void MenuWindow::backToSetting() {
  ui->stackedWidget->setCurrentWidget(ui->settingPage);
}
```

A.4 Kode Sumber Tombol Koneksi

```
//Start Connection to API Server function
void MenuWindow::startConnection()
{
   //Get vote machine ID from configuration file .json
   QString id = this->myConnection->
   getVoteMachineId().value("id").toString();

   //Get input data from input text
   QString password = this->myConnection->
   getConnectionData().value("password").toString();
   QString authString = this->myConnection->
   getVoteMachineId().value("auth_string").toString();

   //Data to be signed
   QString totalstring = id + password + authString;
   QString msg;

   //Signing data with base64 encoding
   int flag = this->myConnection->signingProcess(totalstring,
   1000*1000*10, "base64", &msg);

   //If error signing
   if(flag == -1)
   {
      emit errorType("signing");
      emit connectionError();
   }

   //HTTP POST parameter
   QJsonObject data
   {
      {"id", id},
```

```
    {"password", password},
    {"sign", msg}
  };
  //Get input data from input text
  QString address = this->myConnection->
  getConnectionData().value("address").toString();
  int port = this->myConnection->
  getConnectionData().value("port").toInt();

  //Set REST API from API server
  this->myConnection->setUrl(address +
 "/voteclient/authenticate", port);
  this->myConnection->setParameterData(data);

  //Connection variable
  QEventLoop eventLoop;
  QNetworkAccessManager mgr;
  QNetworkReply *reply;
  QObject::connect(&mgr,
  SIGNAL(finished(QNetworkReply*)),
  &eventLoop, SLOT(quit()));

  //Starting connection
  QNetworkRequest request(this->myConnection->getUrl());
  request.setHeader(QNetworkRequest::ContentTypeHeader,
    "application/x-www-form-urlencoded");

  //Get Reply from Server
  reply = mgr.post(request, this->myConnection->
 getParameterData().toString(QUrl::FullyEncoded).toUtf8());
  eventLoop.exec();
  //Checking if connection error or not
  if (reply->error() == QNetworkReply::NoError) {
    //Reply Data
    QString replyData = (QString)reply->readAll();
```

```
    //Parse data from JSON Reply string
    QJsonParseError err;
    QJsonDocument doc =
    QJsonDocument::fromJson(replyData.toUtf8(), &err);

    //If data from server is JSON Format
    if(doc.isObject())
    {
      //Check key session and serverkey
      if(doc.object().contains("session") &&
         doc.object().contains("serverkey"))
      {
        //Parse data from server reply
        QString sessionStr =
        doc.object().value("session").toString();
        QString serverKeyStr =
        doc.object().value("serverkey").toString();
        QString result;

        //Set data into memory
        this->myConnection->setServerKey(serverKeyStr);
        this->myConnection->setSession(sessionStr);

        emit connectionEstablished();
      }
      else
      {
        //Connection fails because wrong configuration
        qDebug() << "no session or serverkey";
        emit errorType("config");
        emit connectionError();
      }
    }
    else
    {
```

```
            //Data format from server is not valid
            qDebug() << "not object";
            emit errorType("data");
            emit connectionError();
        }
    }
    else {
        //Connection failure
        qDebug() << reply->errorString();
        emit errorType("connection");
        emit connectionError();
    }
}
```

A.5 Kode Sumber Tombol Mulai Memilih

```
//Function to start identifying voter
void MenuWindow::startIdentifying()
{
    //Get data from configuration file .json
    QString id = this->myConnection->
    getVoteMachineId().value("id").toString();
    QString authString = this->myConnection->
    getVoteMachineId().value("auth_string").toString();

    //Get data from input text
    QString nik = this->mySession->
    getVoterData().value("nik").toString();
    QString code = this->mySession->
    getVoterData().value("auth_string").toString();

    //Data to be signed
    QString totalstring = id + authString + nik + code;
    QString msg;
```

```
//Signing process
int flag = this->myConnection->signingProcess(totalstring,
1000*1000*10, "base64", &msg);

//If signing fail
if(flag == -1)
{
   emit errorType("signing");
   emit voterIsInvalid();
}

//Get session given to vote machine from process before
QString session = this->myConnection->getSession();

//HTTP POST parameter
QJsonObject data
{
   {"id", id},
   {"method", "normal"},
   {"session", session},
   {"NIK", nik},
   {"code", code},
   {"sign", msg}
};

//set Data voter
this->mySession->setVoterData(data);

//Setup connection and API
QString address = this->myConnection->
getConnectionData().value("address").toString();
int port = this->myConnection->
getConnectionData().value("port").toInt();

this->myConnection->setUrl(address +
```

```
    "/voteclient/verify", port);
    this->myConnection->setParameterData(data);
    //Connection variable
    QEventLoop eventLoop;
    QNetworkAccessManager mgr;
    QNetworkReply *reply;
    QObject::connect(&mgr,
    SIGNAL(finished(QNetworkReply*)), &eventLoop,
    SLOT(quit()));

    //Start connection
    QNetworkRequest request(this->myConnection->getUrl());
    request.setHeader(QNetworkRequest::ContentTypeHeader,
        "application/x-www-form-urlencoded");

    //Reply from server
    reply = mgr.post(request, this->myConnection->
    getParameterData().toString(QUrl::FullyEncoded).toUtf8());
    eventLoop.exec();

    //If connection error check
    if (reply->error() == QNetworkReply::NoError) {
        //Connection success

        //Reply data from server
        QString replyData = (QString)reply->readAll();

        //Parse data from JSON
        QJsonParseError err;
        QJsonDocument doc =
        QJsonDocument::fromJson(replyData.toUtf8(), &err);

        //If data received was JSON check
        if(doc.isObject())
        {
```

```
//Check key ballot, sign and voter_session from server
   reply
if(doc.object().contains("ballot") &&
   doc.object().contains("sign") &&
   doc.object().contains("voter_session"))
{
   //Get data from reply
   QJsonArray ballot =
   doc.object().value("ballot").toArray();
   QString signStr =
   doc.object().value("sign").toString();
   QString voteSession =
   doc.object().value("voter_session").toString();

   QString ballotStr =
   doc.object().value("ballot_encode").toString();

   //Signing ballot data with error checking
   QString ballotSign;
   int flag2 = this->myConnection->
   signingProcess(ballotStr, 1000*1000*10, "base64",
   &ballotSign);

   if(flag2 == -1)
   {
      emit errorType("signing");
      emit voterIsInvalid();
   }
   //Verifying ballot signature from server
   int flag = this->myConnection->
   verifyProcess(ballotStr, 1000*1000*10, "base64",
   signStr);
   if(flag == -1)
   {
      emit errorType("verify");
```

```
                emit voterIsInvalid();
            }
            else
            if(flag == 0)
            {
                //save data to memory if success
                this->mySession->setVoteSession(voteSession);
                this->mySession->setBallot(ballot);
                this->mySession->setBallotSign(ballotSign);
                emit voterIsValid();
            }
        }
        else
        {
            //If error to verify user because already voted or not
                listed
            emit errorType("auth");
            emit voterIsInvalid();
        }
    }
    else
    {
        //Check if format data from server is JSON
        emit errorType("data");
        emit voterIsInvalid();
    }
  }
  else {
    //Connection error
    emit errorType("connection");
    emit connectionError();
  }
}
```

A.6 Kode Sumber *List* Kandidat

