Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – EC184801

PENGEMBANGAN SISTEM KEAMANAN BERBAGI DATA PACS BERBASIS BLOCKCHAIN

Dito Prabowo
NRP 07211640000004

Dosen Pembimbing
Reza Fuad Rachmadi, ST., MT., Ph.D.
Dr. I Ketut Eddy Purnama, ST., MT.

DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

TUGAS AKHIR – EC184801

PENGEMBANGAN SISTEM KEAMANAN BERBAGI DATA PACS BERBASIS BLOCKCHAIN

Dito Prabowo
NRP 07211640000004

Dosen Pembimbing
Reza Fuad Rachmadi, ST., MT., Ph.D.
Dr. I Ketut Eddy Purnama, ST., MT.

DEPARTEMEN TEKNIK KOMPUTER
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

FINAL PROJECT – EC 184801

DEVELOPMENT OF A BLOCKCHAIN-BASED PACS DATA SHARING SECURITY SYSTEM

Dito Prabowo
NRP 07211640000004

Advisor
Reza Fuad Rachmadi, ST., MT., Ph.D.
Dr. I Ketut Eddy Purnama, ST., MT.

DEPARTMENT OF COMPUTER ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

Halaman ini sengaja dikosongkan

PERNYATAAN KEASLIAN
TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul **"Pengembangan Sistem Keamanan Berbagi Data PACS berbasis Blockchain"** adalah benar-benar hasil karya intelektual sendiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya orang lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun durujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai dengan peraturan yang berlaku.

Surabaya, 31 Juli 2020

Dito Prabowo
NRP. 07211640000004

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

Pengembangan Sistem Keamanan Berbagi Data PACS Berbasis *Blockchain*

Tugas Akhir ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Teknik di Institut Teknologi Sepuluh Nopember Surabaya

Oleh: Dito Prabowo (NRP: 07211640000004)

Tanggal Ujian : 06 Juli 2020 Periode Wisuda : September 2020

Disetujui oleh:

Reza Fuad Rachmadi, ST., MT., Ph.D. (Pembimbing I)
NIP: 198504032012121001

Dr. I Ketut Eddy Purnama, ST., MT. (Pembimbing II)
NIP: 196907301995121001

Prof. Dr. Ir. Mauridhi Hery Purnomo, M.Eng. (Penguji I)
NIP: 195809161986011001

Dr. Surya Sumpeno, ST., M.Sc. (Penguji II)
NIP. 196906131997021003

Dr. Adhi Dharma Wibawa, ST., MT. (Penguji III)
NIP. 197605052008121003

Mengetahui
Kepala Departemen Teknik Komputer

Dr. Supeno Mardi Susiki Nugroho, ST., MT.
NIP. 197003131995121001

Halaman ini sengaja dikosongkan

ABSTRAK

Nama Mahasiswa	:	Dito Prabowo
Judul Tugas Akhir	:	Pengembangan Sistem Keamanan Berbagi Data PACS berbasis *Blockchain*
Pembimbing	:	1. Reza Fuad Rachmadi, ST., MT., Ph.D. 2. Dr. I Ketut Eddy Purnama, ST., MT.

PACS (*Picture Archiving and Communication System*) mampu menggantikan sistem radiologi konvensional dengan menyimpan data kedalam server, sehingga bisa menggantikan kertas dan menghemat biaya. Kondisi ini memberikan keuntungan, namun juga menimbulkan masalah, yaitu apabila seorang pasien pergi ke rumah sakit lain data hasil pemeriksaan sebelumnya tidak tersedia, padahal data pemeriksaan tersebut sangat penting. Dibutuhkan sistem yang mampu mengintegrasikan *record* data pasien antar rumahsakit dengan tetap menjaga privasi dan keamanan dari data tersebut. Kami memberikan solusi dengan pembuatan sistem keamanan berbagi *record* data PACS pasien dengan platform *blockchain* dikombinasikan dengan IPFS serta skema *proxy re-encryption*. Data pasien dalam PACS server akan dilihat oleh dokter dan dilakukan analisis, hasil dari analisis beserta data DICOM akan di enkripsi dan disimpan dalam IPFS. IPFS memberikan sebuah *hash* penanda dari data tersebut, *hash* inilah yang dibuat *block* dan disalurkan ke seluruh *node*. Semua rumah sakit yang terhubung dalam jaringan akan menerima data tersebut, tetapi hanya rumah sakit yang diijinkan oleh pemilik data yang bisa melihat karena adanya proses dekripsi data menggunakan *secret key* dari pasien. Implementasinya, sistem berjalan sesuai dengan yang diharapkan, hanya rumah sakit yang mendapat ijin dari pasien yang bisa melihat data pasien, dengan ini sistem sudah memenuhi prinsip keamanan *Confidentiality* (rahasia), *Integrity* (tidak bisa di rubah), *Availability* (dapat diakses kapanpun), konsisten dan juga transparant tanpa adanya pihak ketiga.

Kata Kunci: PACS, Blockchain, *Smart Contract*, *Distributed Network*.

Halaman ini sengaja dikosongkan

ABSTRACT

Name	:	Dito Prabowo
Title	:	*Development of a Blockchain-based PACS Data Sharing Security System*
Advisors	:	1. Reza Fuad Rachmadi, ST., MT., Ph.D. 2. Dr. I Ketut Eddy Purnama, ST., MT.

PACS (Picture Archiving and Communication System) is able to replace conventional radiology systems by storing data into a server, so that it can replace paper and save costs. This condition provides benefits, but also causes problems, that is if a patient goes to another hospital the data from the previous examination is not available, even though the examination data is very important. A system is needed that is able to integrate patient data between hospitals while maintaining the privacy and security of that data. We provide a solution by creating a PACS data sharing security system with a blockchain platform combined with IPFS and Proxy Re-Encryption schemes. Patient data in the PACS server will be seen by the doctor and analyzed, the results of the analysis along with the DICOM file will be encrypted and stored in IPFS. IPFS assigns a hash identifier to the file, this hash is created and distributed to all nodes. All hospitals that are connected in the network will receive the data, but only hospitals that are allowed by the data owner can see because of the process of decrypting the data using the secret key from the patient. Implementation, the system runs as expected, only hospitals that get permission from patients can see patient data, with this system meets the principles of Confidentiality security (confidential), Integrity (can not be changed), Availability (accessible at any time), Consistent and transparent with no third parties.

Keywords:PACS, Blockchain, *Smart Contract*, *Distributed Network.*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer, FTEIC-ITS serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Bapak Dr. Supeno Mardi Susiki Nugroho, ST., MT. selaku Kepala Departemen Teknik Komputer, Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember.
3. Secara khusus penulis mengucapkan terima kasih yang sebesar-besarnya kepada Reza Fuad Rachmadi, ST., MT., Ph.D. dan Bapak Dr. I Ketut Eddy Purnama, ST., MT. atas bimbingan selama pengerjaan tugas akhir ini.
4. Teman-teman Laboratorium Telematika B201-*crew* dan Laboratorium Telematika B401-*crew* yang tidak dapat saya sebutkan satu-persatu.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, 31 Juli 2020

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar belakang

Unit radiologi memiliki peran penting dalam pelayanan pasien baik sebagai pendiagnosa penyakit pasien ataupun menjadi acuan pemberi arah penanganan bagi pasien dalam sebuah rumah sakit. Untuk saat ini kebanyakan rumah sakit masih menggunakan sistem konvesional. Sudah ada sistem yang mampu menggantikan sistem radiologi konvensional yaitu PACS (*Picture Archiving and Communication sistem*) dengan format data DICOM (*Digital Imaging and Communications in Medicine*). PACS menggantikan kertas dengan langsung menyimpannya kedalam server [13].

Digunakannya PACS dalam rumah sakit dapat memberikan keuntungan dalam hal biaya, namun juga terdapat masalah, yaitu apabila ada seorang pasien melakukan pemeriksaan pada rumah sakit lain, *record* data pasien sebelumnya tidak dapat dilihat karena belum adanya sistem yang mengintegrasikan *record* data pasien, juga apabila dibuat sistem tersebut haruslah aman karena data pasien bersifat sensitif. Penyalahgunaan data data pasien bisa saja dilakukan untuk tindakan kriminal, kepentingan politik, kepentingan pribadi yang menjatuhkan orang lain.

Didukung dengan keadaan bisa berbagi *record* data pasien antar rumah sakit merupakan salah satu *future trend* dalam bidang kesehatan. Sebuah survey yang dilakukan di beberapa rumah sakit kuwait, 100 % *radiologist* sangat setuju untuk menggunakan internet untuk bertukar data gambar rekam medis PACS dengan rumah sakit lain. *Record* data PACS bersifat elektronik yang artinya data tersebut bisa dengan mudah disalurkan ke sistem lainnya[14].

Diperlukan sebuah sistem yang bisa berbagi *record* data pasien dengan integritas yang tinggi. Kebanyakan sistem yang ada menggunakan sistem terpusat yang masih mempunyai kelemahan dalam integritas sebuah data. Maka dari itu, kami memberikan solusi dengan menggunakan sistem *blockchain*. *Blockchain* mampu menyediakan integritas yang tinggi dengan sistemnya yang bersifat desentralisasi, dapat diakses secara global oleh siapa saja, memve-

rifikasi data dan konten yang tersimpan, dan setiap transaksinya transparan jadi bisa dilacak[15].

Terlebih lagi dengan adanya *Ethereum*, salah satu platorm *blockchain* yang mempunyai fitur *smart contract*. Begitu *smart contract* dicatat dalam jaringan *blockchain*, kontrak tersebut tidak dapat diubah. Itu sebabnya disebut transaksi tanpa kepercayaan. Tidak perlu mempercayai individu di jaringan *blockchain* tersebut, karena jika kondisi kontrak tidak dipenuhi, maka transaksi tidak akan terjadi.

Penggunaan *blockchain* untuk penyimpanan data, apalagi data yang besar akan menghabiskan biaya yang mahal dan berat. Untuk efisiensi penyimpanan data yang besar, digunakan IPFS (*Inter-Planetary File System*), yang merupakan sistem yang terdistribusi, desentralisasi, sebuah platform untuk menyimpan data dengan integritas yang tinggi. Dasarnya, IPFS adalah *file system* yang *peer to peer*, *distributed file system* yang digunakan untuk menyimpan data dan berbagi data yang besar. Menggunakan *distributed hash table* (DHT) untuk menerima lokasi data dan informasi konektivitas *node* yang terhubung[2].

Skema PRE (*Proxy re-Encryption*) digunakan dalam hal kerahasiaan data. Suatu data yang di enkripsi dengan public key di enkripsi kembali dengan menggunakan *private key* sebelumnya dan *public key* yang baru, sehingga data tersebut bisa di dekripsi dengan *private key* yang baru. Hal ini bisa membagikan akses dengan cara membuat *re-Encryption key* dan *private key* lalu membagikannya dengan pihak yang ingin dibagikan akses nya.

Pengkombinasian antara *ethereum blockchain*, IPFS, skema PRE menjadi sebuah sistem keamanan berbagi data pasien diharapkan menjadi sebuah solusi yang memiliki prinsip keamanan *Confidentiality* (rahasia), *Integrity* (tidak bisa di rubah), *Availability* (dapat diakses kapanpun), konsisten dan juga transparant tanpa adanya pihak ketiga.

1.2 Permasalahan

Berdasarkan dari latar belakang diatas , maka adapun permasalahan yang dapat diambil adalah belum adanya sistem yang bisa mengintegrasikan *record* data PACS pasien antar rumah sakit yang aman, transparan dan terpercaya.

1.3 Tujuan

Penelitian ini memiliki tujuan untuk membuat sistem yang mengintegrasikan *record* data pasien terhadap semua rumah sakit yang tersedia menggunakan platfrom *blockchain*, agar pasien bisa mendapatkan datanya di rumah sakit manapun dengan aman dan privasi tetap terjaga.

1.4 Batasan masalah

Terdapat batasan batasan masalah yang dicapai dalam penelitian ini. Batasan-batasan tersebut diantara lain:

1. Jenis data yang dibagikan oleh pasien seragam berupa keterangan pasien dan data DICOM, jenis data lain akan ditolak pada sistem.
2. Dari 3 jenis data DICOM yang dimungkinkan untuk dibagikan yaitu *single file* DICOM, *Multiple file* DICOM, dan zip DICOM, sistem ini hanya berbagi data dengan jenis *single file* DICOM.
3. Untuk melakukan transaksi dalam *quorum*, diperlukan jumlah minimal *node* sebanyak 3 *node* dalam jaringan.
4. Sistem ini dibuat dengan batasan aplikasi untuk rumah sakit dan pasien, aplikasi untuk *health insurance* dan laboratorium belum disertakan, namun bisa dibuat untuk pengembangan.

1.5 Sistematika Penulisan

Laporan penelitian ini disusun secara sistematis dan memiliki struktur yang mudah dipahami oleh pembaca maupun seseorang yang hendak meneruskan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :

1. BAB I Pendahuluan
   Bab ini berisi uraian tentang latar belakang, permasalahan, tujuan penelitian, batasan masalah dan sistematika laporan.
2. BAB II Teori Penunjang dan Studi Literatur
   Bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu sebagai informasi terkait Blockchain dan penunjang

lainnya.

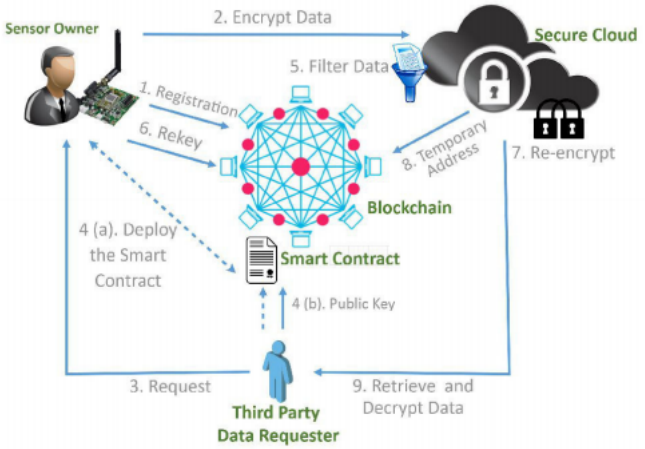
3. BAB III Perancangan Sistem dan Impementasi
   Bab ini berisi tentang penjelasan-penjelasan terkait penelitian yang akan dilakukan dan langkah-langkah data diolah hingga menghasilkan visualisasi. Untuk mendukung penelitian ini, digunakanlah blok diagram agar sistem dapat mudah dibaca untuk implementasi pada pelaksanaan penelitian.
4. BAB IV Pengujian dan Analisa
   Bab ini menjelaskan tentang pengujian yang dilakukan terhadap data dan analisisnya. Beberapa teknik visualisasi akan ditunjukan hasilnya pada bab ini dan dilakukan analisis terhadap hasil visualisasi dan informasi yang didapat dari hasil mengamati visualisasi yang tersaji .
5. BAB V Penutup
   Bab ini berisi kesimpulan yang diambil dari penelitian maupun pengujian yang dilakukan serta saran yang membangun untuk mengembangkan penelitian lebih lanjut dituliskan pada bab ini.

