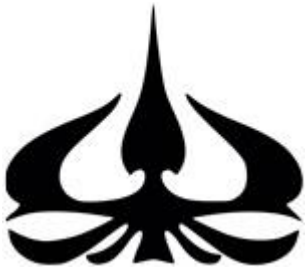


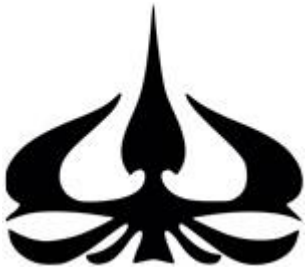
UNIVERSITAS TRISAKTI

APLIKASI *BLOCKCHAIN* PADA *INTERNET OF THINGS* UNTUK SEKURITISASI TRANSAKSI DI ETHEREUM TESTNET

TESIS

JONI FAT
162 171 002

FAKULTAS TEKNOLOGI INDUSTRI
MAGISTER TEKNIK ELEKTRO
UNIVERSITAS TRISAKTI
2019



UNIVERSITAS TRISAKTI

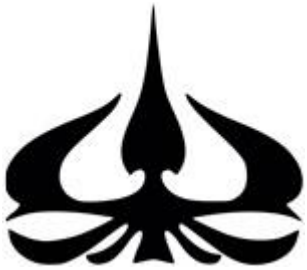
APLIKASI *BLOCKCHAIN* PADA *INTERNET OF THINGS* UNTUK SEKURITISASI TRANSAKSI DI ETHEREUM TESTNET

TESIS

Diajukan sebagai salah satu syarat untuk memperoleh gelar Magister Teknik (M.T.)

JONI FAT
162 171 002

FAKULTAS TEKNOLOGI INDUSTRI
MAGISTER TEKNIK ELEKTRO
UNIVERSITAS TRISAKTI
2019



TRISAKTI UNIVERSITY

BLOCKCHAIN APPLICATION IN INTERNET OF THINGS FOR SECURING TRANSACTION IN ETHEREUM TESTNET

THESIS

Proposed as one of requirements for obtaining Master of Engineering Degree

JONI FAT
162 171 002

FACULTY OF INDUSTRIAL TECHNOLOGY
MASTER OF ELECTRICAL ENGINEERING
TRISAKTI UNIVERSITY
2019

HALAMAN PENGESAHAN

Tesis ini diajukan oleh	:
Nama	: JONI FAT
NIM	: 162 171 002
Program Studi	: Magister Teknik Elektro
Judul Skripsi/Tesis	: APLIKASI *BLOCKCHAIN* PADA *INTERNET OF THINGS* UNTUK SEKURITISASI TRANSAKSI DI ETHEREUM TESTNET

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Magister Teknik (M.T.) pada Program Studi Pascasarjana Teknik Elektro, Fakultas Teknologi Industri, Universitas Trisakti.

DEWAN PENGUJI

Pembimbing	: Henry Candra, S.T., M.T., Ph.D.	( )
Penguji I	: Prof. Dr. Ir. Indra Surjati, MT. IPM.	( )
Penguji II	: Dr. Ir. Yuli Kurnia Ningsih, MT. IPM.	( )
Penguji III	: Dr. Ir. Suhartati Agoes, MT.	( )

Ditetapkan di : Jakarta

Tanggal : 30 Agustus 2019

KATA PENGANTAR

Puji syukur dan ungkapan terima kasih Penulis panjatkan kepada Tuhan atas berkat-Nya atas penyelesaian Tesis ini. Ungkapan terima kasih juga Penulis haturkan kepada keluarga Penulis, dan rekan-rekan yang telah memberikan dukungan baik secara moril mau pun materiil. Terutama, Penulis juga mengungkapkan terima kasih kepada:

(1) Henry Candara, ST. MT. PhD., selaku dosen pembimbing. Beliau telah banyak meluangkan waktu, tenaga dan pikiran untuk membantu Penulis dalam penyelesaian Tesis ini;
(2) Dr. Ir. Yuli K. Ningsih, MT., selaku dosen penguji dalam sidang Tesis;
(3) Prof. Dr. Ir. Indra Surjati, MT., selaku dosen penguji dalam sidang Tesis;
(4) Dr. Suhartati Agoes, atas dukungan dan pengarahannya dalam pembuatan Proposal Tesis serta selaku dosen penguji dalam sidang Tesis;
(5) Dr. Lydia Sari, selaku koordinator Tesis, atas dukungan dan pengarahannya;
(6) Sahabat yang mendoakan Penulis dalam menyelesaikan tesis ini.

Akhir kata, semoga tesis ini membawa manfaat bagi pengembangan ilmu.

Jakarta, 30 Agustus 2019

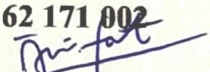
Penulis

HALAMAN PERNYATAAN ORISINALITAS

Tesis ini adalah hasil karya saya,

dan semua sumber baik yang dikutip maupun dirujuk

telah saya nyatakan dengan benar.

Nama	**: JONI FAT**
NIM	**: 162 171 002**
Tanda Tangan	**:** 
Tanggal	**: 31 Juli 2019**

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Trisakti, saya yang bertanda tangan di bawah ini:

Nama : JONI FAT
NIM : 162 171 002
Program Studi : Pascasarjana Teknik Elektro
Fakultas : Teknologi Industri
Jenis Karya : Tesis

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Trisakti **Hak Bebas Royalti Noneksklusif (*Non-exclusive Royalty-Free Right*)** atas karya ilmiah saya yang berjudul:

APLIKASI *BLOCKCHAIN* PADA *INTERNET OF THINGS* UNTUK *SIGNING* TRANSAKSI DI ETHEREUM TESTNET

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalty Noneksklusif ini Universitas Trisakti berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di Jakarta
Pada Tanggal 31 Juli 2019
Yang Menyatakan

JONI FAT

ABSTRAK

Nama : JONI FAT
Program Studi : Pascasarjana Teknik Elektro
Judul : APLIKASI *BLOCKCHAIN* PADA *INTERNET OF THINGS* UNTUK *SIGNING* TRANSAKSI DI ETHEREUM TESTNET

Sistem yang dirancang menggunakan pemroses Lolin D32 dan modul sensor MPL3115A2, berupaya memperlihatkan bahwa peralatan berbasis *Internet of Things* (IoT) dapat disekuritisasi dengan menggunakan teknologi *blockchain*. Teknologi *blockchain* yang digunakan adalah teknologi berbasis Ethereum yang diuji coba pada jaringan Testnet Ropsten. Modul pemroses terhubung dengan Internet menggunakan modul WiFi yang merupakan salah satu fitur dari Lolin D32. Modul WiFi melakukan koneksi ke *router* yang telah disiapkan untuk keperluan perancangan dan pengujian sistem. Sistem ini berupaya menunjukkan bahwa data yang diakuisisi dari modul sensor MPL3115A2 (ketinggian, temperatur dan tekanan) dan data posisi dapat di-*signing* dan verifikasi pada modul pemroses Lolin D32. Sebelum dikirim, data-data tersebut akan diformat terlebih dahulu sehingga kompak. Data ini akan disimpan dalam *smart contract. Smart contract* merupakan program Ethereum yang diprogram dengan bahasa Solidity. *Smart contract* ini di-*deploy* ke jaringan Ropsten. Hasil pengujian sebanyak 516 kali pada jaringan Ropsten menunjukkan bahwa data-data dari modul pemroses berhasil di-*signing* dan verifikasi di modul tersebut. Data juga berhasil dikirim ke jaringan Ropsten dan terbukti data-data tersebut terekam. Ini dibuktikan dengan pemeriksaan pada etherscan.io.

Kata kunci: *Blockchain*, Ethereum, *Internet of Things* (IoT), Lolin D32, MPL3115A2, *signing*, verifikasi

ABSTRACT

Name : JONI FAT
Graduate Program : Master Degree in Electrical Engineering
Judul : BLOCKCHAIN APPLICATION IN INTERNET OF THINGS FOR SIGNING TRANSACTION IN ETHEREUM TESTNET

This system is designed by using Lolin D32 as the processor dan MPL3115A2 for the sensors. This system is designed to show that devices that based on Internet of Things (IoT) could be secured by using blockchain technology. The blockchain technology is based on Ethereum which will be tested in Ropsten Testnet network. Processor module connects to Internet through WiFi module which is one of the Lolin D32 features. This WiFi module connects to router which is used for designing and testing the system. This system wants to prove that the data which are acquired from sensor module MPL3115A2 (altitude, temperature and pressure) and positioning data could be sign and verify within the processor Lolin D32. Before sending the data, the data will be formatted. Data will be stored in smart contract. The smart contract is an Ethereum program which is written using Solidity language. This smart contract is delployed into Ropsten network. The numbers of testing that have been carried out are 516 times. These testings proved as success. The data which were sent to Ropsten network could be proved that they were recorded successfully. These are done by checking through etherscan.io.