```
//Function to change data if list is being clicked
void MenuWindow::changeData(int index)
{
  //Check if there is candidate list
  if(index > -1)
  {
    //Get ballot data received from API server
    QJsonObject choosen = this->mySession->
    getBallot().at(index).toObject();
    QString number =
    QString::number(choosen.value("candidate_number")
    .toInt(), 10);

    //Set UI according to data
    ui->numberOrder->setText(number);
    ui->descriptionView->
    setText(choosen.value("description").toString());

    //Load base64 First Image From Ballot
    if(!choosen.value("candidate_picture").isNull())
    {
      //Load Image
      QString dataType = this->
      getImageInfo(choosen.value("candidate_picture")
      .toString(), "type");
      QByteArray dataImage (this->
      getImageInfo(choosen.value("candidate_picture")
      .toString(), "data").toStdString().c_str());
      QImage image;
      image.loadFromData(
      QByteArray::fromBase64(dataImage),
      dataType.toStdString().c_str());
      int w = ui->imageOne->width();
```

```
        int h = ui->imageOne->height();
        ui->imageOne->
        setPixmap(QPixmap::fromImage(image)
        .scaled(w,h,Qt::KeepAspectRatio));
    }
    else
    {
        //Load Image default if there's no data
        QImage image;
        image.load(
        this->currentPath + "data/anonym.png", "PNG");
        int w = ui->imageOne->width();
        int h = ui->imageOne->height();
        ui->imageOne->
        setPixmap(QPixmap::fromImage(image)
         .scaled(w,h,Qt::KeepAspectRatio));
    }
    //Load base64 Second Image from Ballot
    if(!choosen.value("additional_picture").isNull())
    {
        QString dataType = this->
        getImageInfo(
        choosen.value("additional_picture")
        .toString(), "type");
        QByteArray dataImage2 (this->
        getImageInfo(
        choosen.value("additional_picture")
        .toString(), "data").toStdString().c_str());
        QImage image;
        image.loadFromData(
        QByteArray::fromBase64(dataImage2),
        dataType.toStdString().c_str());
        int w = ui->imageTwo->width();
        int h = ui->imageTwo->height();
        ui->imageTwo->
```

```
        setPixmap(QPixmap::fromImage(image)
        .scaled(w,h,Qt::KeepAspectRatio));
    }
    else
    {
        //Load image default when there's no data
        QImage image;
        image.load(this->currentPath +
         "data/anonym.png", "PNG");
        int w = ui->imageTwo->width();
        int h = ui->imageTwo->height();
        ui->imageTwo->
        setPixmap(QPixmap::fromImage(image)
        .scaled(w,h,Qt::KeepAspectRatio));
    }
    //Set choosen candidate code
    this->mySession->clearVoteCode();
    this->mySession->
    setVoteCode(choosen.value("vote_code").toString());
  }
}
```

A.7 Kode Sumber Tombol Pilih

```
//Function for send data vote to API server
void MenuWindow::sendVote()
{
  //Get Ballot Event ID
  QJsonObject choosen = this->mySession->
  getBallot().at(0).toObject();

  //Data to be signed
  QString signStr = choosen.value("event_id").toString() +
  this->mySession->getVoteSession() + this->mySession->
  getVoteCode();
```

```
QString sign;
//Signing data
int flag = this->myConnection->signingProcess(signStr,
1000*1000*10, "base64", &sign);
if(flag == -1)
{
   emit errorType("signing");
   emit voterIsValid();
}
//Sleep clearing memory time after signing process
QThread::sleep(2);
//Signing proof data from NIK
QString proofSign;

int flag3 = this->myConnection->signingProcess(this->
mySession->getVoterData().value("NIK").toString(),
1000*1000*10, "base64", &proofSign);
if(flag3 == -1)
{
   emit errorType("signing");
   emit voterIsValid();
}

//clearData if existed
this->mySession->clearVote();

//HTTP POST data parameter
QJsonObject voteData
{
   {"id", this->myConnection->
     getVoteMachineId().value("id").toString()},
   {"NIK", this->mySession->
     getVoterData().value("NIK").toString()},
   {"method", "normal"},
   {"session", this->myConnection->getSession()},
```

```
    {"ballot_sign", this->mySession->getBallotSign()},
    {"event_id", choosen.value("event_id").toString()},
    {"vote_session", this->mySession->getVoteSession()},
    {"vote_sign", sign},
    {"proof_sign", proofSign}
};

//Set URL connection
QString address = this->myConnection->
getConnectionData().value("address").toString();
int port = this->myConnection->
getConnectionData().value("port").toInt();

this->myConnection->setUrl(address +
"/voteclient/cast-vote", port);
this->myConnection->setParameterData(voteData);

//Connection variables
QEventLoop eventLoop;
QNetworkAccessManager mgr;
QNetworkReply *reply;
QObject::connect(&mgr,
SIGNAL(finished(QNetworkReply*)), &eventLoop,
SLOT(quit()));

//Start connection
QNetworkRequest request(this->myConnection->getUrl());
request.setHeader(QNetworkRequest::ContentTypeHeader,
   "application/x-www-form-urlencoded");

//Reply data from API server
reply = mgr.post(request, this->myConnection->
getParameterData().toString(QUrl::FullyEncoded).toUtf8());
eventLoop.exec();
```

```
    //Checking reply data
    if (reply->error() == QNetworkReply::NoError) {
        //If success
        QString replyData = (QString)reply->readAll();
        //Parse data from JSON
        QJsonParseError err;
        QJsonDocument doc =
        QJsonDocument::fromJson(replyData.toUtf8(), &err);
        //If data reply is JSON
        if(doc.isObject())
        {
            //If contains success message or not
            if(doc.object().contains("success"))
            {
                emit waitingFinished();
            }
            else
            {
                emit errorType("send");
                emit voterIsValid();
            }
        }
        else
        {
            //If data from server is not JSON formatted
            emit errorType("data");
            emit voterIsValid();
        }
    }
    else {
        //Connection failure
        emit errorType("connection");
        emit connectionError();
    }
}
```

A.8 Kode Sumber API Autentikasi Mesin *Voting*

```
//API function to authenticate vote machine
function(req, res, next) {
  //Search data vote machine in database which match ID
  Vmclient.findOne({identifier : req.body.id}, function(err,
                          client) {
    //If data found
    if(client != null)
    {
      //Compare hash value with password
      if(bcrypt.compareSync(req.body.password,
         client.password))
      {
        //Generate random bytes using crypto module
        require('crypto').randomBytes(24, function(err,
        buffer) {
          //Encode random bytes to base64
          var sessionRandom = buffer.toString('base64');

          //Hash session random with bcrypt module
          var sessionHash =
          bcrypt.hashSync(sessionRandom, 10);


          //Search data vote machine data in DB
          Vmclient.update(
          { identifier: client.identifier },
          {
            //set new session
            $set : { session: sessionHash }
          },
          { upsert: false },
          function(err, callback) {
```

```
            if(err) {
               res.json({error : err})
            }
            else
            {
               //Get public key of vote machine from DB
               var clientPub =
               ursa.createPublicKey(client.public_key);

               //Supposedly signed data
               rcv = new Buffer(client.identifier +
                        req.body.password +
                        client.auth_string).toString('base64');