BAB 2
TINJAUAN PUSTAKA

2.1 Related Work

2.1.1 Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing

Penulis pada [1] menggunakan *blockchain* dengan *proxy re-encryption* untuk mengamankan berbagi data IoT. PRE digunakan untuk mengamankan data data sensor yang ingin dibagikan. Jadi setiap orang yang telah dibagikan akses bisa melihat data data tersebut. Untuk penyimpanan menggunakan penyimpanan terpusat. Digunakan untuk menyimpan pesan yang telah di enkripsi. Arsitektur yang diajukan oleh penulis dapat dilihat pada Gambar 2.1



Gambar 2.1: Arsitektur yang diajukan paper[1]

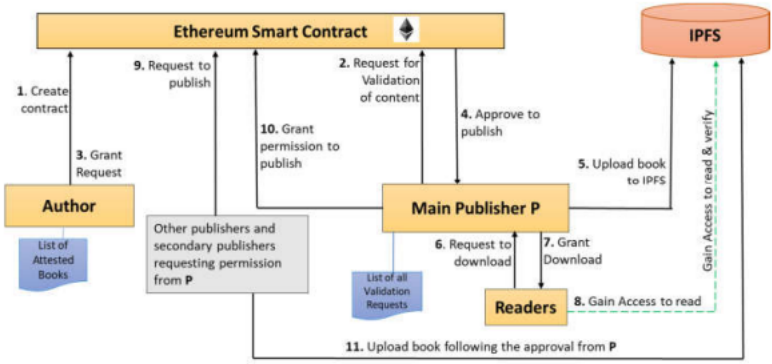
Ketika orang atau kelompok yang meminta data ke *blockchain*, pemilik sensor dan yang meminta melakukan perjanjian pada *smart contract*. Orang yang meminta mengirimkan *public key* nya untuk semua keperluan transaksi. *Cloud server* melakukan dekripsi

dan melakukan enkripsi dengan *public key* orang yang meminta dan data di taruh dalam penyimpanan sementara. Peminta akan mendapat notifikasi apabila data tersedia, dan bisa melakukan deskripsi dengan *private key* nya sendiri.

Penggunaan PRE pada penelitian ini termotivasi dari paper [1] tetapi tidak menggunakan penyimpanan terpusat karena dibutuhkan komputasi yang besar dan apabila penyimpanan tersebut teretas memang benar peretas tidak bisa membaca data yang telah terenkripsi tapi ada kemungkinan data bisa hilang dengan mudah.

2.1.2 IPFS-Blockchain based Authenticity of Online Publications

Penulis pada [2] menggunakan kombinasi IPFS dan *blockchain* untuk memecahkan solusi Autentikasi dan originalitas dari sebuah konten digital yang di posting gratis dalam internet. Penulis pada paper ini menunjukan bagaimana dapat memecahkan masalah untuk buku yang dipublikasikan daring, tetapi solusinya dapat dikembangkan untuk konten multimedia digital lainnya. Penulis juga menunjukan solusinya memiliki kemampuan untuk *tracing* dan *tracking* konten digital yang berbeda versi publisher, yang disetujui oleh penulis asli[2]. Arsitektur sistem pada paper ini dapat dilihat pada Gambar 2.2



Gambar 2.2: Arsitektur yang diajukan paper[2]

Penulis awalnya menulis buku itu, menciptakan kontrak yang

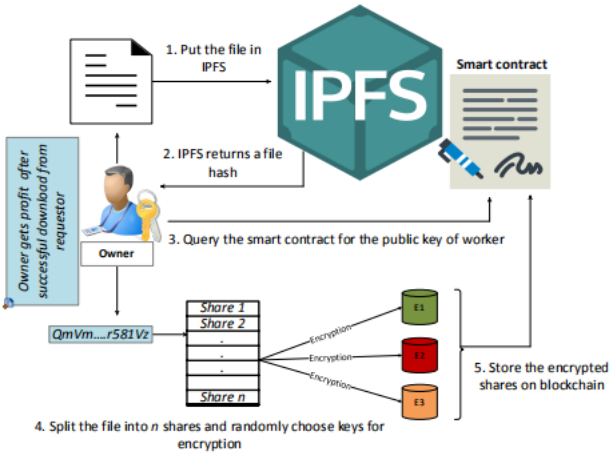
mencakup atribut kunci tentang buku untuk memasukkan judul buku, *hash* asli detail buku dan penulis. *Offline*, penerbit utama meminta izin penerbitan dari penulis setelah menyelesaikan persyaratan penerbitan. Penerbit lalu meminta persetujuan dari penulis asli sebelum mengunggah buku ke IPFS, yang mengembalikan *hash*. Penulis kemudian memeriksa file IPFS yang disimpan (dengan *hash* IPFS yang diberikan) ke aslinya konten yang dikirimkan ke penerbit utama. Penulis menyimpulkan bahwa naskah digital tidak rusak dan membuktikan salinan penerbit. Penulis punya daftar buku, dan juga penerbit memiliki daftar versi buku yang dibuktikan oleh mereka[2].

2.1.3 A Secure Data Sharing Platform Using Blockchain and Interplanetary File System

Untuk [3] juga menggunakan IPFS sebagai penyimpanan, data di enkripsi dan disimpan pada IPFS juga,menggunakan metode enkripsi yang hampir sama namun ada perbedaan, seteleh data di taruh pada IPFS, akan dikembalikan sebuah *hash* penanda dari file tersebut. Setelah mendapat *hash*, pemilik melakukan *query* ke *smart contract* untuk mendapatkan *public key* worker. Data di enkripsi dengan *public key worker* dan disimpan pada IPFS, nantinya hanya *worker* yang mendapat ijin atau yang dipilih yang bisa mendekripsi dengan *private key* miliknya, jadi pemilik memberikan akses dengan cara mengenkripsi data dengan *public key request*. Apabila menggunakan metode ini pastinya setiap *request* pada data yang sama akan menghasilkan pesan enkripsi yang berbeda. Akan ada data yang sama pada IPFS namun berbeda karena di enkripsi menggunakan *public key* yang berbeda beda.

2.1.4 Distributed Offchain Storage of Patient Diagnostic Reports in Healthcare System using IPFS and Blockchain

Pada [4] penulis juga sama seperti kami melakukan berbagi data *patient report* menggunakan *blockchain* dan menyimpan pada IPFS, tetapi berbeda pada proses autentikasi nya. Penulis tidak melakukan proses enkripsi pada data, setiap *node* yang ingin bergabung harus melakukan *proof of identity*. Apabila Sudah tergabung

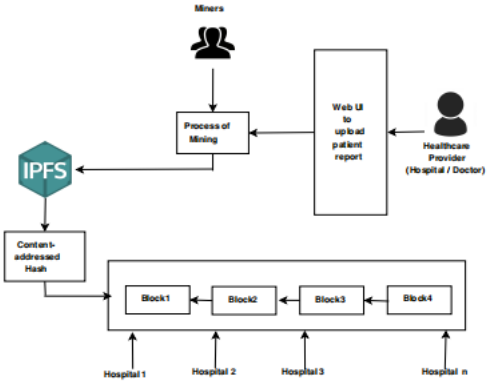


Gambar 2.3: Arsitektur yang diajukan paper[3]

dengan node lainnya, bisa meminta untuk melihat data pada *blockchain*. Dan semua *node* yang tergabung bisa saling melihat dan membuat *record* untuk pasien tanpa adanya proses enkripsi. Pada penelitian ini, *hash* dari IPFS yaitu sebuah *hash* penanda yang digunakan untuk menjaga privasi dari pasien. Apabila tidak mengetahui *hash* penanda tersebut maka tidak bisa melihat atau mengambil data pasien.

2.1.5 Blockchain-Based, Decentralized Access Control for IPFS

Pada paper [16], penulis membuat modifikasi IPFS untuk data yang besar pada *blockchain*. Karena data yang besar tidak bisa efisien untuk disimpan pada *blockchain*. Dan karena *blockchain* mereplika ke banyak *node*, akan membutuhkan penyimpanan yang besar tanpa tujuan yang penting, terlebih apabila pengguna tidak perlu melihat seluruh data yang tersimpan pada *blockchain*. Dibuatlah modifikasi IPFS dengan ACL(*Access Control list*) untuk setiap *node* yang ingin bertransaksi dan menyimpan data nya pada IPFS.



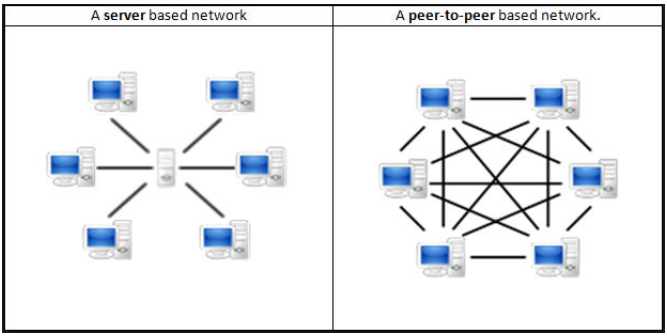
Gambar 2.4: Arsitektur yang diajukan paper[4]

2.2 Teori Penunjang

2.2.1 Jaringan Terdistribusi (Desentralisasi)

Jaringan Terdistribusi adalah sebuah jaringan dimana setiap *node* dalam *cluster* tersebut saling terhubung satu sama lain. Jaringan desentralisasi mendistribusikan data, informasi, dan memproses beban kerja di seluruh komputer yang berpartisipasi dalam jaringan. Hal ini memungkinkan untuk toleransi kesalahan yang lebih besar sebagai komponen kunci dari jaringan di distribusikan di beberapa mesin, jika salah satu jaringan (*hardware*) *down*, maka jaringan secara keseluruhan terus berfungsi [17].

Jaringan terdesentralisasi memiliki keuntungan lebih aman daripada jaringan terpusat, tidak ada satu server pusat yang ditargetkan oleh penyerang. Jika seseorang ingin menyerang jaringan, mereka perlu untuk menyerang beberapa node dalam jaringan. Karena tidak ada kekuasaan dalam jaringan, pengguna bisa lebih mempercayai sistem, namun juga karena tidak ada kekuasaan dalam jaringan, apabila ada masalah dalam jaringan tidak ada otoritas pusat yang membantu menyelesaikan masalah hal ini yang membedakan antara jaringan terpusat dan jaringan terdistribusi sebagaimana dapat dilihat pada Gambar 2.5.



Gambar 2.5: Ilustrasi Perbedaan Jaringan Terpusat dan Jaringan Terdesentralisasi[5]

2.2.2 *Blockchain*

Blockchain adalah database terdistribusi yang digunakan untuk menangani *record* data yang terus bertambah, *record* data ini disebut dengan *block*. Setiap *block* memiliki penanda waktu dan kode unik yang terhubung dengan *block* sebelumnya, sehingga masing masing *block* tersebut saling terhubung satu sama lainnya dan tidak bisa untuk diubah. *Blockchain* biasanya dikelola oleh jaringan *peer-to-peer* yang secara kolektif mengikuti protokol untuk memvalidasi *block* baru. Jika terdapat perintah penambahan *block* baru, maka setiap *node* pada jaringan *peer-to-peer* tersebut akan terlebih dahulu memvalidasi *block* dan kemudian seluruh *node* akan memperbarui *record* data miliknya [18]. *Blockchain* memiliki 3 type [19] yaitu *Public Blockchain* yang dikembangkan secara bersama-sama oleh publik dan siapa saja dapat ikut serta untuk mengembangkan *blockchain*, *Private Blockchain* yang hanya bisa digunakan oleh organisasi tertentu, *consortium blockchain* yang dikembangkan oleh suatu kelompok secara bersama untuk kepentingan tertentu, sederhananya *blockchain* konsorsium merupakan *blockchain* privat yang diberdayakan oleh lebih dari satu kelompok.

Kelebihan *Blockchain*

Berikut merupakan kelebihan dari teknologi *blockchain* [8] :

- Transparansi atau keterbukaan, dalam Blockchain menerapk-

an sistem yang transparan supaya proses yang ada di dalamnya dapat dilihat dan dibagikan kepada semua orang.

- Kekal atau tetap, karena hanya terjadi sekali penulisan data pada *blockchain* dan apabila data tersebut diubah, akan sangat susah sekali dan hampir tidak mungkin untuk mengubah semua data yang telah tersimpan pada *blockchain.* Sebab data yang akan diubah akan mempengaruhi catatan transaksi setelahnya, sehingga dengan mengubah sebuah data, diperlukan upaya untuk mengubah hampir seluruh rekaman data yang telah ada.

- Memiliki sistem keamanan yang kuat dengan menerapkan kriptografi seperti fungsi *hash* untuk memverifikasi dan menjaga integritas data pada setiap *block* sehingga valid serta mencegah dari adanya perubahan data.

- Memiliki kemudahan dalam melacak setiap data transaksi pada jaringan *blockchain*, karena data transaksi yang disimpan pada jaringan *blockchain* tentu akan merujuk pada transaksi sebelumnya, sehingga hal ini dapat mempermudah dalam proses verifikasi dan pencarian data transaksi.

- Bersifat *anonymous*, meskipun data yang disimpan pada jaringan publik *blockchain* bersifat transaparan atau dapat dilihat oleh orang lain. Namun, terkait dengan identitas setiap pengguna yang mengirimkan maupun menerima transaksi dalam jaringan *blockchain* menggunakan suatu alamat tertentu atau yang disbeut dengan *public key*, dan dalam hal ini, identitas sebenarnya dari setiap pengguna tidak ditampilkan pada interaksi transaksi tersebut.

Kelemahan *Blockchain*

Blockchain juga memiliki kelemahan [20] yaitu :

- Kurang Efisien – Konfirmasi *block* data yang terjadi di jaringan blockchain membutuhkan kesepakatan dari pihak-pihak di jaringan blockchain tersebut, sehingga tentunya membutuhkan waktu yang lebih lama dibandingkan dengan adanya oto-

ritas utama. Jika ada otoritas utama mungkin hanya membutuhkan hitungan detik, namun jaringan blockchain membutuhkan waktu 1 menit, 10 menit, atau bahkan sampai hitungan hari, tergantung kecepatan dan kapasitas jaringan tersebut.

- Sulit untuk diatur – Sifat yang terdesentralisasi membuat pihak-pihak yang berada di jaringan blockchain tersebar di berbagai belahan dunia dan tentunya merupakan pihak yang berbeda-beda. Dikarenakan hal ini, jika jaringan tersebut membutuhkan pembaharuan atau pengembangan teknologi, maka pembaharuan tersebut tidak bisa terjadi secara keseluruhan, tergantung pihak-pihak tersebut ingin memperbaharui atau tidak, hal ini berkaitan dengan soft fork dan hard fork dari suatu jaringan blockchain.

Perbedaan *Blockchain* dan Database

Berikut merupakan penjelasan mengenai perbedaan *blockchain* dan database biasa secara umum [21].

Database (Basis data) tradisional adalah struktur data yang digunakan untuk menyimpan informasi. Ini termasuk data yang dapat diminta untuk mengumpulkan wawasan untuk pelaporan terstruktur yang digunakan oleh entitas untuk mendukung keputusan bisnis, keuangan dan manajemen. Pemerintah juga menggunakan basis data untuk menyimpan set data besar yang berskala jutaan catatan.

Database dimulai sebagai sistem hierarki data datar yang menyediakan pengumpulan dan penyimpanan informasi sederhana. Kemudian, basis data menggunakan model relasional yang memungkinkan cara pengumpulan data yang lebih kompleks dengan menghubungkan informasi dari banyak basis data. Informasi yang disimpan dalam basis data dapat diatur menggunakan sistem manajemen basis data. Database sederhana disimpan dalam elemen data yang disebut tabel. Tabel berisi bidang, yang menentukan jenis catatan, yang menyimpan data yang disebut atribut. Setiap bidang berisi kolom yang menggambarkan bidang dan baris yang menentukan catatan yang disimpan dalam database.

Database biasa atau basis data dapat dimodifikasi, dikelola dan dikendalikan oleh satu pengguna yang disebut administrator.

Basis data selalu memiliki pengguna yang berfungsi sebagai admin DB dan pengguna itu memiliki kontrol penuh terhadap basis data. Pengguna ini dapat membuat, menghapus, memodifikasi, dan mengubah catatan apa pun yang disimpan dalam database. Mereka juga dapat melakukan administrasi pada basis data seperti mengoptimalkan kinerja dan mengelola ukurannya ke tingkat yang lebih mudah dikelola. Basis data besar cenderung memperlambat kinerja, sehingga admin dapat menjalankan metode pengoptimalan untuk meningkatkan kinerja.