Keywords: Blockchain, Ethereum, Internet of Things (IoT), Lolin D32, MPL3115A2, sign, verify

BAB 1
PENDAHULUAN

1.1. Latar Belakang Masalah

Internet of Things (IoT) merupakan salah satu fokus dalam Revolusi Industri 4.0. IoT adalah gelombang perubahan berikutnya setelah era Internet. Diperkirakan akan ada 46 milyar peralatan yang terhubung ke Internet pada tahun 2021 [1]. Peralatan yang terhubung ke Internet ini, mulai dari *Body Area Network* (BAN), berupa jaringan dengan peralatan di sekitar tubuh, hingga jangkauan yang sangat luas. Ini berarti akan banyak data sensitif yang akan terekspos. Hingga saat ini, isu sekuritisasi data ini masih menjadi pokok permasalahan, karena berdasarkan hasil penelitian yang dipublikasi oleh HP, 70% komunikasi IoT tidak terenkripsi [2]. Dengan bertambahnya popularitas *blockchain*, timbul ide untuk memadukan *blockchain* dan IoT.

Blockchain memiliki keunggulan dari sisi sekuritisasi. Enkripsi dalam *blockchain* memberikan harapan bahwa sekuritisasi komunikasi mau pun data IoT dapat dilakukan dengan baik. Walau pun hal tersebut masih diiringi dengan beberapa kekurangan dari teknologi *Blockchain*. Teknologi *Blockchain* mengggunakan kritografi secara masif [3]. Dengan demikian, *blockchain* dapat digunakan untuk menjamin transaksi dalam jaringan yang memiliki tingkat keamanan yang rendah seperti Internet. Setiap blok dalam *blockchain* diidentifikasi dengan sebuah kode *hash* dan *nonce*. Kedua kode ini bersifat unik. Setiap blok saling berkaitan dengan blok sebelumnya. Bila sebuah blok diubah, maka keseluruhan rantai bloknya harus diubah juga. Hal ini mengakibatkan, pengubahan blok sangat sulit dilakukan dengan daya komputasi saat ini. Selain itu, teknologi *blockchain* juga memanfaatkan metode konsensus [3]. Ini berarti setiap transaksi dalam *blockchain* harus mencapai nilai konsensus tertentu agar tervalidasi dalam jaringannya. Blok juga terduplikasi dalam setiap *server* dalam jaringan *server* tersebut. Dengan demikian, dapat dipahami bahwa untuk mengubah transaksi dalam *blockchain* secara tidak sah akan sangat sulit dilakukan. Oleh karena itu, *blockchain* memiliki sekuritisasi transaksi yang

mustahil untuk diubah secara tidak sah dengan daya komputasi saat ini. Danzi [4] menunjukkan upaya untuk membangun konsep arsitektur IoT dan *blockchain* untuk menyelesaikan persoalan sekuritisasi dalam metode pembayaran mikro. Danzi membangun model penyelesaian tersebut. Danzi menggunakan mikrokontroler sebagai peralatan IoT terhubung dengan jaringan *blockchain*. Interaksi antara mikrokontroler dan jaringan *blockcahin* ini bersifat *irreversible* dan tanpa perlu *mutual trust*.

Dengan demikian, dapat disimpulkan bahwa mikrokontroler merupakan bagian utama, karena semua proses dimulai dari mikrokontroler ini. Sensor, aktuator, dan peralatan lain terhubung dengan mikrokontroler. Oleh mikrokontroler data diakuisisi atau diproses dan diteruskan ke modul jaringan yang juga terhubung dengan mikrokontroler. Dapat dikatakan bahwa semua proses IoT dimulai di mikrokontroler. Oleh sebab itu, proses yang dalam sekuritisasi adalah proses sekuritisasi dalam pemrosesan oleh mikrokontoler ini. Jadi, untuk dapat memastikan bahwa *blockchain* dapat diaplikasikan pada peralatan IoT, pada perancangan ini akan menggunakan mikrokontroler untuk menunjukkan bahwa *signing* dapat dilakukan [5]. Penggunaan mikrokontroler ini memenuhi tujuan pembuktian bahwa *blockchain* dan IoT dapat dipadukan.

1.2. Rumusan Masalah

IoT sebagai gelombang pembaharuan setelah Internet memiliki kendala utama dalam hal sekuriti. Dengan munculnya teknologi *blockchain*, memberikan satu ide untuk memadukan kedua hal tersebut. Ini dengan pertimbangan bahwa teknologi *blockchain* yang menekankan pada sekuriti dan kemampuan untuk melakukan transaksi dalam jaringan yang tidak aman mampu diterapkan pada mikrokontroler. Oleh karena, mikrokontrolerlah merupakan inti dari IoT.

Dengan demikian, untuk Tesis ini, dapat dikemukakan rumusan masalah sebagai berikut:

1. Apakah transaksi *blockchain* dapat di-*sign* secara langsung di mikrokontroler?

2. Apakah transaksi yang telah di-*sign* dapat diproses oleh jaringan *blockchain* Ethereum?
3. Apakah transaksi tersebut terverifikasi dengan baik?

1.3. Batasan Masalah

Teknologi *blockchain* mau pun IoT memiliki spektrum permasalahan yang sangat luas. Untuk Tesis ini, dibatasi hanya pada proses verifikasi dan *signing* transaksi oleh mikrokontroler, pengiriman transaksi yang telah terenkripsi ke jaringan *blockchain* Ethereum TestNet dengan memanfaatkan *Remote Procedure Call* (RPC) Infura, dan menggunakan etherscan.io untuk memeriksa apakah transaksi berhasil terverifikasi dengan baik oleh *node* dalam jaringan *blockchain*. Dalam Tesis ini, tidak mempersoalkan jenis data transaksi, walau pun data tersebut pada dasarnya dapat berupa data identifikasi modul-modul yang terhubung dengan mikrokontroler mau pun identitas dari mikrokontroler itu sendiri. Data yang digunakan di sini adalah data dari sensor ketinggian, temperatur, tekanan dan posisi.

1.4. Tujuan Penelitian

Tesis ini merupakan upaya awal untuk memperlihatkan bahwa ide memanfaatkan teknologi *blockchain* untuk melakukan sekuritisasi dalam IoT dapat diwujudkan. Konsep untuk memadukan kedua teknologi ini telah banyak dikemukakan dalam berbagai macam jurnal penelitian, yang secara umum dapat dilihat pada **Daftar Pustaka** dalam Tesis ini. Ide pembuktian ini dititikberatkan pada sekuritisasi data sensor (berupa ketinggian, temperatur, tekanan dan posisi) di mikrokontroler. Transaksi ini adalah berupa data *dummy*. Data ini kemudian dikirim ke jaringan *blockchain*. Data ini dapat disimulasikan dalam bentuk seperti yang telah dijelaskan pada **Rumusan Masalah**.

1.5. Manfaat Penelitian

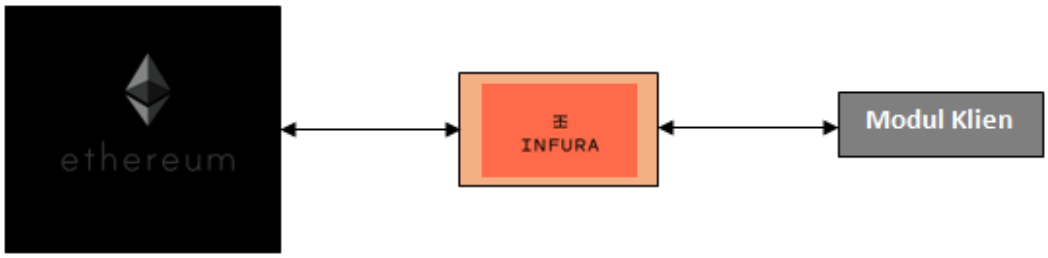
Apabila Tesis ini berhasil memperlihatkan bahwa data transaksi di mikrokontroler dapat diverifikasi oleh jaringan Ethereum, maka diharapkan akan menjadi fondasi untuk mengembangkan IoT yang berbasis *blockchain*. Ini akan menjadi dasar untuk merancang aplikasi-aplikasi IoT yang memiliki sekuritas optimal. Hanya, hal tersebut bukanlah akhir dari integrasi kedua teknologi. Oleh karena teknologi *blockchain* pun masih memiliki kendala-kendala seperti proses penyelesaian transaksi yang masih lambat dibandingkan teknologi yang telah ada sekarang ini, juga apabila dihubungkan dengan jaringan berbayar, artinya setiap transaksi pertukaran data akan melibatkan sejumlah nilai uang tertentu.

BAB 2

KAJIAN TEORI

2.1. Diagram Blok

Modul klien adalah berupa mikrokontroler yang memiliki kemampuan untuk melakukan *signing* terhadap transaksi *blockchain*. Agar dapat berinteraksi ke jaringan Ethereum, perlu adanya *nodeRemote Procedure Call* (RPC). Dalam sistem ini, akan menggunakan Infura. Infura akan menjembatani data dari modul klien ke jaringan Ethereum. Data transaksi tersebut kemudian akan diverifikasi oleh *Ethereum Virtual Machine* (EVM) yang terdapat di dalam jaringan TestNet. Hasil verifikasi kemudian dapat dimonitor melalui situs **etherscan.io**. Gambar 2.1 memperlihatkan diagram blok sistem yang dirancang.