               //Verify signature
               if(clientPub.hashAndVerify('sha256', rcv,
                   req.body.sign, 'base64'))
               {
                  //Give vote machine session and server
                    public key
                  pubKey = new Buffer(fs.readFileSync(
                  interface().public_key)).toString('base64');

                  res.json({session : sessionRandom,
                              serverkey : pubKey});
               }
               else
               {
                  //If signature is not valid, throw error
                  res.json({error : "signature invalid"});
               }
            }
         });
      });
   }
```

```
        else
        {
          //If Password of authority is wrong
          res.json({error : "wrong authorization password"});
        }
      }
      else
      {
        //If ID of vote machine given is not found
        res.json({error : "vote machine client not found"})
      }
    });
}
```

A.9 Kode Sumber API Verifikasi Pemilih

```
//Function to verify voter
function(req, res, next) {
  //Find Vote machine data using ID given
  Vmclient.findOne({identifier : req.body.id},
    function(errVm, client) {
    if(errVm)
    {
      //if error
      res.json({error : errVm});
    }
    else
    if(client != null && req.body.session != null)
    {
      //If data vote machine found and session is not null
      //Get citizen data from DB
      Citizen.findOne({NIK: req.body.NIK},
                      function(errCtz, citizen) {
        if(errCtz)
        {
```

```
            //If error
            res.json({error : errCtz});
        }
        else
        if(citizen == null)
        {
            //If citizen not found
            res.json({error : "data citizen not found"});
        }
        else
        {
            //If citizen found
            //Check if Citizen already Voted
            Voter.findOne({NIK: req.body.NIK},
                                function(errVot, voter) {
              if(errVot)
              {
                 //If error
                 res.json({error : errCtz});
              }
              else
              if(voter != null)
              {
                 //If citizen already voted before throw
                    error
                 res.json({error : "This user with NIK : "
                           + citizen.NIK + " already voted"});
              }
              else
              {
                 //If citizen still not voted before
                 if(bcrypt.compareSync(req.body.session,
                  client.session) &&
                  bcrypt.compareSync(req.body.code,
                  citizen.auth_string))
```

```
{
  //Find data ballot with given event_id
  Ballot.find({event_id : Event}, null,
  {sort: {candidate_number: 1}},
  function(errBallot, ballotdata) {
    //If ballot data found
    if(ballotdata != null)
    {
      //Get client public key from DB
      var publicKey =
      ursa.createPublicKey(
      client.public_key);

      //Supposedly signed data
      rcv = new Buffer(client.identifier +
      client.auth_string + citizen.NIK +
      req.body.code).toString('base64');

      //Encode ballot to Base64 form
      var b64ballot = new
      Buffer(JSON.stringify(
      ballotdata)).toString('base64');

      //If signature is valid
     if(publicKey.hashAndVerify(
      'sha256', rcv, req.body.sign,
      'base64'))
      {
        //Get API server private key
        var privateKey =
        ursa.createPrivateKey(
        fs.readFileSync(
        interface().private_key),
        interface().passphrase);
```

```
//Sign base64 data ballot
var sign =
privateKey.hashAndSign(
'sha256', b64ballot, 'utf8',
'base64');
//Generate random bytes with
 crypto module
require('crypto').randomBytes(48,
function(err, buffer) {
  //Encode random bytes to base64
  var sessionRandom =
  buffer.toString('base64');

  //Hash session using bcrypt module
  var sessionShaHash =
  bcrypt.hashSync(
  sessionRandom, 10);

  //Get Date now
  var now = new Date();

  //Marks Citizen as already
  voted in DB
  var newVoter = Voter({
  NIK: req.body.NIK,
  vote_machine_id: req.body.id,
  session: sessionShaHash,
  timestamp: now, sign: null});
  newVoter.save();

  //Reply data to vote machine
  res.json({ballot : ballotdata,
  ballot_encode: b64ballot,
  voter_session: sessionRandom,
  sign: sign});
```

```
                        });
                    }
                    else
                    {
                        //If signature is invalid
                        res.json({error : "Signing
                        invalid"});
                    }
                }
                else
                {
                    //If ballot data not found for given
                      event ID
                    res.json({error : "ballot id not
                                       found"});
                }
            });
        }
        else
        {
            //If data session of vote machine is invalid
            res.json({error : "session is invalid"});
        }
    }
})
    }
});
}
else
{
    //If vote machine not listed in DB
    res.json({error : "vmclient id not found"});
}
});
}
```

A.10 Kode Sumber API Masukkan Data Pilihan

```
//Function for receive data vote
function receive(req, res, next) {
  //Find Vote Machine Data with given ID
  Vmclient.findOne({identifier : req.body.id},
  function(errVm, client)
  {
    if(errVm)
    {
      //If error
      res.json({error : errVm});
    }
    else
      if(client != null && req.body.session != null)
      {
      //If data found and session is not null
      //Get data citizen from NIK
      Citizen.findOne({NIK: req.body.NIK},
        function(errCtz, citizen) {
        if(errCtz)
        {
          //If error
          res.json({error : errCtz});
        }
        else
          if(citizen == null)
          {
          //If citizen not found in DB
          res.json({error : "data citizen not found"});
        }
        else
        {
          //If citizen exists
```

```
//Check if already voted or not
Voter.findOne({NIK: req.body.NIK},
function(errVot, voter) {
  if(errVot)
  {
    //If error
    res.json({error : errCtz});
  }
  else
    if(voter == null)
    {
    //If citizen still not registered as voter
    res.json({error : "Person with this NIK still
    not registered as voter"});
  }
  else
  {
    //If citizen is legit voter
    //Compare vote machine session and
      voter session
    if(bcrypt.compareSync(req.body.session,
      client.session) &&
      bcrypt.compareSync(
      req.body.vote_session, voter.session))
    {
      //If match, get Ballot data
      Ballot.find({event_id :
      req.body.event_id}, null,
      {sort: {candidate_number: 1}},
      function(errBallot, ballotdata) {
        //If ballot data exist
        if(ballotdata != null)
        {
          //Start process input data to backend
          startProcess(ballotdata, req, res, next);
```

```
                    }
                    else
                    {
                        //If data ballot not found
                        res.json({error : "ballot id not
                        found"});
                    }
                });
            }
            else
            {
                //If session is invalid
                res.json({error : "session is invalid"});
            }
        }
    });
}
});
}
else
{
    //If vote machine data ID not listed
    res.json({error : "vmclient id not found"});
}
});
}

//Function to send data to backend
function startProcess(ballotdata, req, res, next) {
    //Get vote machine public key from DB
    var publicKey = ursa.createPublicKey(client.public_key);

    //Encode ballot to base64
    var b64ballot = new
    Buffer(JSON.stringify(ballotdata)).toString('base64');
```

```
//Get ballot hash
var sha256 = crypto.createHash("sha256");
sha256.update(b64ballot, "utf8");
var ballotHash = sha256.digest("hex");
var hashHex = ballotHash.toString('hex');
var hashStr = new
Buffer(b64ballot).toString('base64')

//If ballot hash matched with vote machine signature
//and there is proof data
if(publicKey.hashAndVerify('sha256', hashStr,
    req.body.ballot_sign, 'base64') &&
    req.body.proof_sign != null)
{
   //Get Voter data
   Voter.update(
      { NIK: req.body.NIK,
        vote_machine_id: req.body.id},
   {
      //Set proof sign data
      $set : { sign: req.body.proof_sign }
   },
   { upsert: false },
   function(err, callback) {
      if(err) {
         //If error
         res.json({error : err})
      }
      else
      {
      //Do bruteforce to guess vote code from signature
      //given by vote machine
      var valid = 0;
      //For every candidate data in ballot
      for(var i = 0; i < ballotdata.length; i++)
```