Sedangkan *blockchain* menyimpan informasi dalam *block* berukuran seragam. Setiap *block* berisi informasi *hash* dari *block* sebelumnya untuk memberikan keamanan kriptografi. *Hash* menggunakan SHA256 yang merupakan fungsi *hash* satu arah. Informasi *hash* ini adalah data dan tanda tangan digital dari *block* sebelumnya, dan *hash* dari *block* sebelumnya yang kembali ke *block* pertama yang diproduksi di *blockchain* yang disebut "*block genesis*". Informasi itu dijalankan melalui fungsi *hash* yang kemudian menunjuk ke alamat *block* sebelumnya. Struktur data *blockchain* adalah contoh *Merkle Tree*, yang digunakan sebagai cara yang efisien untuk memverifikasi data.

2.2.3 Algoritma *Hash*

Fungsi *hash* kriptografi berbeda dari fungsi *hash* biasa, tapi memiliki kesamaan. Di dalam keduanya, ingin mengkompres data menjadi bilangan kecil. Dalam *hash* kriptografi, ingin agar:

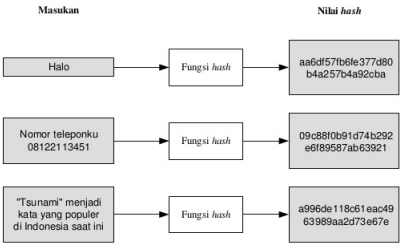
1. Jika data diubah sedikit saja, *hash* nya akan berbeda.

2. Dari sebuah nilai *hash* X, tidak bisa membentuk sebuah input sedemikian sehingga *hash* nya adalah X

Jika tiap kali diberikan sebuah *hash* X dan selalu bisa menemukan sebuah string sedemikian hingga hashnya X tertentu maka fungsi *hash* tersebut berhasi dipecahkan (*broken*). Fungsi *hash* kriptografi biasanya minimal memiliki output 128 bit. Artinya jika mencoba 340282366920938463463374607431768211456 + 1 data yang unik, maka akan ada 1 yang *hash* nya sama.

Saat ini ada banyak fungsi *hash* dan biasanya akan menghasilkan bilangan biner antara 128-512 bit, atau jika dijadikan karakter

heksa: antara 32 - 128 karakter heksa. Beberapa fungsi *hash* yang umum adalah MD5, SHA1, SHA256, SHA512 [22]. Untuk contoh Fungsi *hash* bisa dilihat pada gambar 2.6



Gambar 2.6: Contoh Fungsi *Hash*[6]

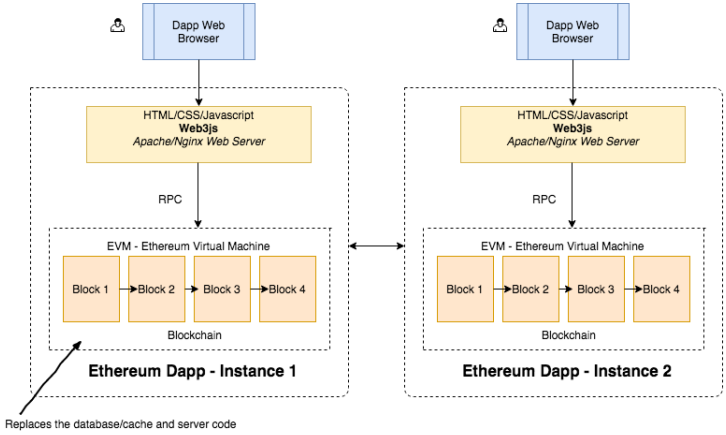
2.2.4 *Quorum*

Blockchain platform yang digunakan pada penelitian ini adalah *quorum*, *quorum* adalah *fork* dari main *ethereum*, sebuah open source *blockchain* platform yang mengkombinasikan inovasi dari komunitas publik *ethereum* dengan tambahan yang dibutuhkan dalam dunia bisnis. Seperti yang dijelaskan pada [23], dalam *ethereum* ada mata uang internal bernama *ether*. Untuk mendeploy atau melakukan transaksi dengan *smart contract*, diperlukan biaya *ether* untuk *miners*. Karena *quorum* adalah *fork* dari ethereum, sama halnya. Tapi dalam *quorum*, *ether* bisa saja tidak berarti karena bisa saja membuat seluruh transaksi tidak memerlukan biaya apabila merancang jaringan private, sehingga bisa fokus pada pengembangan aplikasi desentralisasi. Namun apabila *smart contract* dan aplikasi menggunakan jaringan publik *ethereum* maka transaksi akan dikenakan biaya. Untuk membuat akun, hanya memerlukan *asymmetric key pair*. *Ethereum* menggunakan *Elliptic Curve Cryptography* (ECC) dengan *secp256k1 curve*.

Transaksi dalam *Ethereum* adalah data paket untuk mentransfer *ether* antar akun atau *contract*, memanggil fungsi *contract*, atau

mendeploy *contract* baru. Transaksi dalam *ethereum* menggunakan *Elliptic Curve Digital Signature Algorithm (ECDSA)*, yang mana merupakan *digital signature algorithm* dalam ECC. Dalam transaksi, jumlah maximum dari *computational step* transaksi yang dilakukan dinamakan *gas limit*, biaya pengirim yang melakukan transaksi untuk setiap *computational step* dinamakan *gas price*. Hasil produk dari *gas* yang digunakan dan *gas* price dinamakan *transaction fees*[23].

Secara komponennya *ethereum* memiliki dua komponen penting yaitu prosesor virtual turing (*Turing-complete virtual processor*) yang disebut sebagai *Ethereum Virtual Machine* (EVM) yang memungkinkan prosesor turing dalam *ethereum* tersebut menjalankan script atau bahasa pemrograman yang disebut Solidity untuk membangun aplikasi terdesentralisasi dan juga nilai token yang disebut dengan *ether*, supaya dapat digunakan sebagai mata uang atau *cryptocurrency* yang disahkan oleh jaringan untuk melakukan transaksi antar pengguna atau sebagai kompensasi bagi para *miner* [8]. Untuk lebih jelas mengenai arsitektur aplikasi desentralisasi *ethereum* bisa dilihat gambar 2.7.



Gambar 2.7: Architecture Ethereum Decentralized App[7]

2.2.5 *Raft Consensus Algorithm*

Quorum mendukung beberapa protokol konsensus. Pada penelitian kali ini menggunakan *Raft* konsensus. Setiap *node* dalam jaringan menyimpan daftar semua *node* lain terlepas aktif atau tidak. Setiap *node* bisa menjadi pemimpin atau pengikut. Hanya ada satu pemimpin dalam jaringan. Pemimpin bertanggung jawab untuk membuat dan mengirim *block* ke pengikut.Secara teratur mengirimkan informasi kepada pengikut aktif tidaknya pemimpin. setiap pengikut mengharapkan keberadaan pemimpin dengan batas waktu sekitar 150 - 300 ms. jika tidak ada informasi dari pemimpin, pengikut mengubah statusnya menjadi kandidat dan memulai pemilihan untuk memilih pemimpin baru pada jaringan. *Raft* memerlukan lebih dari 50% dari *node* yang tersedia agar transaksi bisa bisa tersimpan dalam *blockchain*. Apabila sebuah *cluster* memiliki 2 * F +1 node, bisa toleransi F kesalahan dan masih berjalan normal. ketika pemimpin membuat *block*, pertama akan mengirimkan *block* ke semua followers, dan apabila lebih dari 50 % pengikut menerima *block*, pemimpin akan *commit block* ke *blockchain* nya sendiri dan mengirimkan pesan kepada semua pengikut sehingga pengikut juga bisa *commit block* dalam blockchain nya[23].

2.2.6 *Smart Contract*

Smart contract merupakan *self executing contract* yang ditulis dalam sebuah kode. Kontrak dapat terlaksana tanpa bantuan pihak ketiga. *Smart contract* besifat *Immutable* dan *Distributed*. *Immutable* artinya apabila kontrak tersebut dibuat dan di *deploy* dalam jaringan *blockchain* tidak akan bisa di rubah, apabila memang perlu untuk mengganti diperlukan reset dan *deploy* ulang kontrak. *Distributed* berarti hasil transaksi dengan kontrak akan di validasi dan diterima oleh semua *node* yang tergabung dalam network, apabila ada satu *node* yang memaksakan hasil yang berbeda maka transaksi tersebut akan *invalid*. Dalam mengimplementasikan *smart contract* ada 3 tahap yang harus dikerjakan, yaitu pembuatan kode *smart contract*, *compile* kode *smart contract*, kemudian *deploy smart contract* pada jaringan *blockchain*.

Pembuatan Kode *Smart Contract*

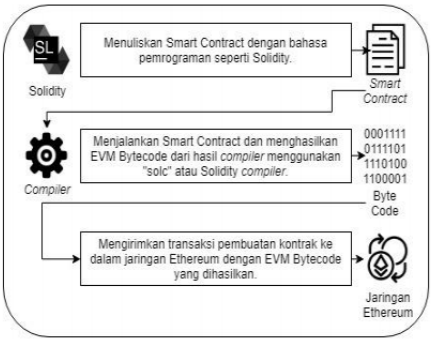
Pembuatan kode kontrak merupakan proses membuat kode sebagai *logic* dari perjanjian atau sistem yang akan dibangun. Pada pembuatan kode kontrak ini, dibuat kode-kode seperti *function modifier*, *function pure* dan *view*, *logging event*, dan fungsi-fungsi lainnya.

Compile Smart Contract

Kode yang telah dibuat dilakukan *compile* menggunakan solidity *compiler*. Pada proses *encoding* dengan *compiler* supaya menghasilkan *bytecode* yang digunakan sebagai referensi fungsi dan kontrak untuk diekseskusi pada EVM. Proses *encoding* tersebut dibantu dengan menggunakan ABI (*Application Binary Interface*) yang merupakan daftar definisi fungsi dalam kontrak dan beberapa argumen yang ditulis dalam format *Javascript Object Notation* (JSON). Daftar fungsi dan argument tersebut diubah dengan *hash* menjadi ABI, kemudian dapat diolah oleh EVM. ABI sangat diperlukan supaya dapat menentukan fungsi mana yang ada pada kontrak untuk dijalankan, serta menjamin fungsi tersebut akan mengembalikan data dalam format yang sudah ditentukan [24].

Deploy Smart Contract

Kontrak perlu di *deploy* agar mendapat suatu alamat yang dapat dikirimkan data sesuai dengan function apa yang dipanggil. Setelah di *deploy*, bisa menggunakan aplikasi yang terhubung ke *node* dengan perantara *web3* dan bisa memanggil function dalam kontrak dan melakukan transaksi karena JSON-RPC yang di *compile* kontrak bisa di *load* oleh aplikasi. Semua fungsi yang ditulis dalam kontrak bisa dipanggil dengan perantara *web3js*. *Web3js* bisa menghubungkan dengan *node* melalui *JavaScript APIs*. Berkomunikasi ke *node* menggunakan JS APIs yang disebut JSON-RPC internal. Kemudian mengubungkan ke *node* dengan memanggil alamat RPC *node* yang telah didapatkan ketika *node* dibuat dan terhubung ke dalam jaringan blockchain. Proses *compile* dan *deploy* bisa dilihat pada gambar 2.8



Gambar 2.8: Proses *Compile* dan *Deploy Smart Contract* [8]

Skema Penulisan Data 2 Arah dalam *Ethereum Smart Contract*

Pada dasarnya dalam *blockchain*, akan sulit untuk melakukan pengolahan data, namun berbeda apabila menggunakan *smart contract* untuk membangun aplikasi desentralisasi. Dengan menggunakan *smart contract*, dimungkinkan untuk dilakukan operasi CRUD pada *blockchain*. Berikut merupakan contoh kode *smart contract* untuk melakukan tambah dan perbarui data.

Dalam *smart contract* (solidity code), dibuat struktur data pasien. Data pasien mempunyai nama, umur, dan *hash* dicom file. Nantinya data pasien akan disimpan dalam *array* Pasiens melalui mapping.

```
struct pasien{
 string name;
 uint256 age;
 string hash_dicom;
 }
 mapping (address => pasien) Pasiens;
```

Dibuat sebuah variable untuk *tracking* dari jumlah pasien dan diberikan nilai 0 pada *contructor* dalam proses inisialisasi. Proses inisialisasi sangat diperlukan untuk menghindari kesalahan atau *error* pada aplikasi.

```
uint256 public total_pasien;
constructor() public {
 total_pasien = 0;
}
```

Dengan struktur data seperti diatas, dapat dibuat sebuah fungsi untuk tambah maupun perbarui data.

```
function insert(string name, uint256 age, string hash_dicomm)
public returns (uint256 total_pasien) {
 pasien memory newpasien = pasien(name , age, hash_dicom);
 pasiens.push(newpasien);
 total_pasien++;
 return total_pasien;
}

function update(string name, uint256 age, string hash_dicom)
public returns (bool success){
for(uint256 i =0; i< total_pasien; i++){
  if(compareStrings(Pasiens[i].name ,name)){
    Pasiens[i].age = age;
    Pasiens[i].hash_dicom = hash_dicom;
    return true;
  }
}
return false;
}
```

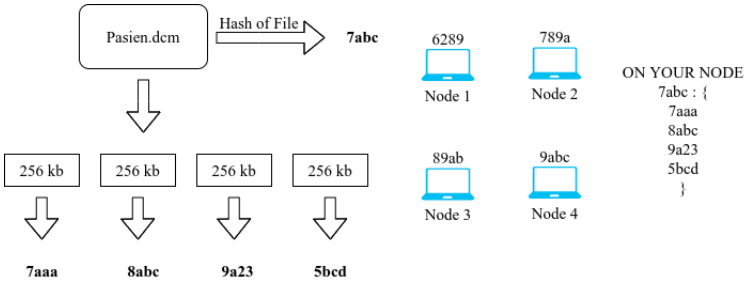
Pada fungsi tambah data memerlukan 3 argumen, argumen tersebut akan dibuat untuk membuat data baru pasien dalam bentuk sebuah *state*. Data pasien dibentuk menggunakan sebuah memori pada solidity dan di *push*, juga dilakukan *increment* pada total pasien. Untuk melakukan update data, juga diperlukan 3 argumen, argumen nama digunakan untuk melakukan pengecekan data, kemudian data yang sesuai akan di update.

Dalam hal ini, *blockchain* mengelola seluruh *state* data. Setiap perubahan akan berpengaruh dalam setiap node yang berpartisipasi dalam network, oleh karena itu setiap pembentukan data maupun perubahan data diperlukan biaya transaksi, namun karena pada penelitian ini menggunakan blockchain type semi-privat yaitu konsorsium *blockchain*, biaya transaksi dapat dihilangkan dan sepenuhnya bisa membuat aplikasi desentralisasi tanpa masalah. Jadi,

sistem dalam penelitian ini bisa untuk mengolah data CRUD (*Create, Read, Update, Delete*) layaknya database konvensional, namun hasil transaksi dari datanya di distribusikan ke seluruh *node* untuk mengatasi masalah keamanan data. Dalam sistem ini persamaannya dengan database konvensional dapat dimungkinkan operasi CRUD, dan perbedaannya hasil dari operasinya disimpan dalam seluruh *node* yang terhubung untuk keamanan datanya. Juga bisa diterapkan *authorized access* dalam *smart contract* dengan token untuk mengatasi siapa saja yang boleh untuk melihat atau menulis data.