Gambar 2.1. Diagram Blok Sistem

2.2. *Internet of Things*

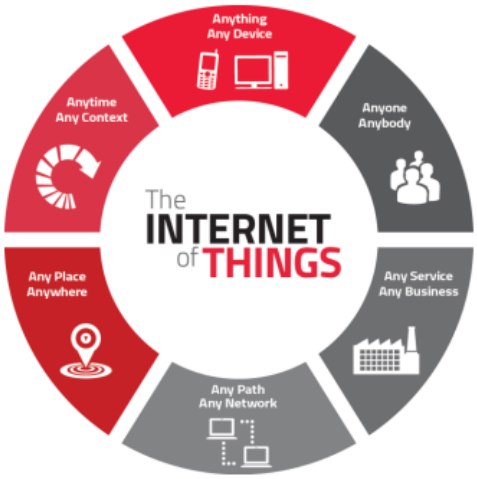
2.2.1. Konsep Dasar

Internet of Things (IoT) menggambarkan interkoneksi antar objek dalam jaringan Internet. Ini merupakan revolusi baru dari Internet. Objek-objek yang terhubung secara berkala akan dikumpulkan datanya, dianalisis dan digunakan untuk menghasilkan aksi, perencanaan, dan pengambilan keputusan [6]. IoT memungkinkan objek untuk saling mengenali dan berkomunikasi. Konsep hubungan di mana saja, kapan saja, apa saja dan siapa saja merupakan konsep

dasar dari IoT. Gambar 2.2 memperlihatkan konsep tersebut. Dapat disimpulkan, IoT merupakan jaringan *worldwide* dari objek-objek yang saling terhubung dan secara unik dapat diidentifikasikan.

Setiap peralatan dan orang dapat saling terhubung kapan saja dan di mana saja, melalui jaringan yang tanpa terbatas, juga dapat mengakses layanan yang beraneka ragam. Konsep ini memudahkan interkonektivitas antar-hal. Di sini membicarakan peralatan dari semua jenis, tidak terbatas pada komputer dan sejenisnya atau *smartphone*. Konsep ini mengubah Internet menjadi tempat keterhubungan berbagai hal dengan aksestabilitas yang tinggi. Sesuai dengan konsep tersebut, IoT memiliki beberapa karakteristik dasar, yaitu interkonektivitas, heterogenitas, perubahan dinamis, skala besar, dan keamanan. Menurut *framework* konseptual 2020, IoT dapat dinyatakan dengan persamaan berikut:

IoT = Servis + Data + Jaringan + Sensor [7]



Gambar 2.2. Konsep IoT [5]

Ada empat teknologi kunci yang digunakan dalam IoT [8], yaitu:

- Untuk penanda objek: teknologi RFID,

- Untuk pengindera objek: teknologi sensor,

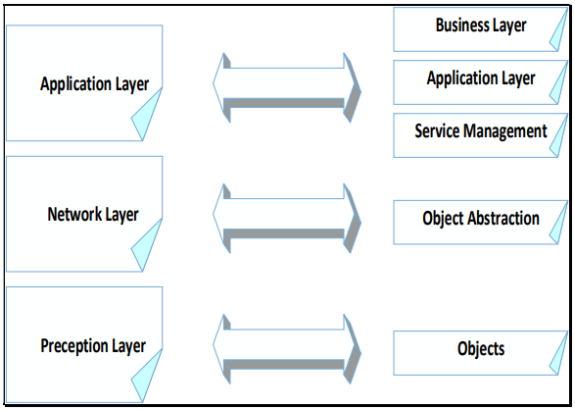
- Untuk kecerdasan: teknologi cerdas,

- Untuk memperkecil ukuran: teknologi nano.

Keempat teknologi kunci ini adalah merupakan teknologi yang digunakan untuk mencapai tiga pilar dalam IoT. Ketiga pilar tersebut adalah interkoneksi antar objek berupa identifikasi, komunikasi dan interaksi.

2.2.2. Arsitektur IoT

Konsep dasar IoT, ketiga pilar dan teknologi kunci dalam IoT memberikan gambaran bahwa arsitektur IoT pada dasarnya terdiri dari beberapa lapisan teknologi yang saling mendukung. Arsitektur ini memberikan gambaran bagaimana teknologi-teknologi saling berkaitan satu dengan yang lain dalam pengembangan IoT dalam berbagai skenario. Arsitektur IoT dapat digambarkan dalam tiga lapisan atau dalam bentuk pengembangan lima lapisan seperti pada Gambar 2.3 berikut.



Gambar 2.3. Arsitektur IoT [8]

Arsitektur tiga lapisan ini merupakan arsitektur dasar untuk IoT. Selain arsitektur tiga lapisan dan lima lapisan, ada pula arsitektur yang terdiri dari empat lapisan. Lapisan keempat ini biasanya digambarkan sebagai lapisan pendukung (misalnya komputasi awan, dan lain sebagainya). Lapisan keempat diletakan di antara lapisan persepsi dan jaringan pada arsitektur konvensional.

2.3. *Blockchain*

2.3.1. Pendahuluan

Teknologi *blockchain* sebenarnya muncul sebelum Bitcoin. Hanya Bitcoin yang diusulkan Satoshi Nakamoto [10] yang mempopulerkan istilah *blockchain*. Bitcoin Satoshi Nakamoto menggunakan teknologi *blockchain* sebagai uang digital dan bukan media penyimpanan. Dengan demikian muncul istilah *cryptocurrency*. Bitcoin agak sulit dimanfaatkan lain selain sebagai mata uang digital, oleh karena dalam algoritmanya hanya menyediakan 40 B. Ini jelas agak tidak realistis sebagai media penyimpanan.

Setelah kemunculan Bitcoin dan naiknya popularitas *blockchain* dan *cryptocurrency*, muncul banyak *platform* sejenis. *Platform* ini menawarkan hal yang lebih dari sekedar mata uang digital. Mulai muncul ide pemanfaatan *blockchain* untuk aplikasi selain mata uang digital [4]. Salah satu yang cukup populer sekarang ini adalah Ethereum. Ethereum diperkenalkan tahun 2015 oleh Vitalik Buterin. Ethereum dikatakan lebih baik dari Bitcoin karena menawarkan fungsi yang dapat diprogram. Ethereum menawarkan *smart contract* yang dapat dieksekusi oleh Ethereum Virtual Machine (EVM). EVM mengenal bahasa Solidity yang merupakan bahasa yang digunakan untuk memprogram *smart contract*. Dengan potensi *smart contract*, mulai muncul istilah Decentralized Applications (Dapps). Dapps merupakan aplikasi yang tanpa memanfaatkan *server* terpusat seperti pada umumnya aplikasi *web* lainnya. Dapps menggunakan jaringan Ethereum atau *node* untuk mengeksekusi algoritmanya. Oleh sebab itu, Dapps disebut sebagai aplikasi yang tidak dapat dihentikan. Ini karena berdasarkan sifat natural dari *blockchain*.

2.3.2. Protokol

Blockchain menekankan *Proof of Works* (PoWs) sebagai cara untuk melakukan validasi. Validator menggunakan metode konsensus dan mempercayai *blockchain* yang paling panjang dalam jaringan [4]. *Blockchain* pada prinsipnya terdiri dari sekumpulan blok yang disimpan dalam bentuk *copy* oleh *nodes* dalam jaringan *blockchain*. Apabila ada penambahan blok baru, maka blok tersebut akan di-*update* keseluruh *nodes* dalam jaringan. Agar dapat ditambahkan, validasi dilakukan. Inilah yang dinamakan dengan PoWs.

Algoritma PoWs dijalankan secara lokal oleh validator atau *node*. Dalam hal jaringan Ethereum, dilakukan oleh EVM. Tugas dari algoritma ini adalah untuk mencari solusi terhadap enkripsi kriptografi dalam blok tersebut. Algoritma PoWs memiliki tujuan utama untuk mencegah penambahan blok baru oleh pihak lain dan menjamin agar setiap *nodes* mendapatkan *copyblock* yang konsisten [11].

2.3.3. Struktur Data

Sesuai dengan namanya, *blockchain* merupakan sekumpulan blok yang saling berkaitan membentuk sebuah rantai atau *linked list*. Setiap blok terdiri dari *header* dan *body*. Dalam *header* dan *body* terdapat sekumpulan *fields*. Banyak dan isi dari *fields* adalah sesuai dengan spesifikasi protokol masing-masing *blockchain*.

Header biasanya terdiri dari kode Hash dan *nonce* yang merupakan *pointer* untuk blok berikutnya dan kode unik. Kedua kode tersebut merupakan solusi PoWs. Selain itu, juga terdapat data lain seperti waktu pembuatan dan *roots* dari pohon Merkle [12]. *Body* terdiri dari data transaksi yang memerlukan verifikasi. *Body* biasanya memiliki ukuran maksimum.