```
{
  //Guess signature
  var voteSign = new Buffer(req.body.event_id +
  req.body.vote_session +
  ballotdata[i].vote_code).toString('base64');
  if(publicKey.hashAndVerify('sha256', voteSign,
     req.body.vote_sign, 'base64'))
  {
    //If exist, get the code from candidate
    //Get private key of API server
    var privateKey =
    ursa.createPrivateKey(
    fs.readFileSync(interface().private_key),
    interface().passphrase);

    //Message queue parameter
    var cmd = 'new_data';
    var event_id = req.body.event_id;
    var vote_data = ballotdata[i].vote_code;

    //signal variable
    var signal = 0;

    //Get random bytes
    require('crypto').randomBytes(48,
    function(err, buffer) {
      //Encode random bytes to base64 as ID
        share
      var share_id = buffer.toString('base64');

      //Get all verificator list
      Verifikator.find({}, function(err,
      verifikatorData) {
        //Get all voter
        Voter.find({}, function(err, countVoter) {
```

```
//Check existing data
if(verifikatorData != null && countVoter
   != null)
{
   //Create share from candidate code
   var shares =
   sssa.create(
   interface().minimum_shares,
   verifikatorData.length, vote_data);

   //Iterate all verificator
   for(var i = 0; i < verifikatorData
                        .length; i++)
   {
      var currentVerificator =
      verifikatorData[i];

      //Share Buffer
      var shareBuff = new
      Buffer(shares[i]);

      //Data to be signed
      var sendBuf = new Buffer(cmd +
      event_id + share_id +
      shareBuff.toString('base64') +
      countVoter.length +
      currentVerificator.auth_string)
      .toString('utf8');

      //Load ZeroMQ modules request
      var zmqReq = zmq.socket('req');
      //Connect to Verificator Message Queue
      zmqReq.connect('tcp://' +
      currentVerificator.ip_address + ':' +
      currentVerificator.port.toString());
```

```
//Message queue parameter
var data = {
   cmd : cmd,
   event_id : event_id,
   share_id : share_id,
   vote_data :
   shareBuff.toString('base64'),
   voter_count : countVoter.length,
   sign :
   privateKey.hashAndSign(
   'sha256', sendBuf, 'utf8', 'base64')
}

//Send data to verificator
zmqReq.send(JSON.stringify(data));

//Listen to reply
zmqReq.on('message',
function(msg) {
   //Get signal
   signal = signal + 1;

   //If reply error
   if(msg.toString() == 'error')
   {
      //supposedly verificator db has
        been tampered
      zmqReq.close();
      res.json({error: "tampered
                  db"});
   }
   //If get all signal from verificator
   if(signal ==
      verifikatorData.length)
   {
```

```
                        //return success to vote machine
                        zmqReq.close();
                        res.json({success: "okay"});
                    }
                });
            }

            //Set timeout from Backend
            setInterval(function(){
                zmqReq.close();
                res.json({error: "no response"});
            }, 10000)
        }
        else
        {
            //If there is inconsistent data
            res.json({error : "Voter count and
                    Verificator count not match"});
        }
    })
})
});
break;
}
else
if(i == ballotdata.length - 1)
{
    //If vote signature is invalid
    res.json({error : "Vote sign is invalid"});
}
}
}
}
);
}
```

```
    else
    {
        //If signature is invalid
        res.json({error : "Signing invalid"});
    }
}
```

A.11 Kode Sumber Fungsi Mengecek Data Masuk

```
//Parse message data
var obj = JSON.parse(msg.toString());
//Variable checking
var validData = 0;
var validRequest = 0;
//Checking command and parameter
//Command request genesis
if(obj.cmd == 'request_genesis' && obj.event_id != null &&
obj.id != null)
{
    //Check signature
    var rcv = new Buffer(obj.cmd + obj.event_id + obj.id +
interface().auth_string).toString('base64');
    if(viewerPub.hashAndVerify('sha256', rcv, obj.sign,
'base64'))
    {
        validRequest = 1;
    }
}
else
//Command request data
if(obj.cmd == 'request_data' && obj.event_id != null)
{
    var rcv = new Buffer(obj.cmd + obj.event_id +
    interface().auth_string).toString('base64');
```

```
   if(viewerPub.hashAndVerify('sha256', rcv, obj.sign,
      'base64'))
   {
      validRequest = 1;
   }
}
else
//Command new data
if(obj.cmd == 'new_data' && obj.event_id != null &&
obj.share_id != null && obj.vote_data != null &&
obj.voter_count != null)
{
   var rcv = new Buffer(obj.cmd + obj.event_id +
   obj.share_id +  obj.vote_data + obj.voter_count +
   interface().auth_string).toString('base64');
   if(apiServerPub.hashAndVerify('sha256', rcv, obj.sign,
      'base64'))
   { validData = 1; }
}
else
//Command request info block
if( (obj.cmd == 'request_info' || obj.cmd == 'request_block' ||
obj.cmd == 'download_data') && (obj.event_id != null &&
obj.header_hash != null) )
{
   var rcv = new Buffer(obj.cmd + obj.event_id +
   obj.header_hash +
   interface().auth_string).toString('base64');
   if(viewerPub.hashAndVerify('sha256', rcv, obj.sign,
   'base64'))
   {
      validRequest = 1;
   }
}
```

A.12 Kode Sumber Fungsi Melakukan Proses *Mining*

```
//Process mining to found golden nounce
var flag = 0;

//Process mining searching for header hash value < 
start_difficulty
for(var i = getRandomInt(0, 4294967294); i < 4294967294; 
i++)
{
   //Iterate nounce i
   nounceBuff.writeUInt32BE(i, 0);

   //Get header data with new nounce
   var headerBuf = Buffer.concat([eventIDBuf, lengthBuff, 
   prevHashBuff, merkleRootBuff, timestampBuff, 
   difficultyBuff, nounceBuff, countDataBuff], totalLength);

   //Get header hash
   var shasum = crypto.createHash('sha256');
   shasum.update(headerBuf);
   var hashNow = new Buffer(shasum.digest('hex'), 'hex');

   //If flag == 1, the header hash satisfies difficulty
   flag = Buffer.compare(startDiffBuff, hashNow, 'hex');
   if(flag == 1)
   {
      //Create block buffer
      var totalBuf = Buffer.concat([eventIDBuf, lengthBuff, 
      prevHashBuff, merkleRootBuff, timestampBuff, 
      difficultyBuff, nounceBuff, countDataBuff, finalData], 
      totalLength + voteDataSize);
      break; //Break from loop
   }
}
```

A.13 Kode Sumber Fungsi Membaca Data Blok