2.2.7 *InterPlanetary File System*

IPFS (*InterPlanetary File System*) adalah sebuah protocol dan *peer to peer* jaringan untuk berbagi data dalam *distributed file system*. Cara kerjanya mirip seperti *BitTorrent*, yaitu bisa memperoleh suatu data dari *node node* yang terhubung bukan dari server terpusat. Sistem pada IPFS bergantung pada *distributed hash table* (DHT) untuk memperoleh lokasi data dan informasi konektifitas antar node. Berdasarkan [16], Berikut contoh bagaimana IFPS bekerja. Ketika sebuah data di upload ke dalam IPFS, data tersebut di pecah menjadi 5 bagian, masing masing mengandung setidaknya 256 kilobytes data atau link ke chunk yang lain. Setiap chunk diidentifikasi dengan cryptographic *hash*, yang juga dinamakan *content identifier*, yang telah dikomputasi dari kontennya sendiri. Link yang di sebut juga mempunyai content identifiers, dan dengan demikian akan membentuk *Merkle directed acyclic graph (Merkle DAG)* yang di deskripsikan dapat digunakan untuk reconstruct file apapun dari chunk nya. Karena *Merkle DAG*, keseluruhan data dapat diidentifikasi hanya dengan menggunakan *root hash* nya. Karena IPFS menggunakan *content identifier* untuk identifikasinya, bisa mengirimkan content identifier ini ke dalam *blockchain*, jadi tidak menyimpan sebuah file dalam *blockchain* tetapi menyimpannya dalam IPFS dan melakukan query ke *blockchain* untuk mendapatkan hash nya yang kemudian direkonstruksi menjadi sebuah data. Dan pastinya tidak ada duplikasi serta setiap data tentunya akan memiliki *hash* yang berbeda beda ketika data di submit dalam IPFS. Gambaran bagiamana cara kerja IPFS bisa dilihat pada gambar 2.9 dan table 2.1[25].



Gambar 2.9: Ilustrasi IPFS

Distributed Hash Table	
Chunk	Peer ID
7aaa	6289 (Node 1)
8abc	7892 (Node 2)
9a23	89ab (Node 3)
5bcd	9abc (Node 4)
7abc	On Your Node

Tabel 2.1: *Distributed Hash Table*

2.2.8 *Proxy re-Encryption*

Untuk mengatur perijinan dalam *blockchain* agar bisa membagikan data kepada siapapun yang diinginkan secara aman, kami menggunakan *library* PRE. Pemilik data dapat mendelegasikan hak dekripsi kepada penerima untuk setiap *ciphertext* yang dimaksudkan untuknya, melalui proses enkripsi ulang yang dilakukan. Ketika dilakukan enkripsi ulang, penerima dapat menggabungkan enkripsi ulang independen ini dan mendekripsi pesan asli menggunakan kunci pribadinya. Meskipun metode nya menggunakan *proxy*, tetapi pada penelitian ini tidak menggunakan *proxy*, melainkan menyimpan *Re-Encryption Key* pada token.

Berdasarkan [26] berikut adalah fungsi yang ada dalam module *Proxy re-Encryption* :

- ***KeyGen***: Algoritma key yang menghasilkan *public/private key pair* (*pk, sk*).

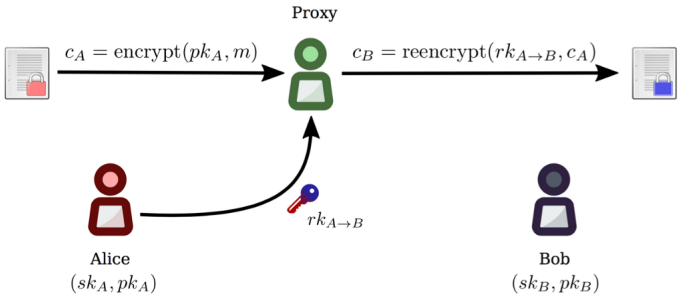
- ***ReKey***: Membutuhkan input *key pair* (pk_i, sk_i) untuk user i dan *keypair* (pk_j, sk_j) untuk user j, algoritma *ReKey* ini dilakukan oleh user i dan akan menghasilkan sebuah *re-encryption key* $rk_{i \to j}$. Dalam case ini, user i bertindak sebagai delegator dan user j yang di delegasi.

- ***Encrypt***: Membutuhkan input pesan plaintext $m \in M$ dan public key pk_i untuk user i, algoritma enkripsi ini menghasilkan output ciphertext $c_i \in C_1$.

- ***ReEncrypt***: Membutuhkan input ciphertext $c_i \in C_1$ untuk user i dan re-encryption key $rk_{i \to j}$ untuk $i \to j$. Menghasilkan ciphertext $c_j \in C_2$ untuk user j atau error symbol $\perp$ yang mengindikasikan c_i invalid.

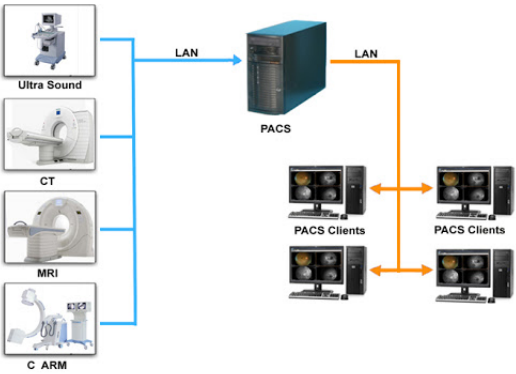
- ***Decrypt***: Membutuhkan private key sk_i atau private key sk_j dan ciphertext c_i atau c_j untuk mendekripsi ciphertext menjadi plaintext message $m \in M$.

Pada Penelitian ini, kami menggunakan semua fungsi kecuali *KeyGen*. Karena menggunakan *key pair*, baik *public key* dan *private key* dari akun yang telah di hasilkan saat membuat *node*. Saat membuat *node* akan mendapat sebuah *private key* untuk melakukan transaksi dan dari *private key* tersebut, bisa mendapatkan *public key* dengan menggunakan modul *ethereum crypto*.



Gambar 2.10: *Proxy re-Encryption*[9]

2.2.9 *Picture Archiving and Communication System*



Gambar 2.11: Ilustrasi Pemanfaatan PACS[10]

PACS (*Picture Archiving and Communication System*) merupakan sistem yang memungkinkan penyimpanan, pengambilan, dan menampilkan sebuah citra medis, lebih khususnya untuk rumah sakit berskala besar dan klinik. Selain itu PACS merupakan sistem dengan konsep *filmless* atau tidak berbentuk fisik dengan metode komputerisasi komunikasi dan dapat menyimpan citra medis seperti computed *radiographic, digital radiographic, computed tomographic, ultrasound, fluoroscopic, magnetic resonance* dan lain-lain. Sebelumnya interpretasi citra medis bergantung pada pengalaman dan riwayat kesehatan pasien untuk memperoleh diagnosis yang tepat. Sebelum citra medis dapat memasuki PACS, diperlukan akuisisi citra yang dilakukan oleh berbagai modality (*computed radiographic,digital radiographic, computed tomographic, ultrasound, fluoroscopic, magnetic resonance* dan lain-lain)[13]. Ilustrasi pemanfaatan PACS dapat dilihat pada gambar 2.11

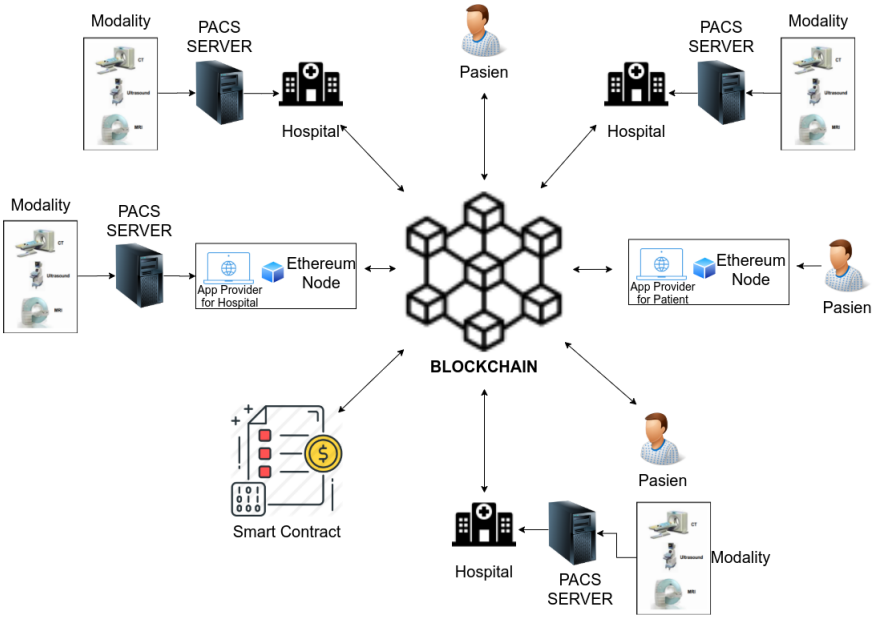
2.2.10 Standar DICOM

DICOM (*Digital Imaging and Communication in Medicine*) merupakan standar yang digunakan dalam PACS untuk komunikasi gambar dan data medis. Standar ini memiliki dua komponen, yai-

tu protokol komunikasi dan format data gambar. Standar DICOM memungkinkan komunikasi digital antara peralatan medis dari tiap vendor yang berbeda. Saat ini semua peralatan sistem pencitraan medis modern (*Imaging Modalities*) seperti Sinar-X, Ultrasound, CT (*Computed Tomography*), dan MRI (*Magnetic Resonance Imaging*) mendukung DICOM dan menggunakannya secara luas. Dengan standar ini, maka praktisi dan vendor dapat dengan mudah bekerja sama untuk membangun sistem medis tanpa adanya kendala[13].

BAB 3
DESAIN DAN IMPLEMENTASI SISTEM

3.1 Desain Sistem



Gambar 3.1: Gambaran Umum Sistem

Penelitian ini memiliki tujuan untuk membuat sistem yang mengintegrasikan *record* data pasien dalam unit radiologi terhadap semua rumah sakit yang tersedia agar pasien bisa mendapatkan datanya di rumah sakit manapun dengan aman dan privasi terjaga, serta pasien bisa melihat *record* datanya sendiri.

Dalam penelitian ini, gambaran umum kerja sistem ditampilkan pada gambar 3.1. Sistem dalam penelitian ini dimulai dari pengembangan PACS sebagai sistem yang memungkinkan penyimpanan, pengambilan, dan menampilkan sebuah citra medis dengan kon-

sep *filmless* atau tidak berbentuk fisik. PACS dikembangkan untuk kebutuhan interpretasi citra medis oleh dokter. Dalam penelitian ini akan diimplementasikan *Orthanc* sebagai PACS. *Orthanc* akan merubah sebuah komputer menjadi sistem PACS, sehingga memiliki kemampuan untuk mengelola seluruh citra medis pasien yang telah diakuisisi, sehingga dapat disimpan, diambil, dan ditampilkan sesuai kebutuhan klinis tanpa adanya kesalahan. PACS sendiri sudah menyediakan sebuah web server sendiri untuk mengambil data dari server, dokter bisa mendapatkan data pasien dan memberikan diagnosa.

Dilanjutkan dengan pembuatan jaringan *blockchain* untuk pasien dan rumah sakit menggunakan *quorum-maker* dan pembuatan *smart contract* kode untuk di *deploy* pada jaringan. Setiap *node* bisa melakukan transaksi dengan *smart contract*, dan setiap *node* juga terhubung dengan aplikasi pasien ataupun rumah sakit. Setiap pasien dan rumah sakit memiliki aplikasi yang berbeda dalam sistem ini. Diagnosa dokter bisa dicatat melalui aplikasi rumah sakit beserta dengan file DICOM dari pasien yang diambil dari PACS, nantinya *record* data ini akan disebarkan ke seluruh rumah sakit dan pasien yang terhubung dalam jaringan *blockchain*. Aplikasi pasien bisa melihat *record* nya sendiri, pada saat pasien pergi ke rumah sakit lain pasien bisa mengijinkan rumah sakit tersebut untuk melihat datanya.

3.1.1 Alur Kerja Pembuatan Data Pasien Pada Sistem

Berikut merupakan alur kerja pembuatan data pasien, pengambilan medical image pada PACS dan proses enkripsi nya.

1. Pasien melakukan pemeriksaan pada rumah sakit, lebih tepatnya pada unit radiology. Disini pasien dapat melakukan berbagai pemeriksaan seperti *MRI, CT , X-ray* dan lainnya.

2. Hasil *scan* dari gambar akan tersimpan pada local *scanner* juga akan tersimpan pada PACS Server.

3. Dokter bisa memberikan analisis untuk data pasien dan mengambil file pasien pada server PACS.

4. Rumah sakit akan mensubmit data pasien dalam aplikasi rumah sakit beserta analisnya ke *blockchain*, apabila sebelumnya belum mendapat ijin, meminta ijin dulu untuk dapat menulis data pasien.

5. Apabila pasien mengijinkan, pasien pada aplikasi menekan tombol *share*, terjadi proses pembuatan token. Token di validasi dan disimpan pada *blockchain* sebagai penanda.

6. Token yang asli diberikan pada rumah sakit sebagai kunci untuk akses.

7. Rumah sakit kemudian melakukan validasi token pada *blockchain*.

8. Akan terdapat *approval* apabila token memang valid.

9. Setelah mendapat akses, rumah sakit akan melakukan submit data pasien.

10. Data pasien akan dienkripsi dengan *public key* pasien.

11. Data pasien yang dienkripsi akan disimpan pada IPFS.

12. IPFS mengembalikan dalam belum bentuk *hash* penanda yang merupakan alamat dari data tersebut.

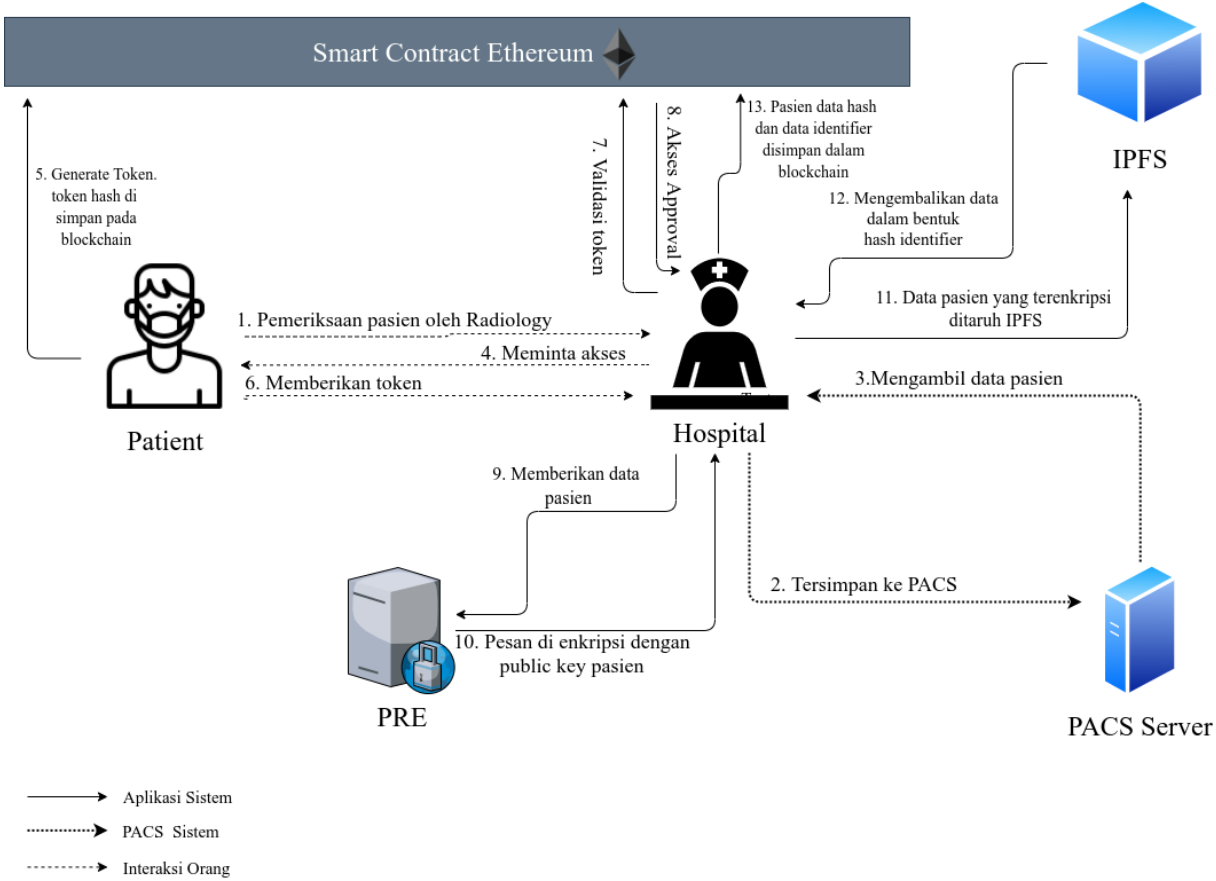
13. Pasien data *hash* dan data penanda disimpan dalam *blockchain*. Disini semua *node* bisa mendapatkan data tersebut apabila memang diijinkan oleh pasien.