2.3.4. *Embedded System*

Embedded system merupakan sistem yang dapat berdiri sendiri atau menjadi bagian dari sistem lain, tetapi memiliki fungsi yang spesifik. *Embedded system* digunakan untuk aplikasi yang luas, mulai dari peralatan konsumen hingga aplikasi militer. Contoh *embedded system* adalah *mobile phone*, konsol *game*, DVD *player*, GPS, *mouse, printer*, *keyboard*, dan lain-lain. Peralatan-peralatan IoT juga dapat dikategorikan sebagai *embedded system*.

Oleh karena fungsi yang dilakukan spesifik, *embedded system* tidak memerlukan daya komputasi besar seperti menggunakan *personal computer* (PC), tetapi cukup menggunakan peralatan berdaya komputasi yang memenuhi dan hemat daya. Untuk hal ini, yang sesuai adalah mikrokontroler.*Embedded system* dapat berupa sistem yang tanpa memiliki antar-muka sama sekali hingga sistem dengan antar-muka yang kompleks menyerupai PC.

2.3.5. Mikrokontroler

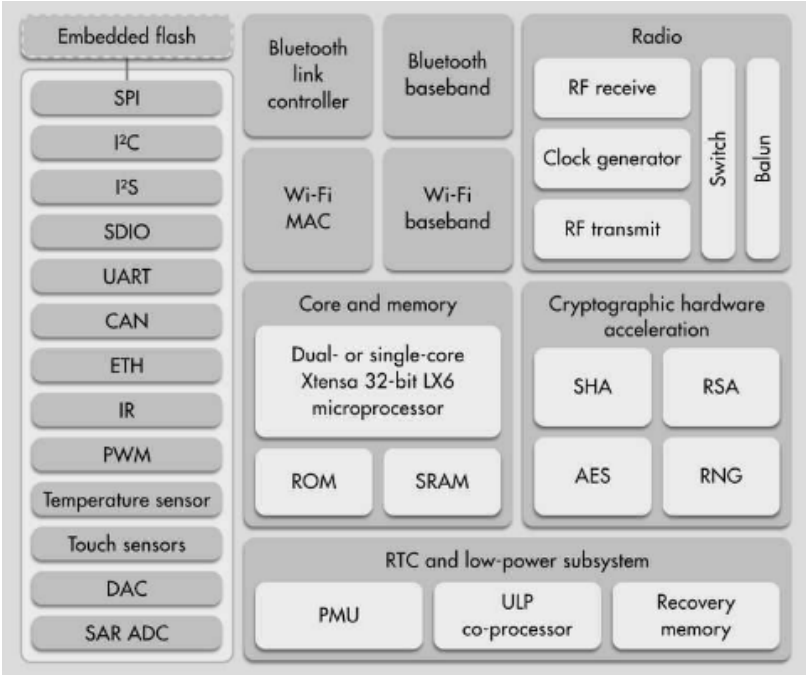
Mikorkontroler merupakan *integrated chip* (IC) yang kecil, murah dan berdaya rendah. Mikrokontroler memiliki daya komputasi yang terbatas dibandingkan mikroprosesor. Mikrokontroler biasa dilengkapi dengan prosesor 8 atau 16-bit, RAM, PROM atau memori *flash*, I/O baik paralel mau pun serial, *timers* dan/atau *counters*, *Analog to Digital Converter* (ADC) dan *Digital to Analog Converter* (DAC).

Contoh-contoh mikrokontroler adalah:

- ATMEGA 89C51, ATTiny,
- Zilog Z8, Z80,
- Motorola 68HC11 68HC12,
- AMD 29K,
- Microchip PIC,
- ESP 32,
- Dan lain-lain.

Masing-masing mikrokontroler sangat spesifik. Oleh karena dalam perancangan ini menggunakan mikrokontroler ESP32, maka pembahasan

kerangka teoritik ini akan difokuskan ke mikrokontroler ini. Gambar 2.4 memperlihatkan diagram blok untuk mikrokontroler ESP32. Di sini, dapat dilihat berbagai macam fungsi yang disediakan oleh mikrokontroler ini seperti I2C, IR, PWM, *Bluetooth*, WiFi, FR, dan lain-lain. Dari fungsi-fungsi ini, dapat disimpulkan bahwa ESP32 sesuai untuk kebutuhan IoT.



Gambar 2.4. Diagram Blok Mikrokontroler ESP32

Prosesor ESP32 menggunakan *dual-core*, kecuali tipe ESP32-S0WD menggunakan prosesor *single-core*. Frekuensi *clock* yang digunakan dapat mencapai 240MHz, frekuensi ini tergolong tinggi dibandingkan misalnya dengan frekuensi yang digunakan pada mikrokontroler ATMEGA8 yang hanya mencapai 12 MHz.

Prosesor ini juga dilengkapi dengan modul untuk koneksi WiFi dan *Bluetooth*. Modul ini biasanya jarang dimiliki oleh mikrokontroler. *Bluetooth* yang disediakan juga termasuk *Bluetooth Low Energy* (BLE), yang berdaya rendah dan dapat melakukan pengiriman data tertentu melalui *sleep mode*. Inilah salah satu faktor yang menyebabkan penggunaan *bluetooth* tipe BLE dapat menghemat daya.

Memori internal ESP32 terdiri dari ROM, SRAM dan *embedded flash*. ROM digunakan untuk proses *booting* dan besarnya adalah 448 KB, sedangkan SRAM digunakan untuk penyimpanan data dan instruksi. SRAM yang disediakan sebesar 520 KB. Selain jenis memori tersebut, ESP32 juga memiliki *embedded flash* yang terhubung via IO16, IO17, SC_CMD, SD_DATA_0 dan SD_DATA_1. Ini dapat memperbesar kapasitas penyimpanan data ESP32. Besaran memori *flash* adalah 2-4 MB. Selain memori internal, ESP32 juga dapat diekstensi dengan *external flash* dan SRAM

2.3.6. Penelitian-Penelitian Terdahulu

Beberapa penelitian yang telah dilakukan terkait dengan IoTdan *blockchain* adalah sebagai berikut:

- U. Guin, P. Cui dan A. Skjellum [13] membuat model sistem menggabungkan IoT dan *blockchain* dalam rangka menangani metode pembayaran mikro. Dalam penelitian ini, para peneliti mengusulkan arsitektur dan model yang sesuai.

- A. Dorri, et. al. [14] membuat studi kasus dan simulasi perihal pentingnya sekuritisasi IoT untuk aplikasi *smart home*. Para peneliti menggunakan *blockchain* untuk sekuritisasi dan privatisasi data.

- S. F. T. O. Mendonca, J. F. S. Junior dan F. M. R. Alencar [15] menjelaskan tentang tantangan yang dihadapi teknologi IoT berbasis *blockchain*. *Systematic Mapping* yang digunakan peneliti ini menemukan adanya *threads of validity*.

- J. Kogure, et. al. [16] memberikan gambaran tentang teknologi *blockchain* beserta aplikasi contohnya. Para peneliti juga mengusulkan alur dalam proses *cross-border transactions*.

- A. Dorri, et. al. [17] mengusulkan *Lightweight Scalable Blockchain* (LSB) untuk sekuritisasi dan privatisasi data IoT. Ini merupakan model teknologi yang dioptimasi untuk IoT.

Jadi, berdasarkan referensi yang ada ini, maka Tesis ini akan berfokus pada pengaplikasian teknologi *blockchain* untuk sekuritisasi data IoT.

BAB 3

PERANCANGAN

3.1. Gambaran Umum

Blockchain merujuk pada Ethereum untuk jaringan TestNet (Ropsten, Rinkerby atau Kovan). IoT merupakan singkatan dari *Internet of Things*. IoT berupa modul *hardware* yang memiliki kemampuan untuk terhubung ke jaringan Internet.

Sistem ini terdiri dari dua bagian, yaitu:

1. Bagian Klien,

2. Bagian *Server*.

Bagian Klien terdiri dari unit *hardware* dan unit *software*. Unit *hardware* terdiri berupa mikrokontroler dengan empat buah sensor, yaitu sensor suhu, kelembaban, tekanan dan posisi. Untuk sensor jarak, digunakan sebagai penanda lokasi. Sensor lainnya untuk mengumpulkan data sesuai dengan manfaat masing-masing sensor. Unit *software* terdiri dari program yang akan mengakuisisi data dari setiap sensor dan kemudian meneruskannya ke Bagian *Server*.