```
// -- Read Block Data --
var blockBuf = fs.readFileSync(__dirname + '/' +
interface().block_dir + blockData.name);
var headerBuf = new Buffer(blockBuf.toString('hex', 0, 120),
'hex');
//Event ID 8 bytes
var eventIDBuf = new Buffer(blockBuf.toString('hex', 0, 8),
'hex');
//Length 4 bytes
var lengthBuff = new Buffer(blockBuf.toString('hex', 8, 12),
'hex');
//Previous Hash 32 bytes
var prevHashBuff = new Buffer(blockBuf.toString('hex', 12,
44), 'hex');
//Merkle root 32 bytes
var merkleRootBuff = new Buffer(blockBuf.toString('hex', 44,
76), 'hex');
//Timestamp 4 bytes
var timestampBuff = new Buffer(blockBuf.toString('hex', 76,
80), 'hex');
//Difficulty 32 bytes
var difficultyBuff = new Buffer(blockBuf.toString('hex', 80,
112), 'hex');
//Nounce 4 bytes
var nounceBuff = new Buffer(blockBuf.toString('hex', 112,
116), 'hex');
//Total Data 4 bytes
var countDataBuff = new Buffer(blockBuf.toString('hex', 116,
120), 'hex');
//Data 120th bytes - end of block
var voteDataBuff = new Buffer(blockBuf.toString('hex', 120,
lengthBuff.readUInt32BE(0)), 'hex');
```

A.14 Kode Sumber Fungsi Membuat *Genesis Block*

```
//Data received
var shareId = new Buffer(obj.share_id, 'base64');
var buffOfData = new Buffer(obj.vote_data, 'base64');

//Save into buffer
var shareIdSize = new Buffer(4);
shareIdSize.writeUInt32BE(shareId.length, 0)
var buffOfDataSize = new Buffer(4);
buffOfDataSize.writeUInt32BE(buffOfData.length, 0);

var finalData = Buffer.concat([shareIdSize, shareId,
buffOfDataSize, buffOfData], 4 + shareIdSize.readUInt32BE(0)
+ 4 + buffOfDataSize.readUInt32BE(0));

//Calculate Data Size
voteDataSize = 4 + shareIdSize.readUInt32BE(0) + 4 +
buffOfDataSize.readUInt32BE(0);

//Get hashes data for merkle process
var shasumForData = crypto.createHash('sha256');
shasumForData.update(finalData);
var hashDataNow = new Buffer(shasumForData.digest('hex'),
'hex');
//push data to array
arrayOfHashes.push(hashDataNow.toString('hex'));
arrayOfData.push(finalData);

if(interface().mode == "DEBUG_DETAIL_MODE")
{
   console.log(arrayOfHashes);
}
```

```
// -- Creating Block --
//Time now
var now = Math.floor( Date.now() / 1000 );
//Name file using timestamp
var stringName = now.toString() + '.dat';
//Event ID buffer
var eventIDBuf = new Buffer(obj.event_id, 0, 8); ;
//Block Length buffer
var lengthBuff = new Buffer(4);
//Calculate length
var totalLength = 8 + 4 + 32 + 32 + 4 + 32 + 4 + 4;
//Save into UInt32
lengthBuff.writeUInt32BE(totalLength + voteDataSize, 0);
//Set previous hash of genesis block as default value
var prevHashBuff = new
Buffer('ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff', 'hex');
//Get merkle root
var tree = merkle('sha256', false).sync(arrayOfHashes);
//Set merkle root buffer
var merkleRootBuff = new Buffer(tree.root(), 'hex');
//Timestamp buffer
var timestampBuff = new Buffer(4);
//Set timestamp to UInt32
timestampBuff.writeUInt32BE(now, 0);
//Start difficulty set
var startDiffBuff = new Buffer(interface().start_difficulty,
'hex');
//Mining difficulty set
var difficultyBuff = new Buffer(interface().mining_difficulty,
'hex');
//Nounce buffer
var nounceBuff = new Buffer(4);
//Data count buffer
var countDataBuff = new Buffer(4);
```

```
//Set data to UInt32
countDataBuff.writeInt32BE(arrayOfData.length, 0);

//Block Create, Flag from process mining
if(flag == 1)
{
   //File descriptor
   var fd = fs.createWriteStream(interface().block_dir +
   now.toString() + '.dat');

   //Set empty array for chain
   var firstArray = [];

   //Write into DB
   var newBlock = Block({name: stringName, header_hash:
   hashNow.toString('hex'), prev_block_name: null, event_id:
   obj.event_id, total_data: countDataBuff.readUInt32BE(0),
   chain_order: firstArray, verified : 1});
   newBlock.save();

   //Write to file total buff from process mining
   fd.write(totalBuf);

   //Send reply
   rep.send('ok');
}
```

A.15 Kode Sumber Halaman Tampilan Data

```
//Function to view data from database
function(req, res, next) {
   //Search all data object in DB
   Object.find({}, function(err, objectData) {
      if(objectData.length == 0)
      {
```

```
        //render view without data
        res.render('view');
    }
    else
    {
        //render view with data
        res.render('view', {data: objectData});
    }
  });
},
```

A.16 Kode Sumber Mengecek Data Pemilih

```
//Function to go to Citizen Info
function(req, res, next) {
  //Search citizien data in DB
  Citizen.findOne({NIK: req.body.nik}, function(errCtz,
                          dataCtz) {
    if(errCtz)
    {
      //If error redirect
      res.redirect('/authorize?error=1');
    }
    else
    if(dataCtz == null)
    {
      //If data citizen not found
      res.redirect('/authorize?error=1');
    }
    else
    {
      //Check whether user has voted before
      Voter.findOne({NIK: req.body.nik}, function(errVtr,
                              dataVtr) {
        if(errVtr)
```

```
        {
          //If error
          res.redirect('/authorize?error=1');
        }
        else
        if(dataVtr == null)
        {
          //If user didn't vote before
          //Create random bytes for session
          require('crypto').randomBytes(24, function(err,
          buffer) {
            //Encode bytes to base64 as session
            var sessionRandom = buffer.toString('base64');
            //Hash session
            var sessionShaHash =
            bcrypt.hashSync(sessionRandom, 10);

            //Set session in DB
            Citizen.update(
              { NIK: req.body.nik },
              {
                //Set session
                $set : { session: sessionShaHash }
              },
              { upsert: false },
              function(err, callback)
              {
                //render info view
                res.render('voter_info', {data: dataCtz,
                session: sessionRandom, nik: req.body.nik});
              });
          })
        }
        else
        {
```

```
                //If user already voted before redirect
                res.redirect('/authorize?error=1');
            }
        });
    }
});
},
```

A.17 Kode Sumber *Generate* Kode Autentikasi

```
// Returns a base-62 (alphanumeric only) string of the given
length
function randomString(length) {
  // We generate a random number in a space at least as big
     as 62^length,
  // and if it's too big, we just retry. This is still statistically
     O(1)
  // since repeated probabilities less than one converge to
      zero. Hat-tip to
  // a Google interview for teaching me this technique! ;)

  // The native randomBytes() returns an array of bytes, each
     of which is
  // effectively a base-256 integer. We derive the number of
      bytes to
  // generate based on that, but note that it can overflow
  after ~150:
  var maxNum = Math.pow(62, length);
  var numBytes = Math.ceil(Math.log(maxNum) /
                      Math.log(256));
  if (numBytes === Infinity) {
    throw new Error('Length too large; caused overflow: ' +
    length);
  }
```

```
    //Generate random
    do {
      //Generate random bytes using crypto modules
      var bytes = crypto.randomBytes(numBytes);
      var num = 0
      for (var i = 0; i < bytes.length; i++) {
        num += Math.pow(256, i) * bytes[i];
      }
    } while (num >= maxNum);