Alur kerja dalam gambar bisa dilihat pada Gambar 3.2. Pada gambar 6 ditunjukan hasil token dan *secret key* yang di hasilkan untuk berbagi akses ke rumah sakit.

3.1.2 Alur Kerja Mengambil Data Pasien Pada Sistem

Berikut merupakan alur kerja engambilan data pasien dan bagaimana proses dekripsi nya.

1. Pasien melakukan konsultasi pada rumah sakit lain.



Gambar 3.2: Desain Sistem Pembuatan Data Pasien

2. rumah sakit meminta ijin ke pasien untuk melihat data nya apabila rumah sakit belum pernah mendapatkan ijin dari pasien tersebut.

3. Pasin melakukan share akses pada aplikasi, terjadi proses generate token dan *private key temporary*. Disini pasien tidak membagikan *private key* nya karena akan mengakibatkan akun pasien bisa di gunakan orang lain. *private key* akun dan private key yang temporary akan di laksanakan proses pembuatan *re-encryption*, didapatkan lah *re-key*. token *hash* dan *re-key* disimpan dalam *blockchain*.

4. Pasien memberikan token dan secret key sebagai kunci.

5. Rumah sakit melakukan validasi token dan request data pasien.

6. Apabila token *valid*, rumah sakit bisa mendapatkan hash identifier dan re-key.

7. Rumah sakit melakukan *request* ke IPFS untuk mendapatkan data pasien dengan *hash* penanda.

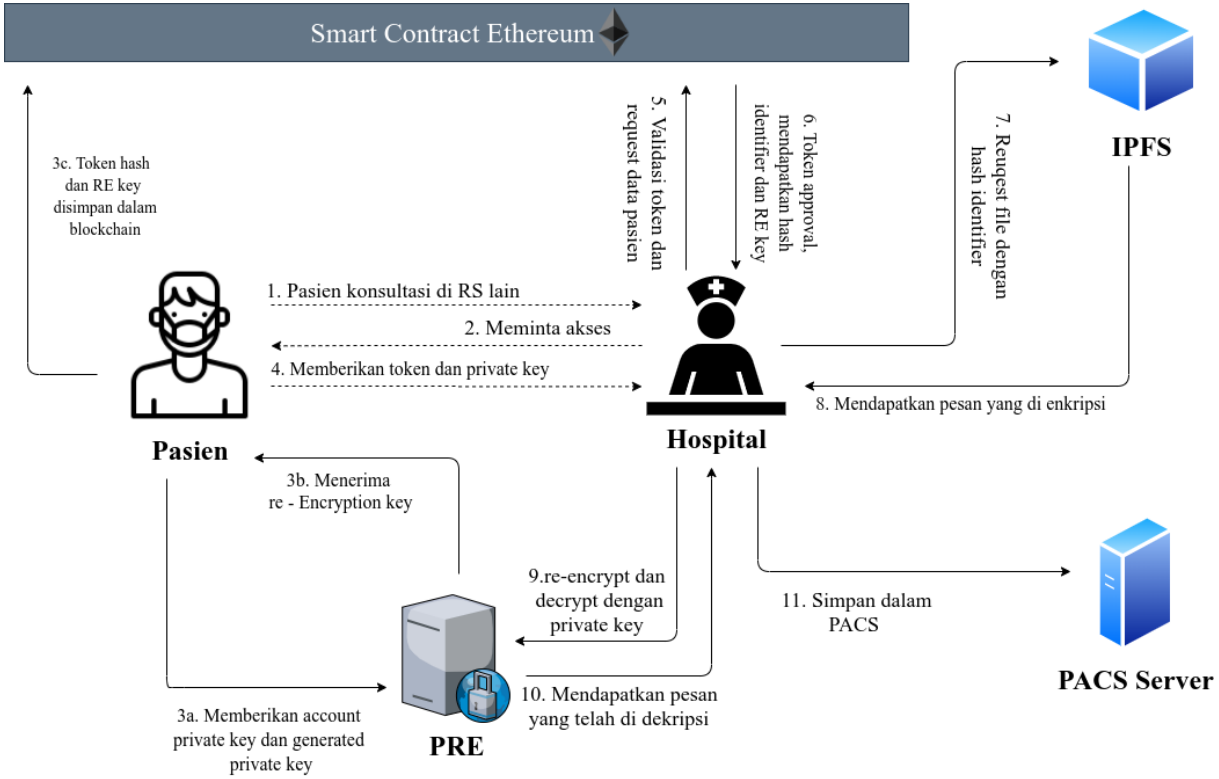
8. Rumah sakit mendapatkan data pasien yang masih terenkripsi.

9. Data pasien akan dilakukan *re encrypt* dengan *re-key* dan dilakukan dekripsi dengan *private key* yang diberikan.

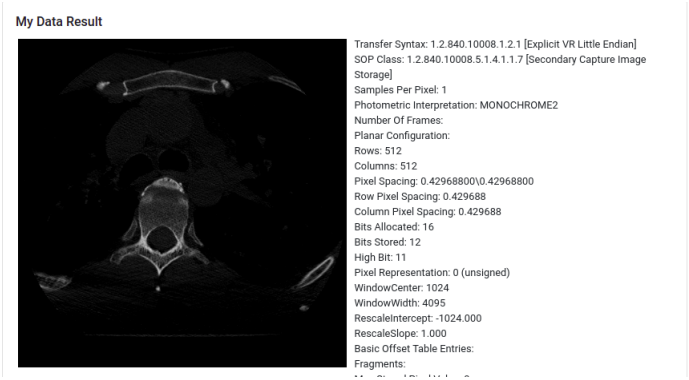
10. Apabila *private key valid*, Rumah sakit bisa membaca data pasien. Bisa preview dengan aplikasi seperti pada gambar 3.4

11. File DICOM yang diperoleh bisa disimpan pada PACS server milik rumah sakit.

Alur kerja pada sistem bisa dilihat pada gambar 3.3.



Gambar 3.3: Desain Sistem Mengambil Data Patient



Gambar 3.4: DICOM *Viewer* Pada Aplikasi

3.2 Implementasi Sistem

Pada penelitian ini terdapat 6 tahap dalam alur mengimplementasikan sistem, yaitu *tools* aplikasi yang digunakan, implementasi *Orthanc* sebagai PACS server, membangun jaringan *blockchain* dengan *quorum network*, pembuatan *smart contract code*, pembuatan aplikasi pasien, dan pembuatan aplikasi rumah sakit.

3.2.1 Tools Aplikasi Yang Digunakan

Sistem ini dibuat pada *operating system* Linux. Untuk dapat berjalan, sistem ini membutuhkan:

1. ***NodeJS* 10**: platform untuk menjalankan kode *JavaScript* melalui sisi server. Digunakan beberapa module dalam *node-js* untuk keperluan sistem, seperti modul *express* untuk web server, *ethereumjs utility* untuk keperluan *node*, *web3js* untuk menghubungkan web server dan *node*, modul IPFS untuk menyambungkan ke *node* IPFS, serta modul *http*, *fileupload*, *body-parser*, *fs*, *childprocess* yang diperlukan untuk kemudahan sistem.

2. ***Docker*** : aplikasi yang bersifat open source yang berfungsi sebagai wadah/container untuk mengepak/memasukkan sebuah software secara lengkap.

3. ***Quorum Maker***: alat yang memungkinkan pengguna untuk membuat dan mengelola jaringan *quorum*.

4. ***Proxy re-Encryption Nuchyper Library***: skema *re-encryption* sebuah pesan yang digunakan untuk berbagi data yang dienkripsi.

5. ***InterPlanetary File System (IPFS)***: protokol dan *peer to peer* network untuk distribusi konten yang cepat, terdistribusi dan mudah disatukan yang kompatibel untuk semua tipe data seperti gambar, stream video, database terdistribusi.

6. ***Orthanc* (untuk rumah sakit)**: aplikasi open source untuk menerima, melihat, menyimpan data citra medis pasien.

Scripting Installation Utility

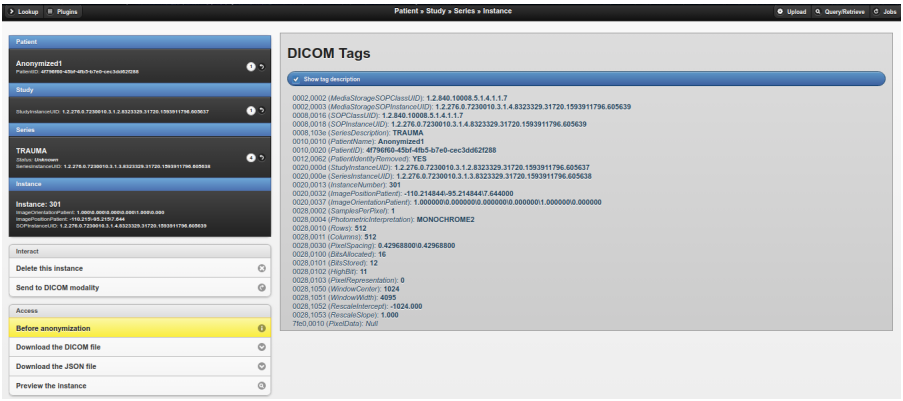
Untuk memudahkan melakukan instalasi aplikasi seperti docker, nodejs, nucypher, IPFS, serta segala utility yang diperlukan untuk menjalankan sistem ini, dilakukan shell scripting instalasi. Script code ketika dijalankan akan langsung menginstall semua utility, jadi hanya tinggal menunggu selesai. Berikut merupakan code script instalasi.

```
#Install :
sudo apt-get update
#Install docker sampai hello world
sudo apt-get install -y \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
    sudo apt-key add -
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu
    \
$(lsb_release -cs) \
stable"

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.
    io
```

```
sudo docker run hello-world

#install nodejs
sudo apt-get install -y build-essential
curl -sL https://deb.nodesource.com/setup_10.x -o
    nodesource_setup.sh
chmod +x nodesource_setup.sh
./nodesource_setup.sh
sudo apt-get install -y nodejs
node -v
#install nucypher
sudo apt install -y python3-pip
sudo apt-get install -y libgmp3-dev
sudo apt-get install -y libssl-dev
pip3 install msgpack
git clone https://github.com/nucypher/nucypher-pre-python.
    git
cd nucypher-pre-python/
sudo python3 setup.py install
#install ipfs
docker pull ipfs/go-ipfs
```

3.2.2 Implementasi Orthanc Sebagai PACS Server

Pada penilitian ini untuk PACS server digunakan *Orthanc* karena open source dan mudah digunakan. Source code *Orthanc* bisa diunduh dengan mudah pada github akun *Orthanc mirror*, dilakukan instalasi dengan *compile* sumber kode pada linux *distro ubuntu 20.04*. Sebelum dilakukan *compile* diperlukan *package utility* untuk mendukung instalasi *Orthanc*. Setelah terpenuhi baru dilakukan *compile* source *Orthanc* dengan *cmake*. Setelah selesai, terdapat program *Orthanc* untuk memulai PACS server. *Orthanc* sudah menyediakan beberapa *plugin*, termasuk web server dan database yang digunakan, karena hal ini sulit untuk di kombinasikan dengan aplikasi rumah sakit, namun dalam web server milik *Orthanc* tetap bisa untuk mengambil data pasien dalam bentuk data DICOM. Dalam web server *Orthanc* juga bisa untuk melakukan *uploads* maupun *Downloads* file DICOM milik pasien apabila mendapat akses server PACS. Web server pada *Orthanc* dapat diakses pada port 8042 (*http://localhost:8042*). Tampilan antarmuka *Orthanc* dapat dilihat pada gambar 3.5.



Gambar 3.5: Tampilan antarmuka Orthanc

Konfigurasi pada *Orthanc*

Pada *Orthanc* terdapat komponen Configuration File yang berguna untuk mengatur akses dari setiap perangkat yang terhubung satu jaringan intranet dengan *Orthanc* untuk mendapatkan akses pada citra medis yang tersimpan. Di dalam jaringan PACS untuk berinteraksi antar perangkat, perlu dilakukan pendaftaran terhadap *IP address, TCP port, dan AE title* dari setiap perangkat yang terhubung dengan Orthanc agar dapat dikenali dan melakukan pertukaran data [13].

3.2.3 *Quorum Network*

Implementasi platform *blockchain* pada penelitian ini adalah dengan menggunakan *quorum*, *fork* dari *main ethereum*. Kami mengimplementasikan quorum network dengan menggunakan *quorum-maker* dengan metode konsensus *RAFT*, sebuah teknologi open source di github dari synechron-finlabs. Dengan menggunakan *quorum-maker*, bisa membangun sebuah *cluster blockchain*. Membuat jaringan baru untuk *production environment*, bergabung dengan jaringan yang sudah ada, membangun *development environtments* untuk mencoba dan *research* yang didasari dengan penggunaan docker agar mudah untuk *deploy quorum network*. *Quorum-maker* juga mempunyai beberapa kelebihan seperti yang dijelasakan dalam gi-

thub nya, diantaranya adalah memiliki tampilan UI dan node manager untuk melihat transaksi yang telah dikerjakan serta menerima notifikasi bergabung melalui *user interface* [11].

Untuk memulai menggunakan privat jaringan quorum, sudah disediakan sebuah *docker image* yang berisi *geth* dan *constellation* yang dibutuhkan untuk membuat *cluster blockchain*. Dengan *docker image* tersebut bisa membuat *container* yang menjalankan sebuah *node*. Sudah disediakan *script* linux untuk memulai setup.

Create Quorum Network

Setelah mempunyai *docker image* yang dibutuhkan, perlu menjalankan *script shell* yang tersedia. Saat memilih untuk membuat jaringan, akan ada beberapa pertanyaan untuk melengkapi konfigurasi *node* tersebut. Mulai dari *node name, ip address*, dan *option port* yang diperlukan, untuk *port* disediakan default *port* yaitu mulai dari 22000 sampai dengan 22004 untuk satu *node*. Disini akan membahas bagaimana proses dalam *script* tersebut sehingga bisa membangun privat *permissioned blockchain*.