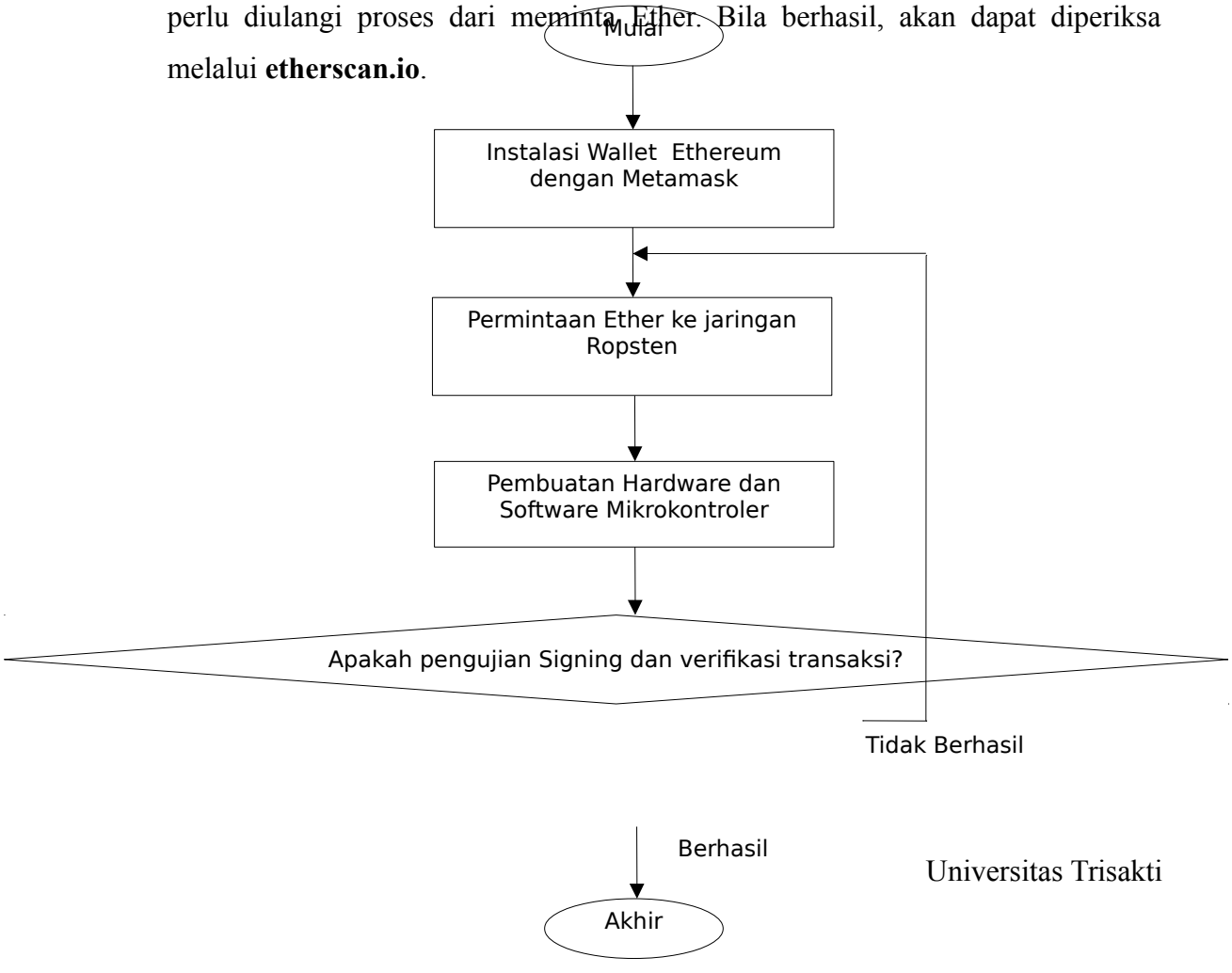
Bagian *Server* terdiri dari program smart contract yang menerima data-data sensor dan memprosesnya ke *Ethereum Virtual Machine* (EVM). Hasil pemrosesan dapat diperiksa melalui etherscan.io.

3.2. Diagram Alir Penelitian

Metodologi dalam Tesis ini adalah studi pustaka dan perancangan. Studi pustaka diperlukan untuk menentukan poin permasalahan dalam konsep integrasi teknologi *blockchain* dan IoT. Perancangan dilakukan sebagai upaya pembuktian

bahwa konsep tersebut dapat diwujudkan secara nyata sehingga pada akhirnya aplikasi IoT berbasis *blockchain* dapat diwujudkan. Diagram alir pada Gambar 3.1 berikut memperlihatkan metodologi yang digunakan.Studi pustaka dalam Tesis ini dilakukan dengan ekstensif karena kedua hal tersebut tergolong baru. Banyak hal masih sebatas konsep dan memerlukan eksplorasi lebih lanjut. Awal dari studi pustaka adalah perihal masing-masing teknologi tersebut, *Blockchain* dan IoT. Lalu melihat apakah ide memadukan kedua teknologi tersebut telah ada atau tidak.

Hal pertama yang dilakukan dalam penelitian ini adalah melakukan instalasi *wallet* Ethereum dengan menggunakan Metamask. Ini akan menjadi bagian pertama dalam melakukan interaksi dengan jaringan *blockchain* Ethereum. Dalam interaksi, diperlukan sejumlah nilai Ether sebagai biaya untuk proses. Oleh karena itu, setelah *wallet* terinstalasi, perlu meminta sejumlah nilai Ether ke jaringan TestNet. Jaringan TestNet yang digunakan adalah Ropsten. Selanjutnya, mikrokontroler dan program untuk melakukan *signing* serta *smart contract* perlu dibuat. Pengujian merupakan langkah berikutnya. Bila pengujian gagal, maka perlu diulangi proses dari meminta Ether. Bila berhasil, akan dapat diperiksa melalui **etherscan.io**.



Gambar 3.1. Metodologi Penelitian

3.3. Spesifikasi Perangkat

Sistem ini terdiri dari dua bagian, yaitu bagian perangkat keras dan bagian perangkat lunak. Bagian perangkat keras merupakan modul IoT, sedangkan modul perangkat lunak merupakan sistem pengendali modul IoT tersebut. Berikut adalah spesifikasi perangkat keras dan lunak sistem:

1. Dimensi modul perangkat keras: 5,5 x 8 x 10 cm^3,
2. Menggunakan catudaya dari *micro-USB*,
3. Menggunakan modul Wi-Fi,
4. Menggunakan modul sensor ketinggian, kelembaban, tekanan dan posisi.
5. Menggunakan *wallet* Ethereum,
6. Menggunakan infura.io untuk RPC,
7. Menggunakan EVM di jaringan TestNet Ropsten.

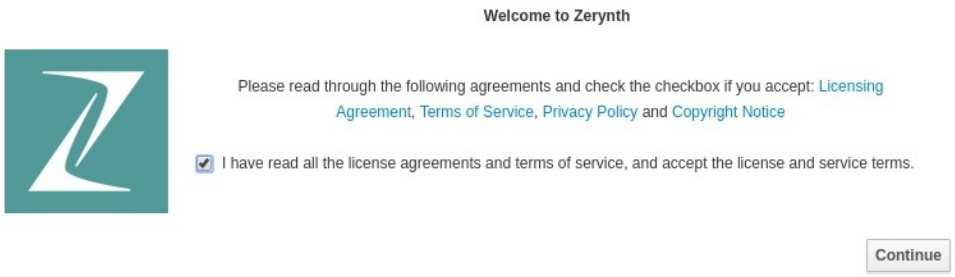
3.4. Perangkat Lunak

3.4.1. Zerynth Studio

Zerynth Studio merupakan perangkat lunak yang digunakan memrogram modul mikrokontroler ESP32 *Development Board* LOLIN D32. Bahasa Phyton

digunakan dalam pemrograman tersebut. Berikut adalah cara untuk melakukan instalasi Zerynth Studio:

1. *Download* Zerynth Studio versi terbaru di *link*www.zerynth.com/download. Sebagai catatan, Zerynth Studio hanya dapat diinstalasi pada *platform* 64-bit.
2. Klik dua kali pada *file* hasil *download* untuk menjalankan instalasi pada sistem operasi Microsoft Windows.
3. Pilih "Yes" untuk layar peringatan "allowing unknown publisher to make changes to this computer".
4. Setelah proses instalasi dimulai, akan muncul layar pemberitahuan seperti pada Gambar 5 berikut. Ikuti proses seperti pada Gambar 3.2.

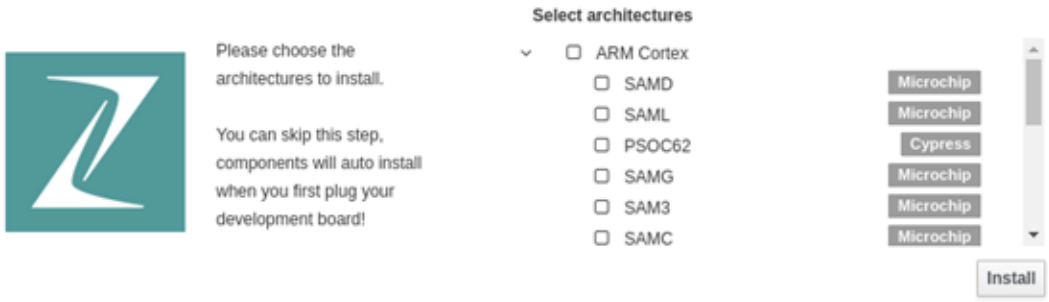


Gambar 3. 2. Persetujuan TOS dan Lisensi

5. Setelah disetujui, maka akan muncul pilihan untuk sumber instalasi. Pilihan ini terdiri dari dua, yaitu *online* dan *offline*. Pilihan ini diperlihatkan pada Gambar 3.3. Untuk pilihan *online*, akan muncul pilihan versi instalasi yang dapat dipilih. Untuk pilihan *offline*, tidak tersedia pilihan versi, tetapi pilihan ini memungkinkan *downloader* untuk dibagi dan lebih cepat dalam penyelesaian proses instalasi.