    //Return base62 generated string
    return bases.toBase62(num);
}

//Function to go to generate key page
function(req, res, next) {
    //Get citizen data in DB
    Citizen.findOne({NIK: req.body.nik}, function(errCtz,
                                dataCtz) {
      if(errCtz)
      {
        //If error redirect
        res.redirect('/authorize?error=1');
      }
      else
      if(dataCtz == null)
      {
        //If error redirect
        res.redirect('/authorize?error=1');
      }
      else
      if(bcrypt.compareSync(req.body.session,
          dataCtz.session))
      {
        //If session match
```

```
//Check if already voted before
Voter.findOne({NIK: req.body.nik}, function(errVtr,
                    dataVtr) {
  if(errVtr)
  {
    //If error redirect
    res.redirect('/authorize?error=1');
  }
  else
  if(dataVtr == null)
    //If didn't vote before
    //Generate Random Code
    var keyRandom = randomString(12);

    //Hash random code using bcrypt module
    var keyRandomHash =
    bcrypt.hashSync(keyRandom, 10);

    //Update data
    Citizen.update(
    { NIK: req.body.nik },
    {
      //Set authentication code
      $set : { auth_string: keyRandomHash }
    },
    { upsert: false },
    function(err, callback)
    {
      //Go to code view page
      res.render('get_key', {key: keyRandom, nik:
      req.body.nik});
    });
  }
  else
  {
```

```
            //If already voted redirect
            res.redirect('/authorize?error=1');
          }
        });
      }
    });
}
```

A.18 Kode Sumber *Request List* Data Blok

```
//Function Recoosntruct Page
function(req, res, next) {
  //Find data verificator in DB
  Verifikator.find({}, function(err, verifikatorData) {
    if(verifikatorData.length == 0)
    {
      //If no verificator data, shows no data
      res.render('reconstruct', {data: null, list: null});
    }
    else
    {
      //If verificatot exists
      //Message Queue variable
      var signal = 0;
      var incomeData = [];
      var idList = [];
      //Load Private key
      var privateKey =
      ursa.createPrivateKey(
      fs.readFileSync(interface().private_key),
      interface().passphrase);

      //Set timeout 10 seconds
      var timeup = setInterval(function(){
        if(signal < verifikatorData.length)
```

```
        {
            res.render('reconstruct', {data: incomeData, list:
            idList});
            clearInterval(timeup);
        }
    }, 10000)

    //Iterate every verificator
    for(var i = 0; i < verifikatorData.length; i++)
    {
        //Command to verificator
        var cmd = 'request_genesis';
        //ZeroMQ Module load Request type
        var zmqReq = zmq.socket('req');
        //Current Data
        var currentVerifikator = verifikatorData[i];
        //Data to be signed
        var signBuf = new Buffer(cmd + interface().event_id
        + currentVerifikator.identifier
        + currentVerifikator.auth_string).toString('utf8');
        //connect to message queue
        zmqReq.connect('tcp://' +
        currentVerifikator.ip_address + ':' +
        currentVerifikator.port.toString());

        //Message queue parameter
        var data = {
            cmd : cmd,
            event_id: interface().event_id,
            id : currentVerifikator.identifier,
            sign : privateKey.hashAndSign('sha256', signBuf,
                    'utf8', 'base64')
        }
```

```
//Send data through message queue
zmqReq.send(JSON.stringify(data));

//Listener for reply
zmqReq.on('message', function(msg) {
  //Signal received from verificator incremented
  signal = signal + 1;

  //Check data error
  if(msg.toString() != 'no data' || msg.toString() !=
    'error')
  {
    console.log('got reply -> ', msg.toString());

    //Parse data to json
    var rcvData = JSON.parse(msg.toString());

    //Push data to Array
    idList.push(rcvData.id);
    incomeData.push(JSON.parse(rcvData.data));

    //If all verificator answered
    if(signal == verifikatorData.length)
    {
      //Render view
      res.render('reconstruct', {data: incomeData,
      list: idList});
      clearInterval(timeup);
    }
  }
  else
  {
    //If error and all verificator answered request
    if(signal == verifikatorData.length)
    {
```

```
                    //render only the received data
                    res.render('reconstruct', {data: incomeData,
                    list: idList});
                    clearInterval(timeup);
                }
            }
        });
    }
  }
});
}
```

A.19 Kode Sumber *Download* Data Blok

```
//Function viewing Download Page
function(req, res, next) {
  //Search Verificator data in DB
  Verifikator.findOne({ identifier:
                        req.body.chooseVerifikator},
                        function(err, verifikatorData) {
    if(verifikatorData == null)
    {
      //If no data, return no data
      res.render('download');
    }
    else
    {
      //Command to verificator
      var cmd = 'download_data';
      //Load ZeroMQ Request Module
      var zmqReq = zmq.socket('req');
      //Load Private Key
      var privateKey =
      ursa.createPrivateKey(fs.readFileSync(
      interface().private_key), interface().passphrase);
```

```
        //Seach which block that wants to be downloaded
        for(var key in req.body) {

if(req.body.hasOwnProperty(req.body.chooseVerifikator)) {
            //Get Header Hash of Block
            var searchHash =
                req.body[req.body.chooseVerifikator];
            var signBuf = new Buffer(cmd +
                interface().event_id + searchHash +
                verifikatorData.auth_string).toString('utf8');
          }
        }
        //Connect to verificator
        zmqReq.connect('tcp://' + verifikatorData.ip_address +
                        ':' + verifikatorData.port.toString());
        //Message queue parameters
        var data = {
          cmd : cmd,
          event_id: interface().event_id,
          header_hash : searchHash,
          sign : privateKey.hashAndSign('sha256', signBuf,
                'utf8', 'base64')
        }

        //Send data through message queue
        zmqReq.send(JSON.stringify(data));

        //Listen to message queue reply
        zmqReq.on('message', function(msg) {
          //Error Checking
          if(msg.toString() == 'error' || msg.toString() == 'no
             data')
          {
            //If error, render no data
            res.render('download');
```

```
            }
            else
            {
               //If success, redirect to
               var rcvData = JSON.parse(msg.toString());
               res.redirect('http://' + rcvData.address + ':' +
rcvData.port + '/?file=' + rcvData.name);
            }
            //Close connection
            zmqReq.close();
         })
      }
   });
}
```

A.20 Kode Sumber Halaman Hasil Suara

```
//Function to reconstruct data and view it
function(req, res, next) {
   //Search verificator data
   Verifikator.find({}, function(err, verifikatorData) {
      if(verifikatorData.length == 0)
      {
         //If no data, return empty
         res.render('result', {data: []});
      }
      else
      {
         //If success
         //Get cache folder file list
         var cacheFolder =
         fs.readdirSync(interface().cache_dir);

         //Clear all cache file
         for (var i = 0; i < cacheFolder.length; i++) {
```

```
        fs.unlinkSync(interface().cache_dir + cacheFolder[i]);
    };
    //variable process
    var signal = 0;
    var pairedData = [];
    var trueData = [];
    var blockHeader = [];

    //Load private key
    var privateKey =
    ursa.createPrivateKey(
    fs.readFileSync(interface().private_key),
    interface().passphrase);