Disini sudah mempunyai sebuah container yang berisi *geth* dan *constellation* yang diperlukan untuk membuat *node*. setelahnya, konfigurasi *constellation node* terlebih dahulu, buat *asymmetric key pair*, berikut contoh untuk membuat.

```
./constellation-node --generatekeys=node1
```

Kemudian setelah tercipta *key pairs*, bisa menjalankan *constellation node* tersebut, namun diperlukan data konfigurasi untuk menjalankannya. Data konfigurasi berisi beberapa keterangan seperti *url, port, storage*, letak *socket*,letak *public key* dan *private key* serta setingan tls. Bisa membuat template nya dan menaruhnya pada *docker*, berikut contoh file konfigurasi *constellation*.

```
url = "http://127.0.0.1:9001/"
port = 9001
storage = "dir:./cnode_data/cnode1/"
socket = "./cnode_data/cnode1/constellation_node1.ipc"
publickeys = ["./cnode1.pub"]
privatekeys = ["./cnode1.key"]
tls = "off
```

Langkah berikut setelah konfigurasi *constellation* adalah menggenerate *enodes*, *node ID* agar tiap tiap node bisa saling mengenali satu sama lain. Untuk *node*, bisa membuat static node dan dynamic node. Untuk *static node* bisa membuat file *static-nodes.json* dan menambahkan beberapa *node id* yang telah digenerate. Contohnya: *enode://[nodeid]@[ip]:[port]*. Berikut perintah untuk menggenerate node id dan contoh *static-nodes.json*.

```
./bootnode -nodekey enode_id_1
```

Contoh *static-nodes.json* :

```
[
"enode://480
    cd6ab5c7910af0e413e17135d494d9a6b74c9d67692b0611e
4eefea1cd082adbdaa4c22467c583f
b881e30fda415f0f84cfea7ddd7df45e1e7499ad3c680c@127
    .0.0.1:23000?raftport=21000",
"enode://60998
    b26d4a1ecbb29eff66c428c73f02e2b8a2936c4bbb46581
ef59b2678b7023d300a31b899a7d82
cae3cbb6f394de80d07820e0689b505c99920803d5029a@127
    .0.0.1:23001?raftport=21001",
"enode://
    e03f30b25c1739d203dd85e2dcc0ad79d53fa776034074134ec2b

f128e609a0521f35ed341edd12e43
e436f08620ea68d39c05f63281772b4cce15b21d27941e@127
    .0.0.1:23002?raftport=21002"
]
```

Dalam *file* diatas, *port* 2300x adalah port untuk komunikasi *ethereum* protokol dan 2100x adalah *port* untuk komunikasi *raft* protokol.

apabila mengkonfigurasi *dinamic node*, perlu menjalankan perintah *raft.addPeer* seperti contoh berikut.

```
raft.addPeer(
"enode://27
    d3105b2c1173792786ab40e466fda80edf9582cd7fa1a867123dab
9e2f170be0b7e16d4065cbe81637759555603cc0619fcdf0fc7296d506
b9c26c26f3ae0c@127.0.0.1:23003?raftport=21003")
```

Setelahnya, perlu untuk membuat akun *ethereum* agar bisa melakukan transaksi satu dengan *node* lain. Terdapat *private key* setelah membuat akun yang terdapat pada *keystore*. Berikut perintah untuk menggenerate akun.

```
./geth --datadir ./accounts account new
```

Untuk konfiggurasi terakhir, perlu membuat *genesis block* yang bisa dibuat dengan file *genesis.json*, karena menggunakan privat jaringan *blockchain*, bisa membuat konfigurasi sendiri tergantung apa yang diinginkan dan agar optimal. Berikut contoh *file genesis.json*

```
{
"alloc": {
"0x65d8c00633404140986e5e23aa9de8ea689c1d05": {
"balance": "1000000000000000000000000000"
}
},
"coinbase": "0x0000000000000000000000000000000000000000",
"config": {
"homesteadBlock": 0
},
"difficulty": "0x0",
"extraData": "0x",
"gasLimit": "0x7FFFFFFFFFFFFFFF",
"mixhash": "0x00000000000000000000000000000000000000000
647572616c65787365646c6578",
"nonce": "0x0",
"parentHash": "0x00000000000000000000000000000000
0000000000000000000000000000000000",
"timestamp": "0x00"
}
```

Perlu diperhatikan, perlu mengganti alamat 0x65d8c006334041 40986e5e23aa9de8ea689c1d05 dengan alamat *node* sendiri.

Setelah konfigurasi selesai, bisa memulai menjalankan masing-masing node dan melakukan transaksi antar *node*. Dan untuk case ini, sudah disediakan *script* otomatis untuk konfigurasi *node*. Pada dasarnya, proses nya sama mulai dari konfigurasi *constellation* sampai membuat *genesis.json* juga untuk semua file dibuat template untuk memudahkan. Saat menjalankan script tersebut, akan terbentuk semua file dan konfigurasi yang diperlukan dan menjalankan

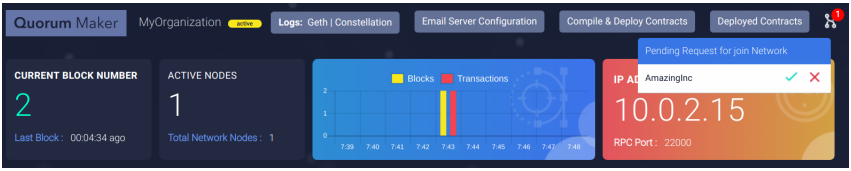
node tersebut dalam sebuah *docker container*. Tinggal menunggu *node* lain untuk bergabung.

Join Quorum Network

Pada dasarnya, untuk gabung pada jaringan diperlukan langkah yang hampir sama sebelumnya seperti membuat jaringan, yaitu melakukan konfigurasi *node* baru kemudian gabung dalam node yang sudah ada dengan menambahkan dengan *raft.addpeer*. Karena pembuatan node telah di lakukan *scripting* untuk mempermudah, maka tinggal mengisikan beberapa setting untuk bergabung pada jaringan. Ketika *script* berjalan akan membuat sebuah *node* baru menjalankan lewat *docker* dan melakukan *request join* ke *node* yang sudah ada.

Approve/Reject Join Request

Setelah menjalankan *setup* untuk bergabung ke jaringan, permintaan untuk bergabung akan dikirimkan ke salah satu *node* seperti yang dilihat pada gambar 3.6. Apabila permintaan gabung di terima, *node* yang *request* akan menerima main node's *constellation port*, *genesis file* dan *network ID*, namun apabila permintaan bergabung ditolak, *script* akan keluar tanpa menjalankan apapun.



Gambar 3.6: Notifikasi Bergabung *Quorum-Maker*[11]

Development network

Developer dapat mengimplementasikan *quorum network* dengan simulasi mengunakan *development network* yang artinya bisa membuat beberapa node pada *localhost* yang saling terkoneksi satu sama lain. Kemudian tiap *node* disambungkan dengan aplikasi pasien dan rumah sakit dan memasukan *private key* masing-masing node ke aplikasi. Setelah menjalankan *script* untuk membuat *development network*, akan dihasilkan konfigurasi masing masing *node*

dan sebuah file *docker-compose.yml*. File tersebut merupakan file *dokcer* berisi konfigurasi untuk menjalankan container dengan settingan yang telah diatur dan saling terkoneksi satu sama lain.

3.2.4 *Smart Contract Code*

Smart Contract diperlukan untuk membuat perjanjian dengan berbagai kondisi agar pada blokchain bisa mengatur sendiri tanpa adanya pihak ketika, disini kami membuat beberapa kondisi pada smart contract dalam bentuk kode. *Smart contract* kode dibuat dengan referensi [23]. Namun perlu dilakukan modifikasi untuk menyesuaikan dengan desain sistem yang digunakan pada penelitian ini.

Struktur Data Pada *Smart Contract*

Berikut merupakan struktur dari data pasien dan rumah sakit pada *smart contract*.

```
struct RumahSakit {
 string publicKey;
}
struct Permission {
 bool read;
 bool write;
 string reEncKey;
}
struct Token {
 int status;
 bool read;
 bool write;
 string reEncKey;
}
struct Data {
 uint number;
 string hashemr;
 string storagehash;
 address issuer;
}
struct Pasien {
 string publicKey;
 mapping (address => Permission) permissions;
 mapping (bytes32 => Token) tokens;
 bool closed;
 uint filecount;
 mapping(uint => Data) Datas;
}
```

Rumah sakit hanya menyimpan *public key* saja. Untuk *permission* ada *read, write, dan reKey. Read* dan *write* bertype *boolean* yang isinya hanya *true* dan *false*. Untuk token yang diberikan pasien, selain *read*, *write*, dan *reKey*, ada sebuah status, status ini yang membuat pembeda token belum di validasi, belum digunakan, ataupun sudah digunakan. Untuk data pasien, yang terpenting menyimpan *storage hash* yang merupakan alamat untuk data pasien. Untuk Pasien mempunyai *variable mapping* yang intinya setiap pasien bisa membuat data, token, dan *permission* yang berbeda beda.

Daftarkan *Public Key* pada Jaringan

Public key diperlukan untuk mengenkripsi pesan dengan PRE. Saat pertama kali pasien ataupun rumah sakit wajib mendaftarkan *public key* pada jaringan. Pada fungsi ini, diperlukan *public key* yang dihasilkan dari aplikasi.

```
function tambahPasien(string publicKey) public {
 if(address(keccak256(fromHex(publicKey))) == msg.sender) {
  Pasiens[msg.sender].publicKey = publicKey;
  }
}

function tambahRumahSakit(string publicKey) public {
 if(address(keccak256(fromHex(publicKey))) == msg.sender) {
  RumahSakits[msg.sender].publicKey = publicKey;
  }
}
```

Tambah Token

Untuk berbgai data akses pasien, dibuat sebuah token. Berikut kontrak kode untuk tambah token, token akan di validasi apabila belum pernah dibuat sebelumnya dan akan mendapatkan *permission* sesuai dengan apa yang diberikan oleh pasien. Pada fungsi ini, diperlukan 4 argument yaitu token *hash* dari token yang dihasilkan pada aplikasi, *read dan write* bertype *boolean*, dan *reEncKey* yang dibuat pada PRE melalui *private key* pasien dan *private key* yang dibuat pada aplikasi secara random dan berukuran 32 bytes.

```
function tambahToken(bytes32 hash, bool read, bool write,
    string reEncKey) public {
 if(Pasiens[msg.sender].tokens[hash].status == 0 && Pasiens
     [msg.sender].closed == false) {
  Pasiens[msg.sender].tokens[hash].status = 1;
  Pasiens[msg.sender].tokens[hash].read = read;
  Pasiens[msg.sender].tokens[hash].write = write;
  Pasiens[msg.sender].tokens[hash].reEncKey = reEncKey;
  }
}
```

Request Access

Request akses dilakukan untuk mendapatkan akses menulis ataupun membaca. apabila token sudah divalidasi, token akan bisa digunakan dan bernilai 2 (bertanda sudah digunakan). Dan dilakukan perjanjian pada *node* yang diberikan token telah mendapatkan akses untuk menulis atau membaca data pasien. Pada fungsi ini, diperlukan 3 argumen yaitu token, alamat pasien, dan alamat rumah sakit.

```
function requestAccess(string token, address Pasienadd,
    address myaddress) public {
 bytes32 hash = sha256(token);
 if(Pasiens[Pasienadd].tokens[hash].status == 1) {
  Pasiens[Pasienadd].tokens[hash].status = 2;
  Pasiens[Pasienadd].permissions[myaddress].read =
  Pasiens[Pasienadd].tokens[hash].read;
  Pasiens[Pasienadd].permissions[myaddress].write =
  Pasiens[Pasienadd].tokens[hash].write;
  Pasiens[Pasienadd].permissions[myaddress].reEncKey =
  Pasiens[Pasienadd].tokens[hash].reEncKey;
  emit tokenVerified(hash, Pasienadd, myaddress);
  }
}
```

Tambah dan Dapatkan Data

Ini merupakan contract untuk menambahkan data pasien dan mendapatkan data pasien pada *blockchain*. Data akan tersimpan dengan *filecount*, dengan ini pasien bisa mendapatkan history datanya. Dan dengan kontrak ini, pasien bisa melihat datanya sendiri.

```
function tambahData(string hashemr,string storagehash,
    address Pasienadd,address myaddress) public {
 if(Pasiens[Pasienadd].permissions[myaddress].write == true
     ) {
  Pasiens[Pasienadd].filecount ++;
  Pasiens[Pasienadd].Datas[Pasiens[Pasienadd].filecount] =
  Data(Pasiens[Pasienadd].filecount,hashemr,storagehash,
      myaddress);
  Pasiens[Pasienadd].permissions[Pasienadd].read = true;
  emit emrAdded(Pasienadd, myaddress, hashemr);
  }
}

function dapatkanData(address Pasienadd, address myaddress)
     public view returns (string storagehash) {
 if(Pasiens[Pasienadd].permissions[myaddress].read == true)
     {
  return Pasiens[Pasienadd].Datas[Pasiens[Pasienadd].
      filecount].storagehash;
  }
}
```

Dapatkan *Public Key*

Kode kontrak untuk mendapatkan *Public key* pasien maupun rumah sakit. *Public key* pasien diperlukan untuk mengenkripsi data pasien.

```
function dapatkanPasienPublicKey(address Pasienadd) public
    view returns (string publicKey) {
  return Pasiens[Pasienadd].publicKey;
}

function dapatkanRumahSakitPublicKey(address RumahSakitadd)
     public view returns (string publicKey) {
  return RumahSakits[RumahSakitadd].publicKey;
}
```

Revoke Access

Kode kontrak yang digunakan untuk mencabut akses. Pasien bisa menghilangkan perijinan suatu rumah sakit yang sudah diberikan.

```
function revokeRumahSakitAccess(address RumahSakitadd)
    public {
  Pasiens[msg.sender].permissions[RumahSakitadd].read =
      false;
  Pasiens[msg.sender].permissions[RumahSakitadd].write =
      false;
}
```

Check Permission dan Token

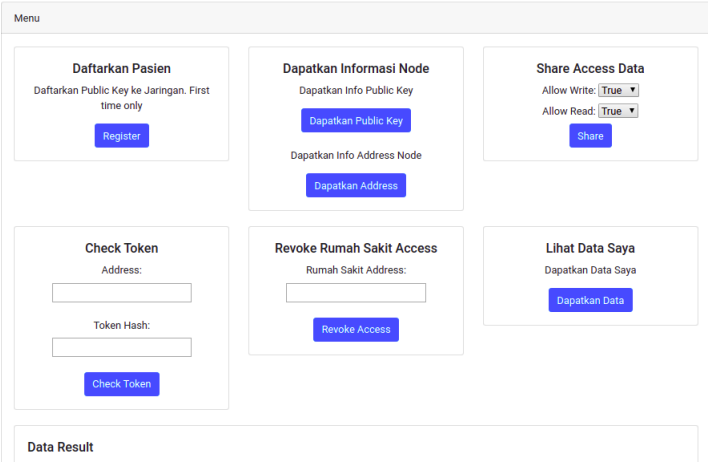
Kontrak kode yang digunakan untuk mengecek *permission* dan keterangan dari sebuah token.

```
function dapatkanPermission(address Pasienadd, address
    RumahSakitadd) public view returns(bool read, bool
    write, string reEncKey) {
  return (Pasiens[Pasienadd].permissions[RumahSakitadd].
      read, Pasiens[Pasienadd].permissions[RumahSakitadd].
      write, Pasiens[Pasienadd].permissions[RumahSakitadd
      ].reEncKey);
}

function dapatkanToken(address Pasienadd, bytes32 hash)
    public view returns (int status, bool read, bool write
    , string reEncKey) {
  return (Pasiens[Pasienadd].tokens[hash].status, Pasiens[
      Pasienadd].tokens[hash].read, Pasiens[Pasienadd].
      tokens[hash].write, Pasiens[Pasienadd].tokens[hash].
      reEncKey);
}
```

3.2.5 Pembuatan Aplikasi Pasien

Fitur terpenting dari aplikasi ini adalah sistem berbagi data, bagaimana enkripsi dekripsi pesan bisa dalam *blockchain* bisa membuat membagikan data ke orang lain dan data tersebut dapat langsung diakses, akan tetapi tetap memerlukan fitur tambahan. Adapun fitur tambahan dari pasien adalah *revoke hospital access*, cek token, serta *get personal information*. *Revoke hospital access* adalah fitur dimana pasien bisa mencabut *permission* dari rumah sakit yang diberikan akses sebelumnya, dengan mengubah ijin tulis dan membaca data nya menjadi *false*. Cek token bisa pasien gunak-

an untuk mengecek token yang sudah di hasilkan sebelumnya. Sedangkan untuk *get personal information*, pasien bisa melihat *public key* serta alamat akun miliknya. *Public key* dihasilkan dari *private key* yang dihasilkan *node* saat terbentuk dan tergabung dalam jaringan *blockchain*. Aplikasi pasien bisa dilihat pada github penulis. Tampilan *user interface* aplikasi sederhana bisa dilihat pada gambar 3.7