Gambar 3.3. Pilihan Instalasi

6. Berikutnya adalah memilih jenis arsitektur yang diinginkan. Untuk mempermudah proses instalasi, dapat langsung diklik tombol *Install*. Bila pilihan ini diambil, maka arsitektur baru akan diinstalasi apabila modul perangkat keras terpasang ke komputer tempat Zerynth Studio diintalasi. Gambar 3.4 memperlihatkan pilihan arsitektur yang didukung oleh Zerynth Studio.



Gambar 3.4. Pilihan Arsitektur

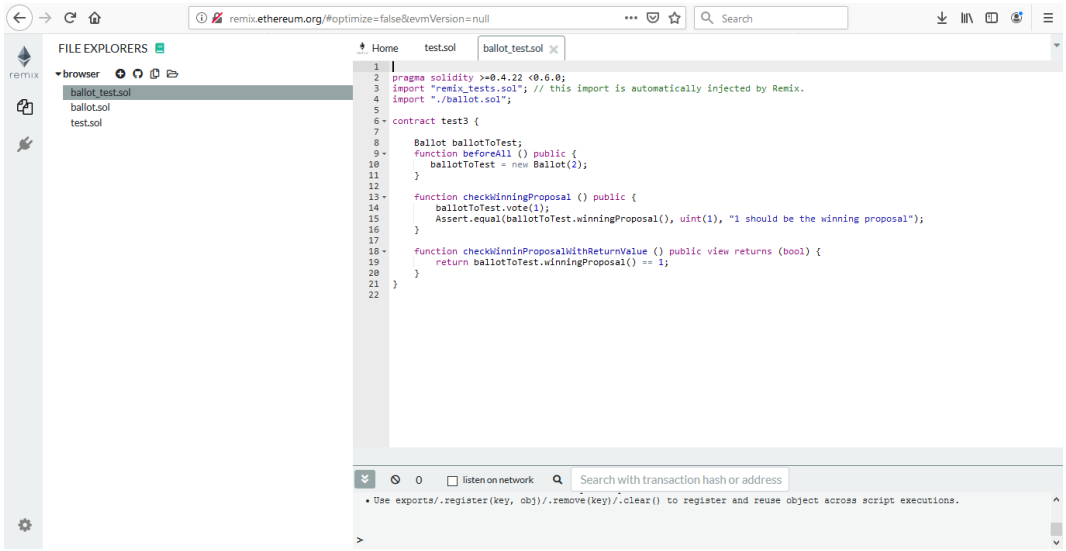
7. Proses instalasi akan berjalan sebagaimana diperlihatkan pada Gambar 3.5 berikut.

Gambar 3.5. Proses Instalasi Zerynth Studio

3.4.2. Remix

Remix merupakan *Integrated Development Environment* (IDE) untuk memprogram *smart contract* Ethereum. Remix ditulis dengan menggunakan

JavaScript. Remix tersedia secara *online* melalui *link* remix.ethereum.org. Di Remix, bahasa yang digunakan untuk memprogram *smart contract* adalah Solidity. Ini merupakan bahasa yang didukung oleh Ehtereum.



Gambar 3.6. IDE Remix

Smart contract Ethereum dapat diprogram, diuji, di-*debug* dan di-*deploy* dalam Remix ini. Jadi, programer tidak perlu berpindah-pindah aplikasi. Dengan menggunakan *plugins* Metamask untuk mengakses ke *Ethereum Virtual Machine*(EVM), Remix mampu men-*deploysmart contract* ke *node* mana pun yang dipilih. Gambar 3.6 memperlihatkan IDE Remix. Pada gambar ini, dapat dilihat pada sisi kiri panel adalah panel *File Explorers*. Programer dapat memilih jenis *file* yang ingin digunakan. Isi *file* pilihan akan muncul pada panel di kanan. Di panel kanan inilah, programer menuliskan *smart contract*-nya dengan bahasa Solidity.

Panel kanan Gambar 3.6 memperlihatkan contoh program Solidity dengan nama *file* ballot_test.sol. Ekstensi *file* sol menunjukkan bahwa program tersebut ditulis dengan bahasa Solidity. Pada program ini, terlihat bahwa kompiler yang digunakan adalah versi 0.4.22. Versi kompiler ini penting dinyatakan karena akan mempengaruhi instruksi yang digunakan. Sebagai contoh, untuk kompiler versi

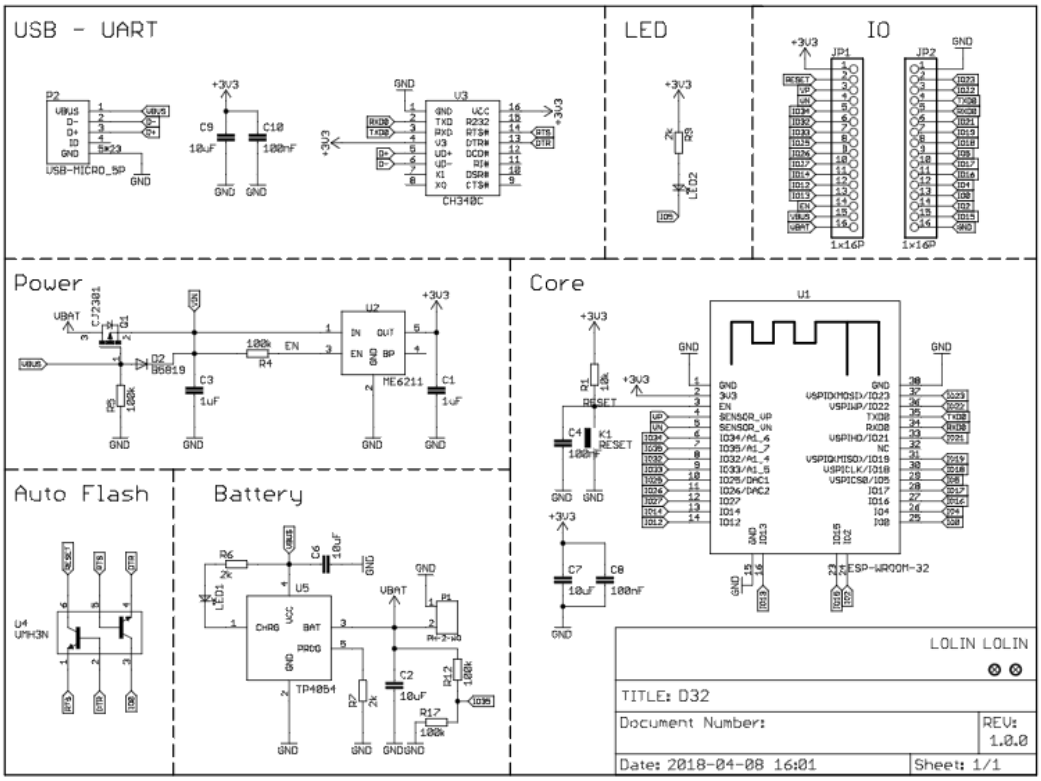
yang lebih rendah dari versi ini, penggunaan **constructor** masih didukung, tetapi untuk versi ini, hal tersebut tidak didukung lagi atau *deprecated*. Di program ini, juga diperlihatkan nama kontrak saat di-*deploy* di *node*. Untuk program ini, namanya adalah test3.

3.5. Perangkat Keras

3.5.1. LOLIN D32

Lolin D32 merupakan *development board* yang dikeluarkan oleh Wemos. Perangkat ini menggunakan mikrokontroler ESP32. Lolin D32 menggunakan sistem yang dinamakan *Espressif* dan merupakan sistem yang berdaya rendah. *Development board* ini memiliki fitur antara lain mikrokontroler *dual core* berbasis ARM, Wi-Fi, *Bluetooth*, I2C, I2S, SPI, ADC, DAC dan memori *flash* 4 MB.

Perangkat ini mendukung bahasa program MicroPhyton dan juga kompatibel dengan Arduino. Gambar 3.7 memperlihatkan diagram skematik dari Lolin D32. Dari skematik ini, dijelaskan bahwa *core* dari perangkat ini adalah ESP-WROOM-32 yang merupakan mikrokontroler *dual core* berbasis ARM. Selain itu, perangkat ini juga dilengkapi dengan *micro*-USB UART, sebuah *blinking* LED untuk menandakan proses sedang terjadi, dan pin-pin IO.