    //Iterate verificator data
    for(var i = 0; i < verifikatorData.length; i++)
    {
        //Command to verificator
        var cmd = 'request_data';
        //Load ZeroMQ Request Module
        var zmqReq = zmq.socket('req');
        //Get Current data
        var currentVerifikator = verifikatorData[i];
        //Data to be signed
        var rcv = new Buffer(cmd + interface().event_id +
        currentVerifikator.auth_string).toString('utf8');
        //Connect to verificator
        zmqReq.connect('tcp://' +
        currentVerifikator.ip_address + ':' +
        currentVerifikator.port.toString());

        //Message queue parameters
        var data = {
            cmd : cmd,
            event_id : interface().event_id,
```

```
    sign : privateKey.hashAndSign('sha256', rcv, 'utf8',
          'base64')
}

//Send data through message queue
zmqReq.send(JSON.stringify(data));

//Listen reply
zmqReq.on('message', function(msg) {
  //Increment signal received
  signal = signal + 1;
  var incomeData = [];
  var errFlag = 0

  //Checking Error
  if(msg.toString() == 'error' || msg.toString() == 'no
     data')
  {  //If error
     errFlag = 1;
  }

  if(errFlag == 0)
  {
     //If not error
     //Read received block
     var blockBuf = new Buffer(msg.toString(),
     'base64');
     var headerBuf = new
     Buffer(blockBuf.toString('hex', 0, 120), 'hex');


     //Read each block data bytes
     var eventIDBuf = new
     Buffer(blockBuf.toString('hex', 0, 8), 'hex');
     var lengthBuff = new
```

```
Buffer(blockBuf.toString('hex', 8, 12), 'hex');
var prevHashBuff = new
Buffer(blockBuf.toString('hex', 12, 44), 'hex');
var merkleRootBuff = new
Buffer(blockBuf.toString('hex', 44, 76), 'hex');
var timestampBuff = new
Buffer(blockBuf.toString('hex', 76, 80), 'hex');
var difficultyBuff = new
Buffer(blockBuf.toString('hex', 80, 112), 'hex');
var nounceBuff = new
Buffer(blockBuf.toString('hex', 112, 116), 'hex');
var countDataBuff = new
Buffer(blockBuf.toString('hex', 116, 120), 'hex');
var voteDataBuff = new
Buffer(blockBuf.toString('hex', 120,
lengthBuff.readUInt32BE(0)), 'hex');

//Get block header hash
var shasum = crypto.createHash('sha256');
shasum.update(headerBuf);
var headerHashCurrent = new
Buffer(shasum.digest('hex'), 'hex');

//Test if block satisfies difficulty
var difficultyFlag = Buffer.compare(difficultyBuff,
headerHashCurrent, 'hex');

if(difficultyFlag == 1)
{
  //If pass test
  //Variable for reading block
  var arrayOfHashes = [];
  var arrayOfNextHashes = [];
  var counterBytes = 0;
  var endBytes = 0;
```

```
var rawData = [];

//Iterate each data in block
for(var j = 0; j <
countDataBuff.readUInt32BE(0); j++)
{
  //Get ID share size and Share data size
  var idSizeBuff =
  voteDataBuff.readUInt32BE(counterBytes);
  var dataSizeBuff =
  voteDataBuff.readUInt32BE(idSizeBuff + 4);

  //Iterate
  endBytes = endBytes + 4 + idSizeBuff + 4 +
  dataSizeBuff;

  //Get share_id and share data
  var eachDataBuff = new
  Buffer(voteDataBuff.toString('hex',
  counterBytes, endBytes), 'hex');
  var eachIdStringBuff = new
  Buffer(voteDataBuff.toString('hex',
  counterBytes + 4, counterBytes + idSizeBuff
  + 4), 'hex');
  var eachDataStringBuff = new
  Buffer(voteDataBuff.toString('hex',
  counterBytes + idSizeBuff
  + 8, endBytes), 'hex');

  //Decode from base64
  var decodedEachDataStringBuff = (new
 Buffer(eachDataStringBuff.toString('base64'),
  'base64')).toString('utf8');
```

```
        //Get hash
        var shasumForData =
        crypto.createHash('sha256');
        shasumForData.update(eachDataBuff);
        var hashDataNow = new
        Buffer(shasumForData.digest('hex'), 'hex');

        //Push each data to array
        rawData.push({id:
        eachIdStringBuff.toString('base64'), code:
        decodedEachDataStringBuff});

        arrayOfHashes.push(
        hashDataNow.toString('hex'));
        arrayOfNextHashes.push(
        hashDataNow.toString('hex'));

        //Iterate
        counterBytes = counterBytes + 4 + idSizeBuff
        + 4 + dataSizeBuff;
    }

    //Get merkle root
    var treeCurrent = merkle('sha256',
    false).sync(arrayOfHashes);

    //Check merkle root
    if(treeCurrent.root() ==
    merkleRootBuff.toString('hex'))
    {
        //Push each header hash to array
        blockHeader.push(
    headerHashCurrent.toString('hex'));
        //Write raw data to cache
        var fd = fs.writeFileSync(
```

```
            interface().cache_dir + 'block_' +
            signal.toString() + '.json',
            JSON.stringify( rawData ), "utf8");
        }
    }
    else
    {
        //If failed
        console.log('failed');
    }

    if(signal == verifikatorData.length)
    {
        //Close connection
        zmqReq.close();

        //Varible process
        var blocks = [];
        var allPairs = [];
        var TryResult = [];

        //Read cache folder
        cacheFolder =
        fs.readdirSync(interface().cache_dir);
        for(var j = 0; j < cacheFolder.length; j++)
        {
            //Push raw data to array for further process
            var dataCache =
            JSON.parse(fs.readFileSync(
            interface().cache_dir + cacheFolder[j]));
            for(var k = 0; k < dataCache.length; k++)
            {
                blocks.push(dataCache[k]);
            }
        }
```

```
//Algorithm to pairing share data from
  different blockchain
//Each elements has flag 0
var flagArr = [];

for (var j = 0; j < blocks.length; j++)
{
  //set flags to 0
  flagArr.push(0);
}
//Iterate all data in rawData array
for (var j = 0; j < blocks.length - 1; j++)
{
  //Array for pairs with same ID Share
  var pairs = [];
  var any = 0;
  for(var k = j+1; k < blocks.length; k++)
  {
    //If raw data has the same ID Share, then
      pair them
    if(blocks[j].id == blocks[k].id && flagArr[j]
       == 0)
    {
      //Set flag to 1 as it has same ID with base
        data
      flagArr[k] = 1;
      any = 1;
      pairs.push(blocks[k]);
      console.log(flagArr);
    }
  }
  //Set flag to 1 in base data as already
    processed
  flagArr[j] = 1;
  if(any == 1)
```

```
            {
                //If it has any pair, push data to pairs
                pairs.push(blocks[j]);
                //Push all pairs to another array, array 2
                    dimensions
                allPairs.push(pairs);
            }
        }
        //We get all pairs, now reconstruct it
        //Iterate all pairs
        for(var j = 0; j < allPairs.length; j++)
        {
            //Array for saving share
            var shares = [];
            //Get all data from same ID
            for(var k = 0; k < allPairs[j].length; k++)
            {
                shares.push(allPairs[j][k].code);
            }
            //If total shares received >= minimum shares
            //It can be reconstructed
            if(shares.length >=
                interface().minimum_shares)
            {
                //Push Result
                TryResult.push(sssa.combine(shares));
            }
        }
        //Get data ballot according to current event ID
        Ballot.find({event_id : interface().event_id},
        function(errBallot, ballotData) {
            //render data result
            res.render('result', {data: ballotData, result:
            TryResult, block: blockHeader});
        })
```