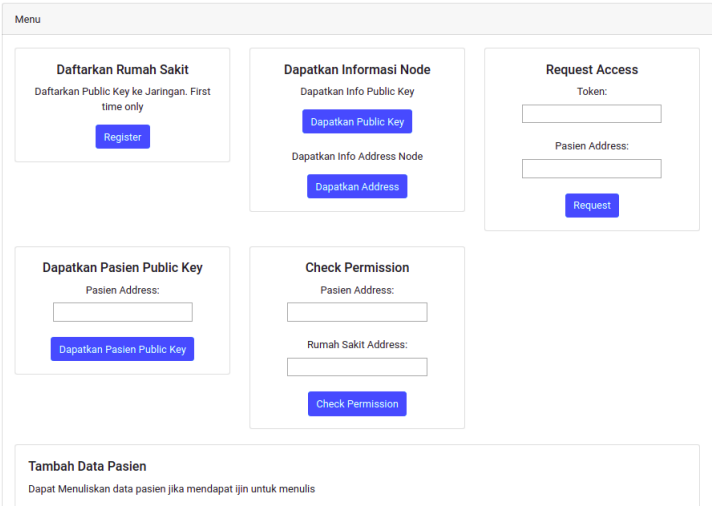


Gambar 3.7: *User Interface* Aplikasi Pasien

3.2.6 Pembuatan Aplikasi Rumah Sakit

Fitur utama pada rumah sakit adalah membuat data pasien dan melihat data pasien apabila diijinkan. Setelah diijinkan, rumah sakit bisa menuliskan data data pasien mulai dari nama umur dan keterangan pasien lainnya dan juga file data DICOM pasien yang dihasilkan dari rumah sakit tersebut yang bisa diambil dari server PACS. Untuk melihat data pasien membutuhkan *secret key* dari pasien karena data dienkripsi, setelahnya rumah sakit bisa *preview* data pasien dan mengunduh untuk dilihat menggunakan DICOM *viewer* milik rumah sakit. Pada rumah sakit juga ada fitur tambahan, yaitu cek *permission* pasien dengan memasukan alamat akun pasien, nantinya hubungan rumah sakit dan pasien akan diketahui boleh melihat boleh menulis atau tidak ada hubungan, kemudian

get patient public key dan juga *get my personal information* untuk melihat *public key* dan alamat akun. *User interface* aplikasi rumah sakit bisa dilihat pada gambar 3.8.



Gambar 3.8: *User Interface* Aplikasi Pasien

Halaman ini sengaja dikosongkan

BAB 4
PENGUJIAN DAN ANALISIS

Pada bab ini dipaparkan hasil pengujian serta analisis dari desain sistem dan implementasi. Dengan dilaksanakannya pengujian serta analisis tersebut, maka dapat ditarik kesimpulan dari pelaksanaan penelitian ini.

4.1 Persiapan Pengujian

Sebelum melakukan pengujian, maka diperlukan persiapan terlebih dahulu agar pengujian menjadi terarah. Pengujian pada penelitian ini meliputi pengujian fungsi pada sistem, serta pengujian *scalability* serta efisiensi penyimpanan dari sistem.

4.1.1 Perangkat yang Digunakan

Berikut perangkat yang digunakan dalam pengujian ini.

1. Laptop Intel i5-7200U (4) @ 2.50GHz dengan RAM 8 GB
2. 4 VPS dari azure dengan spesifikasi 1 vCPUs 2 GB RAM

4.1.2 Data simulasi yang Digunakan

Data simulasi yang digunakan untuk melakukan pengujian pada penelitian ini adalah data dummy informasi pasien dan data sample dicom dengan size yang beragam mulai dari 1 MB sampai dengan 64 MB.

4.2 Pengujian dan Analisis Sistem

Pengujian yang dilakukan pada penelitian ini meliputi:

1. Menguji Skenario *Sharing Access*
2. Menguji Skenario *Revoke Access*
3. Menguji Sistem *Scalability*
4. Menguji *Mining Execution Time* pada fitur.
5. Menguji Efisiensi penggunaan penyimpanan.
6. Analisis Serangan Terhadap Sistem.

Node	IP	RPC	Latency(ms)	Role
node1	10.50.0.2	22000	0,11	Rumah Sakit A
node2	10.50.0.3	22000	0,14	Rumah Sakit B
node3	10.50.0.4	22000	0,11	Rumah Sakit C
node4	10.50.0.5	22000	0,13	Pasien 1
node5	10.50.0.6	22000	0,10	Pasien 2
node6	10.50.0.7	22000	0,14	Pasien 3

Tabel 4.1: Konfigurasi yang Digunakan Untuk Simulasi

Test Case	Action
1	Pasien 1 memberikan akses membaca dan menulis ke rumah sakit 1
	Pasien 1 memberikan akses membaca dan menulis ke rumah sakit 2
	Pasien 1 memberikan akses membaca dan menulis ke rumah sakit 3
2	Pasien 2 memberikan akses membaca dan menulis ke rumah sakit 2
	Pasien 2 memberikan akses membaca ke rumah sakit 1
	Pasien 2 memberikan akses membaca ke rumah sakit 3
3	Pasien 3 memberikan akses menulis ke rumah sakit 3
	Pasien 3 memberikan akses membaca ke rumah sakit 2
	Pasien 3 tidak memberikan akses ke rumah sakit 1

Tabel 4.2: Keterangan *Test Case* untuk Pengujian Berbagi Data

4.2.1 Pengujian Skenario Berbagi Data

Dilakukan pengujian skenario untuk *sharing access* dan *support feature* pada aplikasi. Dibuat *development quorum network* yang terdiri dari 6 node, 3 pasien dan 3 rumah yang berbeda seperti pada TABLE 4.1 dan mengacu pada gambar 3.1 yang berjalan dalam *docker container*. Dilakukan percobaan dengan beberapa test case seperti pada TABLE 4.2.

Test Case 1 Didapatkan hasil pada TABLE 4.3. Ketika Rumah sakit mendapatkan token dan *secret key* dari pasien 1 yang berisi akses menulis dan membaca data pasien, maka rumah sakit

Pasien 1	Rumah Sakit 1	Rumah Sakit 2	Rumah Sakit 3
Dapat Menulis Data Pasien	Ya	Ya	Ya
Dapat Membaca Data Pasien	Ya	Ya	Ya

Tabel 4.3: Hasil *Test Case* 1

Pasien 2	Rumah Sakit 1	Rumah Sakit 2	Rumah Sakit 3
Dapat Menulis Data Pasien	Tidak	Ya	Tidak
Dapat Membaca Data Pasien	Ya	Ya	Ya

Tabel 4.4: Hasil *Test Case* 2

Pasien 3	Rumah Sakit 1	Rumah Sakit 2	Rumah Sakit 3
Dapat Menulis Data Pasien	Tidak	Tidak	Ya
Dapat Membaca Data Pasien	Tidak	Ya	Tidak

Tabel 4.5: Hasil *Test Case* 3

bisa menulis pasien data maupun membaca data pasien dan mendapatkan file dicom dari pasien.

Test Case 2 Didapatkan hasil pada TABLE 4.4. Bagi rumah sakit yang mendapat token dan *secret key* yang dapat akses membaca dan menulis bisa menuliskan data pasien dan membaca nya. Namun apabila rumah sakit menerima token dan *secret key* yang dapat akses membaca saja, maka rumah sakit tersebut hanya bisa membaca data pasien, tidak bisa menuliskan data pasien atau mengganti nya.

Test Case 3 Didapatkan hasil pada TABLE 4.5. Rumah sakit 3 hanya bisa menuliskan data pasien, tidak bisa membaca data pasien tersebut meskipun rumah sakit tersebut yang membuat data nya. Rumah sakit 2 hanya bisa membaca karena hanya mendapatkan akses untuk membaca data. Sedangkan rumah sakit 1 tidak bisa membaca maupun menulis karena tidak mempunyai akses apapun.

4.2.2 Pengujian Skenario *Revoke Access*

Dilakukan pengujian dengan skenario pasien telah memberikan akses kepada salah satu rumah sakit yang tersedia, kemudian dilakukan *revoke access* terhadap rumah sakit tersebut. Rumah sakit yang sebelumnya diberikan token untuk akses, token tersebut berubah *permission* nya menjadi *false* baik menulis ataupun membaca. Rumah sakit tersebut menjadi tidak bisa membaca atau menuliskan data pasien.

Node	Peran	Ports	IP	*Latency* (ms)
Node1	RS	22:ssh	52.163.226.108	2,108
		22000-22005:Quorum		
		8080,4001,5001:IPFS		
		8000:web		
		icmp		
Node2	Pasien	22:ssh	52.187.189.207	1,562
		22000-22005:Quorum		
		8080,4001,5001:IPFS		
		8000:web		
		icmp		
Node3	RS	22:ssh	13.76.161.210	1,432
		22000-22005:Quorum		
		8080,4001,5001:IPFS		
		8000:web		
		icmp		
Node4	Pasien	22:ssh	52.163.246.196	1,726
		22000-22005:Quorum		
		8080,4001,5001:IPFS		
		8000:web		
		icmp		

Tabel 4.6: Keterangan Masing-Masing Node Pada Vps

4.2.3 Pengujian *System Scalability*

Pengujian ini dilakukan dengan 4 vps yang dibuat dalam provider Azure. Masing-masing vps dilakukan konfigurasi mulai dari *package* yang diperlukan, hingga berjalannya *script* pembuatan privat *blockchain*. Kemudian masing-masing vps dipasang aplikasi dengan peran yang berbeda-beda. Berikut merupakan tabel keterangan dari 4 vps tersebut 4.6.

Selanjutnya setelah semua *node* terhubung satu sama lain, dilakukan pengujian *scalability* berupa unggah dan unduh data pada sistem dengan file DICOM yang beraneka ragam dari 1 Mb sampai 64 Mb.Tabel 4.7 dan tabel 4.8 merupakan hasil dari pengujian.

Dapat dilihat bahwa sistem mampu menangani berbagai ukuran data dicom mulai dari 1 Mb sampai dengan 64 Mb, setiap ke-

Ukuran (MB)	Unggah 1 (s)	Unggah 2 (s)	Unggah 3 (s)	Unggah 4 (s)	Unggah 5 (s)	Rata - Rata (s)
1	3,63	3,54	3,53	3,42	3,32	3,49
2	4,53	4,43	4,20	4,28	4,64	4,42
4	9,67	9,59	9,70	9,57	9,50	9,60
8	15,96	16,15	15,71	16,01	16,03	15,97
16	33,83	34,12	32,81	32,59	33,58	33,39
32	70,72	68,68	68,75	68,43	69,47	69,21
64	128,56	128,29	128,97	129,35	128,48	128,73

Tabel 4.7: Hasil Pengujian Unggah Pada Sistem

Ukuran (MB)	Unduh 1 (s)	Unduh 2 (s)	Unduh 3 (s)	Unduh 4 (s)	Unduh 5 (s)	Rata - Rata (s)
1	0,62	0,71	0,86	0,79	0,75	0,75
2	1,05	1,01	0,92	1,00	0,93	0,98
4	2,11	2,02	1,99	1,83	1,85	1,96
8	3,32	3,25	3,21	3,06	3,35	3,24
16	6,20	6,67	6,40	6,20	6,20	6,33
32	14,21	14,63	14,26	14,46	14,26	14,36
64	25,48	25,14	25,73	25,62	25,14	25,42

Tabel 4.8: Hasil Pengujian Unduh Pada Sistem

naikan ukuran data akan menambah waktu untuk unduh maupun unggah jelas karena data lebih besar. Untuk setiap percobaan, terjadi kenaikan atau penurunan jumlah waktu, tetapi tidak banyak. Hal ini membuktikan bahwa sistem penyimpanan IPFS cukup stabil, dan apabila ada sebuah data yang *hash* nya sama seperti yang sudah ada, akan lebih cepat transaksinya karena sistem IPFS terdapat *cache*. Semua percobaan diatas menggunakan data dengan *hash* yang berbeda beda, dengan ini bisa bandingkan kestabilan sistem penyimpanan IPFS. Begitu juga untuk unduh, apabila pertama kali melakukan unduh akan terjadi proses untuk mencari data dan menggabunggkan nya, namun setelah proses unduh pertama selesai, data akan langsung didapatkan karena sudah pernah digabungkan sebelumnya. Percobaan diatas menggunakan data *hash* yang berbeda saat unduh sehingga bisa membandingkannya. File pada IPFS pun terbagi bagi ke *node* yang terdekat, jadi penyimpanannya merata tidak terpusat. Untuk security data nya, tidak akan bisa mendapatkan data nya apabila tidak tau akan data *hash* yang disimpan. Dan data hash tersebut disimpan dengan keterangan pasien yang di enkripsi sehingga sangat sulit untuk mengambilnya.

4.2.4 Pengujian *Mining Execution Time*

Setiap fitur yang melakukan penambahan atau perubahan data pada *blockchain* dilakukan proses *mining*. Data yang akan di submit dibentuk *block* lalu disebarkan ke seluruh *node*. Setiap fitur memiliki perubahan atau penambahan yang berbeda karena memiliki struktur data yang berbeda beda. Tabel 4.9 merupakan hasil dari pengujian setiap fitur. Penambahan data pasien merupakan eksekusi yang paling lama karena data pasien lebih banyak, namun pembentukan *block* bisa dibilang cepat kurang dari satu detik. *Mining* data pasien bisa cepat karena data DICOM yang besar sudah disimpan dalam IPFS dan hasil *hash* dari IPFS yang dibuat *block* beserta data-data pasien. Pengujian ini dilakukan dengan 4 VPS publik. Dari pengujian ini didapatkan semua fitur berjalan dengan baik dan cepat.

Fitur	Mn 1 (s)	Mn 2 (s)	Mn 3 (s)	Mn 4 (s)	Mn 5 (s)	Rata - Rata (s)
Tambah Pasien *Public Key*	0,46	0,46	0,39	0,40	0,37	0,42
Tambah RS *Public Key*	0,54	0,45	0,45	0,43	0,46	0,47
Tambah Token	0,41	0,39	0,38	0,34	0,38	0,38
Request Access	0,57	0,50	0,46	0,44	0,48	0,49
Tambah Data Pasien	0,61	0,51	0,49	0,50	0,47	0,52
Cabut Akses RS	0,52	0,44	0,49	0,45	0,44	0,47

Tabel 4.9: Hasil Pengujian *Mining* pada Fitur Sistem

Ukuran (MB)	Rata - Rata (s)
1	3,49
2	4,42
4	9,60
8	15,97
16	33,39
32	69,21
64	128,73

Tabel 4.10: Hasil Pengujian Dengan Menyimpan Data Pasien Pada IPFS

4.2.5 Pengujian Penggunaan Penyimpanan

Dilakukan juga pengujian dengan memakai IPFS dan tidak memakai IPFS dengan cara menyimpan nya langsung dalam *blockchain*. Pengujian dilakukan sama menggunakan data DICOM mulai dari 1 Mb sampai dengan 64 Mb. Didapatkan hasil pengujian pada tabel 4.10 dan tabel 4.11. Dengan menggunakan IPFS diketahui semua data bisa tersimpan dan transaksi berhasil dilakukan, didapatkan rata-rata setiap unggah file. Pada pengujian tanpa menggunakan IPFS, didapatkan hasil file tidak bisa disimpan karena file terlalu besar. Dari sini bisa dapatkan sebuah hasil bahwa *blockchain* tidak cocok untuk penyimpanan yang besar, tetap membutuhkan penyimpanan. Penyimpanan secara desentralisasi dengan IPFS ataupun penyimpanan terpusat bisa saja dikombinasikan dengan *blockchain*.