Gambar 3.7. Diagram Skematik Lolin D32

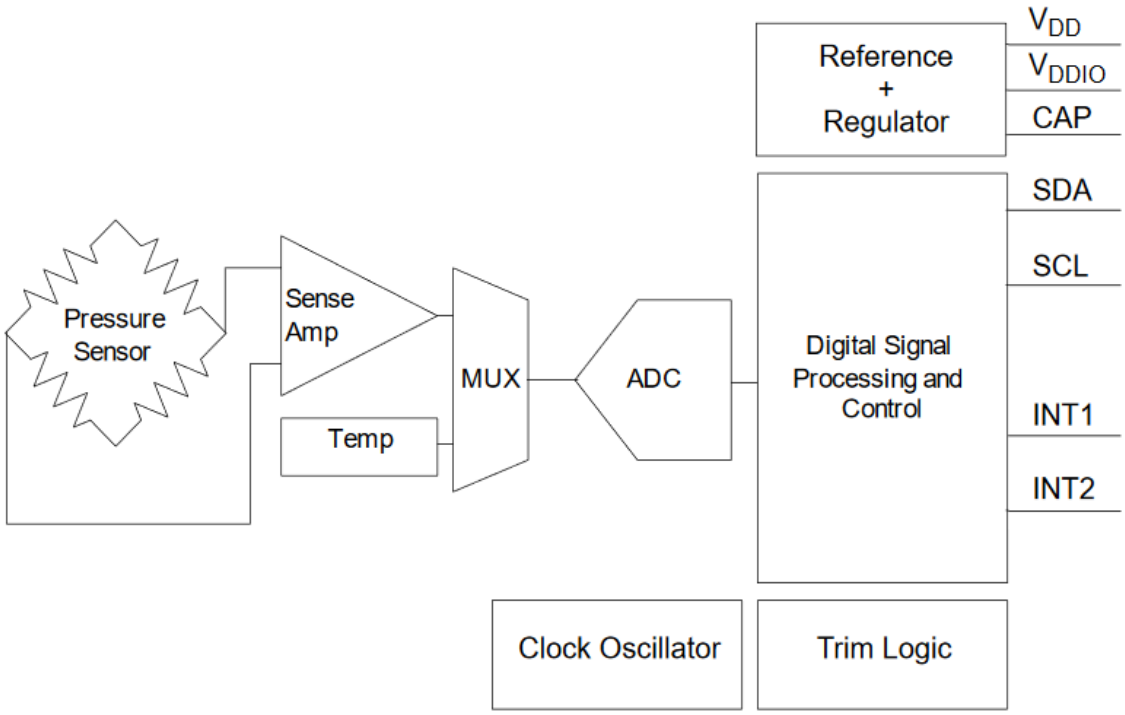
3.5.2. Modul Sensor MPL3115A2

Modul sensor yang digunakan adalah modul sensor altimeter MPL3115A2. Modul ini merupakan keluaran dari Freescale. MPL3115A2 memiliki sensor-sensor untuk mengukur parameter temperatur, tekanan dan ketinggian.Gambar 3.8 memperlihatkan diagram blok untuk modul sensor ini, dan Gambar 3.9 memperlihatkan koneksi pin pada modul tersebut. Berikut adalah resolusi untuk masing-masing parameter yang diukur [https://cdn.sparkfun.com/datasheets/Sensors/Pressure/MPL3115A2.pdf]:

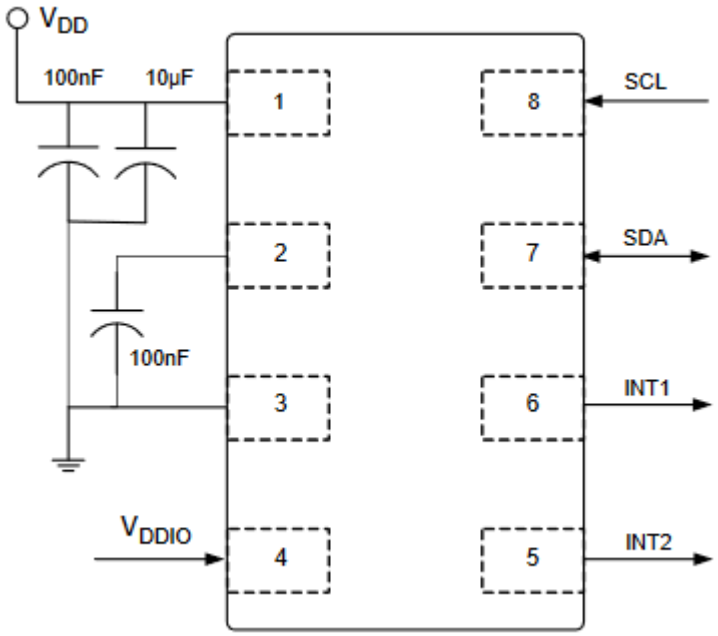
- Temperatur : ± 1°C (maksimum 3°C),
- Tekanan : 1,5 Pa,
- Ketinggian : 0,3 m.

Untuk jangkauan pengukuran masing-masing parameter tersebut adalah sebagai berikut:

- Temperatur : -40°C hingga 85°C,
- Tekanan : 20 hingga 110 kPa.



Gambar 3.8. Diagram Blok MPL3115A2



Gambar 3.9. Koneksi Pin MPL3115A2

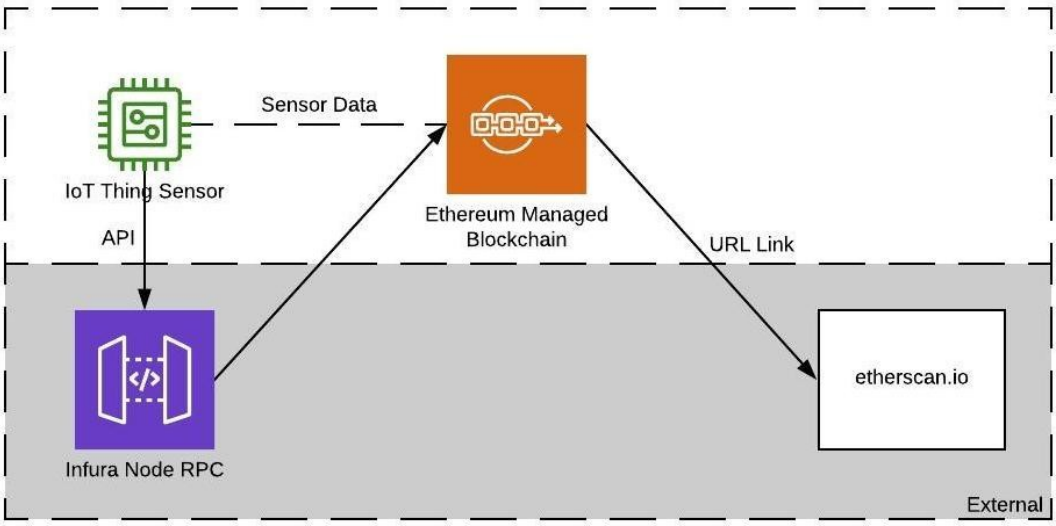
3.5.3. Kabel *Micro* USB

Kabel ini merupakan kabel data yang terdiri dari dua jenis konektor yaitu USB dan *micro* USB. Konektor USB dihubungkan ke komputer, sedangkan konektor *micro* USB ke perangkat/modul. Gambar 3.10 memperlihatkan kabel *micro* USB.