```
                }
            }
            else
            {
                if(signal == verifikatorData.length)
                {
                    //If get data error
                    zmqReq.close();
                    //render empty
                    res.render('result', {data: incomeData, result: [],
                                block: []});
                }
            }
        });
    }
    //Set timeout 20 seconds
    var timeup2 = setInterval(function(){
        if(signal < verifikatorData.length)
        {
            //If signal received is not enough
            var incomeData = [];
            //render empty data
            res.render('result', {data: incomeData, result: [],
                block: []});
            clearInterval(timeup2);
        }
    }, 20000)
  }
 });
}
```

A.21 Contoh *File* Konfigurasi untuk Mesin *Voting*

```
{
  "id": "VMJTSBY00001",
  "name" : "Voting Machine TPS 1 Surabaya",
  "authority": "Bambang Sudibyo",
  "province" : "Jawa Timur",
  "region" : "Surabaya",
  "city" : "Kenjeran",
  "place_number" : 1,
  "auth_string" : "test-string-vote-machine",
  "private_key" : "data/private.pem",
  "public_key"  : "data/public.pem"
}
```

BAB VI
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan mekanisme yang diajukan ini nantinya.

6.1 Kesimpulan

Kesimpulan yang diperoleh berdasarkan uji coba dan evaluasi yang dilakukan antara lain:

1) Dengan melihat semua uji fungsionalitas, semua kontrol aplikasi berjalan dengan baik dan implementasi sistem yang dirancang berjalan dengan baik pula.
2) Rancangan sistem *backend* dengan menggabungkan metode *Multi Party Computation* dengan *Blockchain* terbukti menjaga integritas hasil *voting* karena untuk membuat data baru atau mengubah data memerlukan daya komputasi, waktu dan persetujuan semua *node* yang terhubung di jaringan yang mana hal ini dianggap impraktikal.
3) Data yang diterima ditiap komunikasi antar *node* memiliki *signature* yang mana hal ini bisa mencegah *impersonation attack*.
4) Data yang diubah akan langsung dapat diketahui dan dilihat pada sistem *backend* yang dirancang.
5) Hasil suara yang masuk disimpan dan terdiri dari beberapa versi data mulai dari awal sampai paling baru. Hal ini membuat data yang disimpan memiliki *backup* jika terjadi data rusak di satu blok
6) Tiap mesin *voting* bersifat unik sehingga orang yang tidak memiliki konfigurasi, aplikasi, dan kunci yang diperlukan tidak dapat mengakses API *server*.

7) Data pilihan pemilih tidak dikirim dalam bentuk *plaintext* namun disisipkan dalam *signature* dan disimpan di *backend* dalam bentuk *share*.
8) Menghitung data hasil suara dilakukan dengan mengumpulkan blok dari jaringan *Blockchain* yang berbeda lalu direkonstruksi dengan *Multi Party Computation*.

6.2 Saran

Beberapa saran yang hendak disampaikan terkait dengan pengerjaan tugas akhir ini:

1) Semakin banyak jumlah jaringan *Blockchain* yang disediakan dan jumlah *node* yang terhubung, semakin terjaga integritas data yang disimpan pada sistem ini.
2) Penggunaan metode autentikasi yang lebih aman dari sekedar kode yang diacak dan tidak perlu bergantung pada panitia lebih direkomendasikan misal *scan* sidik jari, penggunaan E-KTP, NFC dan lainnya.
3) Penyesuaian data *ballot* dengan tampilan aplikasi di mesin *voting* untuk tiap kasus yang berbeda misal pemilihan DPRD dan partai.

DAFTAR PUSTAKA

[1] Shamir, A, "How to share a secret," *Communication of the ACM*, 1979, 22, 612-13.

[2] Andrew C. Yao, "Protocols for secure computations," *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, p.160-164, November 03-05, 1982 [doi>10.1109/SFCS.1982.88]

[3] Nakamoto, Satoshi, "Bitcoin: A Peer-to-peer Electronic Cash System," (2009), *Bitcoin Organization*, Web. 14 Dec. 2015.

[4] Nair, Divya G, V. P. Binu, and G. Santhosh Kumar, "An Improved E-voting Scheme Using Secret Sharing Based Secure Multi-party Computation," *2014 International Conference on Data Science & Engineering (ICDSE)* (2014), *Digital Library CUSAT*, Web. 14 Dec. 2015.

[5] Guy Zyskind, Oz Nathan, Alex 'Sandy' Pentland, "Decentralizing Privacy: Using Blockchain to Protect Personal Data", *2015 IEEE Security and Privacy Workshops (SPW) 2015*, pp. 180-184, doi:10.1109/SPW.2015.27

[6] A. Wibawa, Interviewee, *Eropa Menarik Diri dari e-Voting, Indonesia sedang Tertarik*. [Wawancara]. 8 November 2014

[7] Kementrian Dalam Negeri Republik Indonesia, "Desa Mendoyo, Contoh "e-Voting" Pertama," 31 Juli 2013. [Online]. Available : http://otda.kemendagri.go.id/index.php/categoryblog/1072-desa-mendoyo-contoh-qe-votingq-pertama. [Diakses 20 Juni 2016]

[8] R. Sadikin, Kriptografi untuk Keamanan Jaringan, Yogyakarta: Andi, 2012.

[9] National Institute of Standard and Technology, “Federal Information Processing Standard (FIPS) Publication 180-2 Announcing the Secure Hash Standard,“ 1 Agustus 2002.

[10] D. Wood, V. Stoss, L. Chan-Lizardo, G. S. Papacostas and M. E. Stinson, "Virtual private networks," *Private Switching Systems and Networks, 1988., International Conference on*, London, 1988, pp. 132-136.

[11] OWASP Foundation, “Testing for NoSQL Injection”, 20 Agustus 2014, [Online]. Available : https://www.owasp.org/index.php/Testing_for_NoSQL _injection [Diakses 26 Juni 2016]

BIODATA PENULIS

R. M Iskandar Zulkarnaen merupakan anak dari pasangan Bapak R. Suprapto dan Ibu Siti Towiyah. Lahir di Bangkalan pada tanggal 10 September 1993. Penulis menempuh pendidikan formal dimulai dari TK YKK Bangkalan (1999-2000), SD Demangan 1 Bangkalan (2000-2006), SMP Negeri 2 Bangkalan (2006-2009), SMA Negeri 1 Bangkalan (2009-2012) dan S1 Teknik Informatika ITS (2012-2016). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Ikatan Mahasiswa Bangkalan di ITS (IMABITS) (2012-2015), ITS *Foreign Language Society* (IFLS) (2012-2013), Himpunan Mahasiswa Teknik Computer-Informatika (2013-2014), dan Keluarga Muslim Informatika (2013-2014). Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2013, INOCHI 2013, dan Parade KMI (2013). Penulis juga aktif mengikuti lomba dan menjadi finalis seperti Gemastik 7 (2014) dan Gemastik 8 (2015). Penulis memiliki hobi membaca, menggali informasi, dan menyukai hal baru. Penulis dapat dihubungi melalui email: handyboy.ikhank@gmail.com