4.2.6 Analisis Serangan yang Bisa Terjadi Terhadap Sistem

Pada keamanan *Smart Contract*, menurut penelitian milik Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli, terdapat beberapa attack yang bisa terjadi terhadap *ethereum smart contract*. *Taxonomy* dari *vulnerabilities Ethereum Smart Contract* dipaparkan pada table 4.12.

Semua *vulnerability* yang ada pada level solidity dijalankan untuk mengexploitasi pencurian uang dari *contracts*. Sedangkan sistem pada penelitian ini tidak menggunakan function yang bersi-

Ukuran (MB)	Hasil
1	Error File to Large
2	Error File to Large
4	Error File to Large
8	Error File to Large
16	Error File to Large
32	Error File to Large
64	Error File to Large

Tabel 4.11: Hasil Pengujian Dengan Menyimpan Data Pasien Langsung Pada *Blockchain*

Level	*Cause of vulnerability*
Solidity	*Call to the unknown*
	Gasless send
	Exception disorders
	Type casts
	Reentrancy
	Keeping secrets
EVM	*Immutable bugs*
	Ether lost in transfer
	Stack size limit
Blockchain	*Unpredictable state*
	Generating randomness
	Time Constrain

Tabel 4.12: *Taxonomy of Vulnerability Ethereum Smart Contract* [12]

fat *payable* dan tidak terdapat transaksi untuk mengirimkan atau menerima dana. Fungsi-fungsi yang dipakai pada sistem ini hampir semua hanya untuk membuat sebuah *state* data pasien dan *permission* nya dalam blockchain.

Vulnerabiliy pada EVM, *Immutable bugs* pernah terjadi pada sistem, saat kontrak sudah di *deploy* pada jaringan dan ternyata terdapat sebuah *bug* pada kode kontrak saat penambahan data pasien. Kontrak tidak akan bisa diubah apabila sudah di *deploy* pada *blockchain*, untuk itu diharuskan untuk melakukan *reset* terhadap jaringan *blockchain*, dari ini disarankan untuk menggunakan *development network* untuk pengujian sebelum di *deploy* pada *network production*. *Ether lost in transfer* tidak terjadi dalam sistem ini, karena menggunakan *quorum* dan *blockchain* bersifat konsortium atau semi-privat sehingga *ether* tidak digunakan dalam sistem ini karena *transaction fee* bisa dihilangkan dalam jaringan *quorum*. Sedangkan untuk *vulnerability Stack size limit*, kontrak pada penelitian ini hanya satu, tidak terdapat kontrak lain, sehingga tidak terdapat *nested calls* pada kontrak.

Vulnerability pada *blockchain*, *unpredictable state* belum terjadi pada sistem dan mungkin bisa terjadi. *Vulnerability* ini terjadi saat jaringan pada *blockchain* terdapat banyak *node*, saat pengguna melakukan transaksi 1 kepada *smart contract* sebuah *state*, transaksi ini belum dilakukan mining dan pembuatan *block* karena jaringan masih sibuk untuk melakukan transaksi, kemudian pengguna melakukan transaksi 2 saat transaksi 1 belum tersubmit, dan hasilnya transaksi 2 tersubmit baru transaksi 1 menyebabkan terjadinya *unpredictable state*. Untuk *vulnerability generating randomness*, tidak digunakan random pada *smart contract* tetapi random token yang dihasilkan pada sisi aplikasi untuk menghindari *vuln* ini.

Dalam jaringan *quorum* sendiri, *transaction fee* dapat dihilangkan sehingga dapat dimungkinkan untuk terjadi serangan *DDoS* dalam jaringan *quorum*, namun quroum tidak banyak memperpedulikan tentang serangan *DDoS*, mengapa? berikut merupakan beberapa point penting dalam masalah ini:

1. Dalam *quorum*, konsep *gas* untuk transaksi masih terdapat dalam *quorum*, tetapi benar tidak terdapat *gas price* untuk transaksi, yang artinya setiap transaksi tetap terikat oleh *gas*

limit yang dikonfigurasi pada *genesis*, dan transaksi yang melebihi *gas limit* akan di tolak (sama seperti ethereum) [27].

2. *Quorum* baik digunakan untuk jaringan privat. Artinya setiap pengguna yang berada pada jaringan tersebut bisa mengetahui *node* mana yang melakukan serangan pada *blockchain*, juga bisa dilakukan blokir terhadap *node* yang melakukan serangan [28].

3. Bisa diterapkan *firewall* pada *quorum node* untuk mengatasi serangan *DDoS* [28].

Halaman ini sengaja dikosongkan

BAB 5
PENUTUP

5.1 Kesimpulan

Sistem berbagi *record* data PACS pasien dalam penelitian ini telah berhasil dalam mengintegrasikan *record* data pasien antar rumah sakit juga tambahan keamanan dengan pemberian akses untuk berbagi, rumah sakit yang mendapat akses token dan *secret key* dapat membaca maupun menulis data pasien sesuai akses token yang diberikan pasien. Bagi yang tidak mendapatkan akses tidak akan bisa membaca maupun menulis data pasien, dengan ini keamanan dan privasi pasien terjaga penuh. Kendali atas datanya tetap pasien yang menentukan. Kejadian data breach juga sulit dilakukan pada sistem aplikasi yang bersifat desentralisasi dan menggunakan platform blockchain apalagi data juga dienkripsi.

Terkait dengan *scalability*, sistem mampu menangani berbagai size file yang beragam mulai dari 1Mb sampai dengan 64 Mb, dan dari dilakukannya beberapa pengujian didapatkan hasil eksekusi waktu yang mendekati sama, dari ini bisa diketahui bahwa IPFS stabil, juga bisa lebih cepat apabila file sudah pernah unggah sebelumnya (*hash* file sama).

Pada fitur aplikasi, semakin banyak data yang di buat *block* akan semakin banyak waktu yang diperlukan untuk *mining*. Terbukti pada feature panambahan data pasien didapatkan rata-rata waktu *mining* 0,51 detik, lebih lama dari feature yang lain.

Untuk penyimpanan, penyimpanan file menggunakan IPFS lebih baik daripada menyimpannya langsung dalam *blockchain*, karena menyimpan langsung dalam *block* mempunyai batas. Tidak menutup kemungkinan untuk menggunakan penyimpanan terpusat seperti *mysql*, atau *postgresql* dan database lainnya karena bisa dikombinasikan. Untuk penelitian ini menggunakan IPFS karena ingin sepenuhnya sistem berjalan dengan desentralisasi juga mudah digunakan dan dikombinasikan dengan platform *blockchain*.

5.2 Saran

1. Pembuatan aplikasi pasien pada *smart phone* apabila *node* bisa berjalan lewat *mobile*.

2. Alamat akun, token dan *secret key* dibuat QR Code untuk memudahkan berbagi akses.

DAFTAR PUSTAKA

[1] A. B. S. S. K. M. Y. Ahsan Manzoor, Madhsanka LiyanageT, Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing. (Dikutip pada halaman xi, 5, 6).

[2] N. Nizamuddin, H. Hasan, and K. Salah, IPFS-Blockchain-Based Authenticity of Online Publications, pp. 199–212. 06 2018. (Dikutip pada halaman xi, 2, 6, 7).

[3] R. K. N. J. A. M. Q. M. K. A. Muqaddas Naz, Fahad A. Alzahrani and M. Shafiq, A Secure Data Sharing Platform Using Blockchain and Interplanetary File System. (Dikutip pada halaman xi, 7, 8).

[4] R. T. Randhir Kumar, Ningrinla Marchang, of Patient Diagnostic Reports in Healthcare System using IPFS and Blockchain. (Dikutip pada halaman xi, 7, 9).

[5] Jaringan Peer to peer Image. (Dikutip pada halaman xi, 10).

[6] Contoh Fungsi Hash Image. (Dikutip pada halaman xi, 14).

[7] Dapp Architecture Image. (Dikutip pada halaman xi, 15).

[8] D. A. Badawi, SISTEM VERIFIKASI DOKUMEN HASIL INVESTIGASI FORENSIK DIGITAL BERBASIS TEKNOLOGI BLOCKCHAIN. (Dikutip pada halaman xi, 10, 15, 18).

[9] Proxy Re-Encryption Image. (Dikutip pada halaman xi, 22).

[10] Picture Archiving and Communication System Ilustration Image. (Dikutip pada halaman xi, 23).

[11] Synechron-Finlabs, "Quorum-maker." `https : / / github.com / synechron - finlabs / quorum - maker`, 2019. (Dikutip pada halaman xi, 35, 38, 65, 66).

[12] M. B. Nicola Atzei and T. Cimoli, A survey of attacks on Ethereum smart contracts. (Dikutip pada halaman xiii, 55).

[13] R. T. Adi, Implementasi PACS Orthanc Berbasis PostgreSQL dan Radiology Information System. Institut Teknologi Sepuluh Nopember, 2019. (Dikutip pada halaman 1, 23, 24, 34).

[14] S. M. I. Mona Al-Hajeri, Malcom Clarke, Future Trends in Picture Archiving and Communication System(PACS). 2015. (Dikutip pada halaman 1).

[15] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system. (Dikutip pada halaman 2).

[16] R. N. W. S. Mathis Steichen, Beltran Fiz and R. State, Blockchain-Based, Decentralized Access Control for IPFS. (Dikutip pada halaman 8, 20).

[17] "Definisi perbedaan jaringan terpusat vs terdistribusi (sentralisasi dan desentralisasi)." https://medium.com/@arcx/definisi-perbedaan-jaringan-terpusat-vs-terdistribusi-sentralisasi-dan-desentralisasi-a8e10216eedd. terakhir diakses tanggal 27 April 2020. (Dikutip pada halaman 9).

[18] W. Santoso, Sistem Keamanan Data Rekapitulasi Suara Pemilu Dengan Menggunakan Metode Penyimpanan Blockchain Secara Bertingkat. Institut Teknologi Sepuluh Nopember, 2019. (Dikutip pada halaman 10).

[19] "Type blockchain." https://blockchainisme.com/blockchain/3-tipe-blockchain-yang-perlu-anda-ketahui/. terakhir diakses tanggal 12 July 2020. (Dikutip pada halaman 10).

[20] "Kelemahan blockchain." https://digitalis.id/blog/mengenal-kekurangan-kelemahan-dari-teknologi-blockchain/. terakhir diakses tanggal 12 July 2020. (Dikutip pada halaman 11).

[21] "Perbedaan blockchain." https://www.kompasiana.com/ariftrick/5e79bdf9d541df155b6efdc2/apa-itu-blockchain-bedanya-dengan-database-

penjelasan-untuk-orang-awam. terakhir diakses tanggal 12 July 2020. (Dikutip pada halaman 12).

[22] "Fungsi hash kriptografi)." https://blog.compactbyte.com/2019/02/20/fungsi-hash-kriptografi-bagian-1/. terakhir diakses tanggal 27 April 2020. (Dikutip pada halaman 14).

[23] N. Prusty, Blockchain For Enterprise. Packt Publishing Ltd., 2018. (Dikutip pada halaman 14, 15, 16, 39).

[24] V. Dhillon, D. Metcalf, and M. Hooper, Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You. 01 2017. (Dikutip pada halaman 17).

[25] "How ipfs works." https://medium.com/@akshay_111meher/how-ipfs-works-545e1c890437. terakhir diakses tanggal 27 April 2020. (Dikutip pada halaman 20).

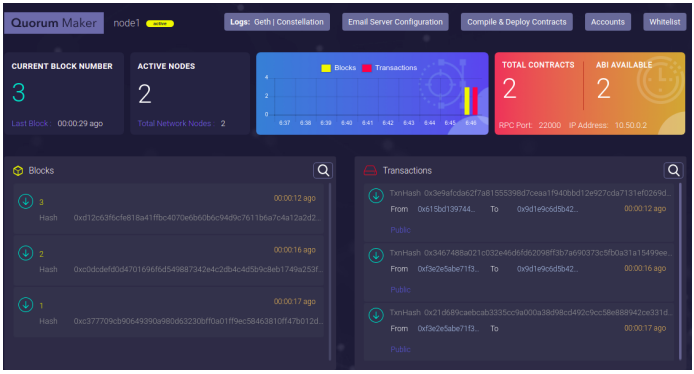
[26] S. W. Zhiguang Qin, Hu Xiong and jennifer Batamuliza, A Survey of Proxy Re-Encryption for Secure Data Sharing in Cloud Computing. (Dikutip pada halaman 21).

[27] "Prevent ddos attack in quorum." https://github.com/jpmorganchase/quorum/issues/793. terakhir diakses tanggal 15 July 2020. (Dikutip pada halaman 57).

[28] "Guard again ddos in quorum." https://ethereum.stackexchange.com/questions/52758/quorum-how-to-guard-against-ddos-if-there-are-no-gas-transaction-fee-in-privat. terakhir diakses tanggal 15 July 2020. (Dikutip pada halaman 57).

Halaman ini sengaja dikosongkan

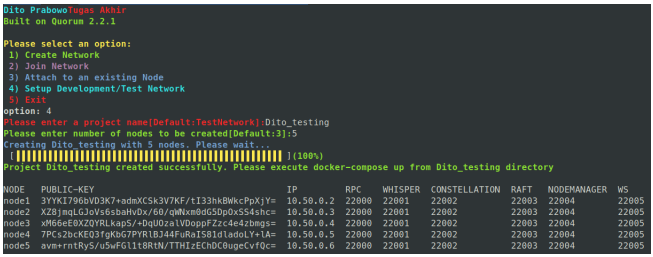
LAMPIRAN



Gambar 1: Quorum-Maker Interfaces[11]

```
2_deploy_contract.js
====================

   Deploying 'Health'
   ------------------
   > transaction hash:    0x3a06db61f9f2768bd32791789f29e09c3f9aea88559d835ae6ea66db420c0f8b
   > Blocks: 0            Seconds: 0
   > contract address:    0x58392ec02D011DB90F25071f99e53061599E944a
   > block number:        8
   > block timestamp:     0x160a2e3e5b4996ec
   > account:             0x0de757f15ec95b3B44f228fe856EDb07670d1e47
   > balance:             1000000000
   > gas used:            3172720
   > gas price:           0 gwei
   > value sent:          0 ETH
   > total cost:          0 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:                  0 ETH
```

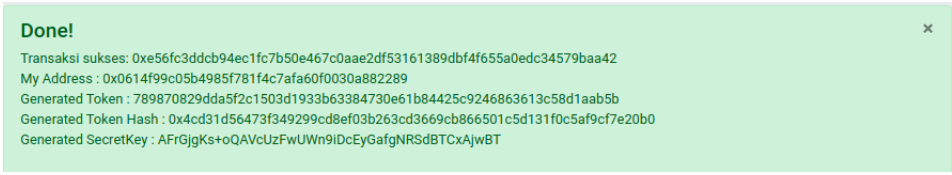
Gambar 2: Deploy Smart Contract



Gambar 3: Quorum-Maker Development Network

Gambar 4: Quorum-Maker Create Network[11]

Gambar 5: Quorum-Maker Join Network[11]



Gambar 6: Patient Share Access Result

Halaman ini sengaja dikosongkan

BIOGRAFI PENULIS

Dito Prabowo, lahir pada 10 Juni 1998 di Nganjuk, Jawa Timur. Penulis lulus dari SMP Negeri 1 Nganjuk pada tahun 2013 kemudian melanjutkan pendidikan ke SMA Negeri 2 Nganjuk hingga akhirnya lulus pada tahun 2016. Penulis kemudian melanjutkan pendidikan S-1 ke Departemen Teknik Komputer, FTEIC-ITS Surabaya. Saat di kuliah penulis aktif menjadi Assistant Laboratorium B201. Bagi pembaca yang memiliki kritik, saran atau pertanyaan mengenai penelitian ini dapat menghubungi penulis melalui email ditoprabowo98@gmail.com.

Halaman ini sengaja dikosongkan