Gambar 3. 10. Kabel *Micro* USB

3.6. Perancangan Sistem

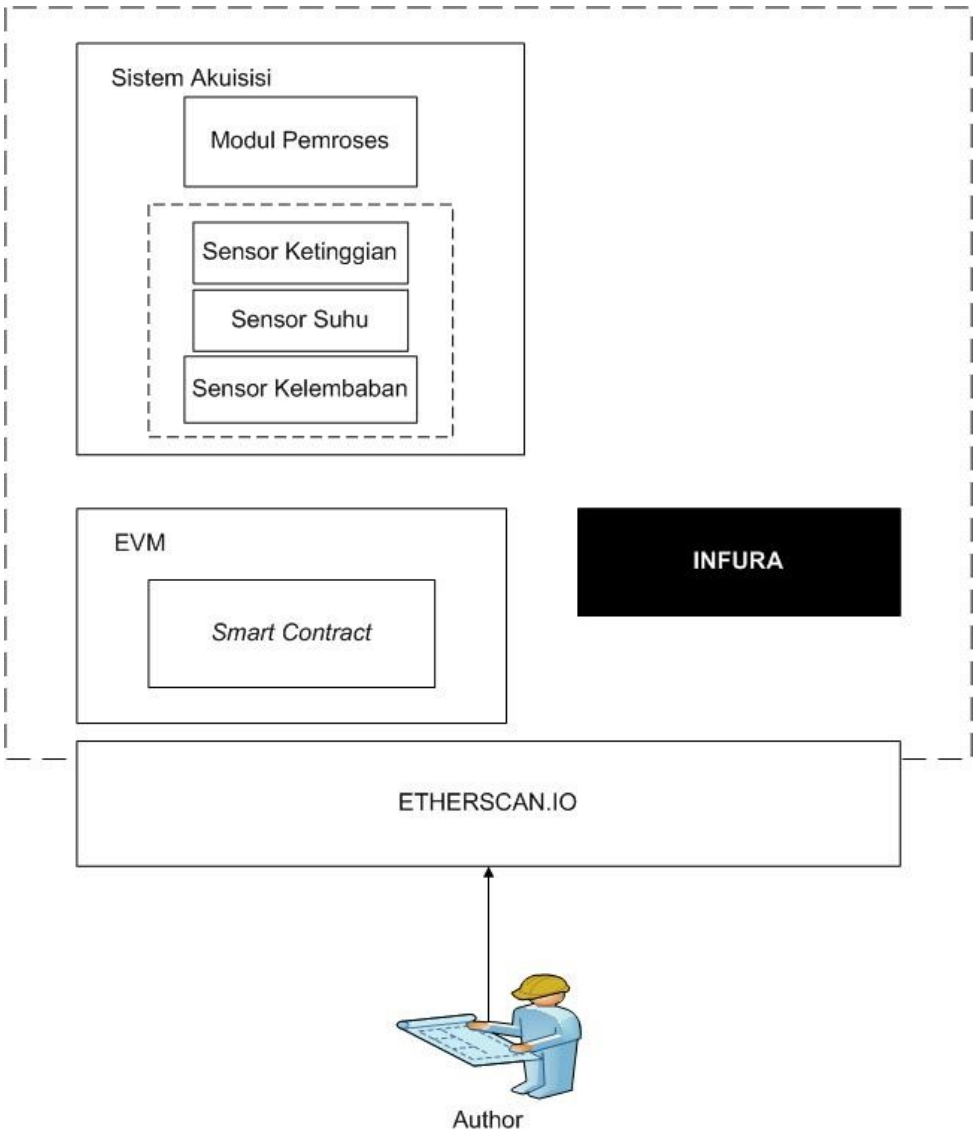
3.6.1. Arsitektur Sistem



Gambar 3.11. Arsitektur Sistem

Gambar 3.11 memperlihatkan Arsitektur sistem yang dirancang. Arsitektur sistem memperlihatkan bahwa sistem terdiri dari dua bagian, yaitu bagian eksternal dan bagian internal. Bagian eksternal berupa **infura.io** dan **etherscan.io**. Infura akan menghubungkan modul IoT Thing Sensor (perangkat keras) dengan EVM (tempat *smart contract* berada) dengan menggunakan *Remote Procedure Call* (RPC). Etherscan merupakan situs yang digunakan untuk membantu melihat hasil verifikasi oleh EVM. Bagian internal berupa IoT Thing Sensor dan *smart contract* yang di-*deploy* di EVM.

3.6.2. *System Environment*



Gambar 3.12. *System Enviroment*

Gambar 3.12 ini memperlihatkan *system environment*. Pada gambar, terlihat bahwa *Author* atau *Client* atau perancang sistem atau dapat juga *user* berinteraksi dengan **etherscan.io** untuk memeriksa validitas transaksi. *Author* akan memasukan *Transaction Hash* sebagai identitas transaksi atau memasukkan alamat kontrak (Subbab III.7) bila ingin memeriksa keseluruhan transaksi yang terjadi. Inilah satu-satunya interaksi antara sistem dengan *Author*.

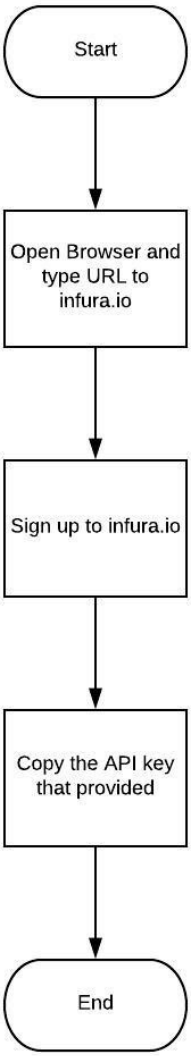
Dalam sistem sendiri, seperti tergambar pada Gambar 3.11 tentang Arsitektur Sistem, sistem terdiri dari dua bagian. Bagian yang merupakan bagian internal adalah Sistem Akuisisi. Sistem ini terdiri dari perangkat keras berupa modul Lolin D32 dan perangkat Program Transaksi (Subbab III.8). Selanjutnya dari Sistem Akuisisi menghubungi **infura.io** melalui RPC sesuai dengan *link* yang telah diprogram. Transaksi diteruskan dengan cara demikian ke *smart contract* pada jaringan Ropsten.

3.6.3. Diagram Alir Sistem

Pada Subbab ini akan dibahas perihal diagram alir yang diperlukan untuk mendukung sistem yang dirancang. Secara garis besar, terdiri dari tiga diagram alir, yaitu:

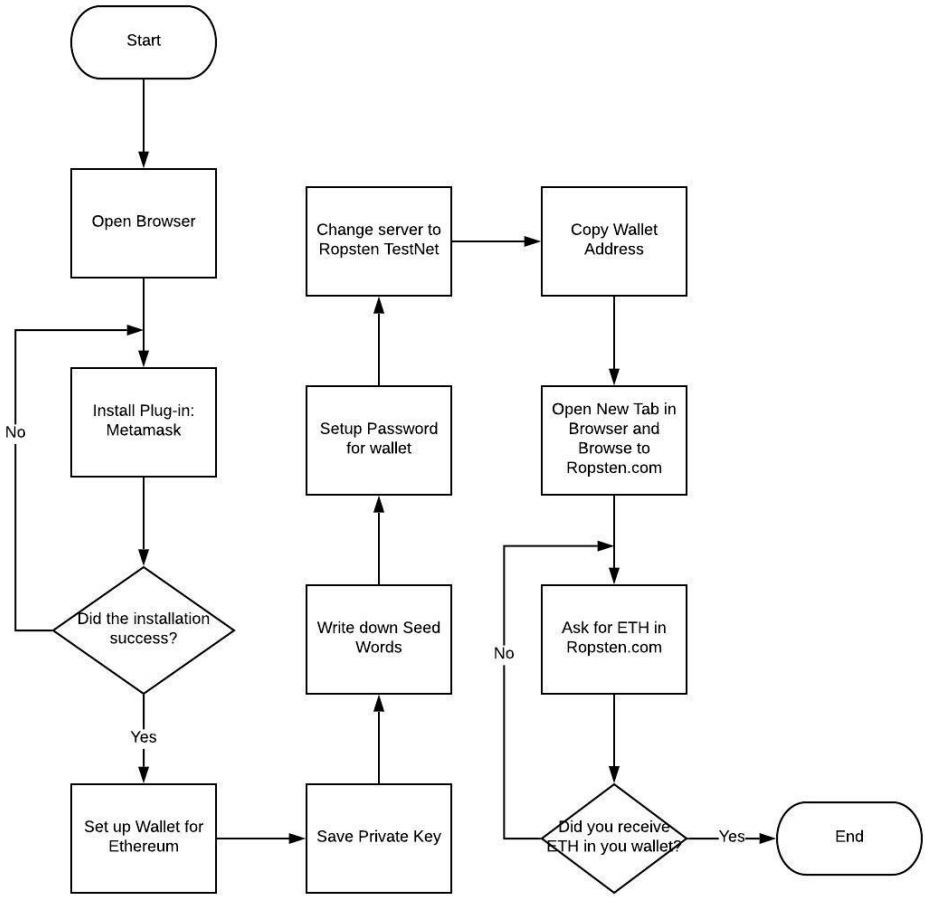
1. Diagram alir yang memperlihatkan prosedur untuk mendapatkan API *key* di **infura.io**,
2. Diagram alir untuk melakukan instalasi *wallet* dengan menggunakan *plugins* Metamask,
3. Diagram alir untuk memeriksa validitas transaksi di **etherscan.io**.

API *key* dari **infura.io** diperlukan bagi perangkat lunak program transaksi (Subbab 3.7.2) untuk melakukan interaksi dengan jaringan Ropsten. Gambar 3.13 berikut memperlihatkan prosedur untuk mendapatkan API *key* ini.Gambar 3.14 memperlihatkan diagram alir untuk membuat *wallet* Ethereum dengan *plugins* Metamask pada *browser* Chrome. Tahap pertama dalam prosedur ini adalah melakukan instalasi *plugins* Metamask. Tahap berikutnya adalah membuat akun baru atau alamat *wallet* baru. Ini dilakukan dengan beberapa tahap yang dijelaskan dalam diagram alir. Tahap akhir dari prosedur ini adalah mendaftar ke **ropsten.com** untuk mendapatkan sejumlah Ethereum agar dapat melakukan transaksi.

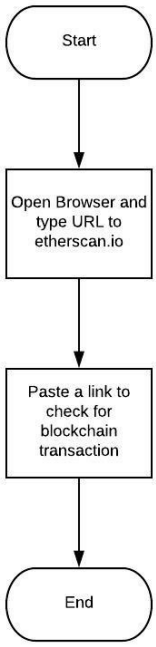


Gambar 3.13. Diagram Alir Prosedur Mendapatkan API Key

Diagram alir terakhir yang diperlukan untuk mendukung sistem adalah diagram alir yang menjelaskan proses pemeriksaan data hasil verifikasi transaksi. Setiap proses transaksi mengandung identitas spesifik yang dinamakan *Transaction Hash*. Identitas ini dapat diperiksa di **etherscan.io**. Gambar 3.15 memperlihatkan diagram alir untuk melakukan prosedur tersebut.



Gambar 3.14. Diagram Alir Prosedur Pembukaan *Wallet* dan Mendapatkan ETH



Gambar 3. 15. Diagram Alir Cara Memeriksa Hasil Verifikasi

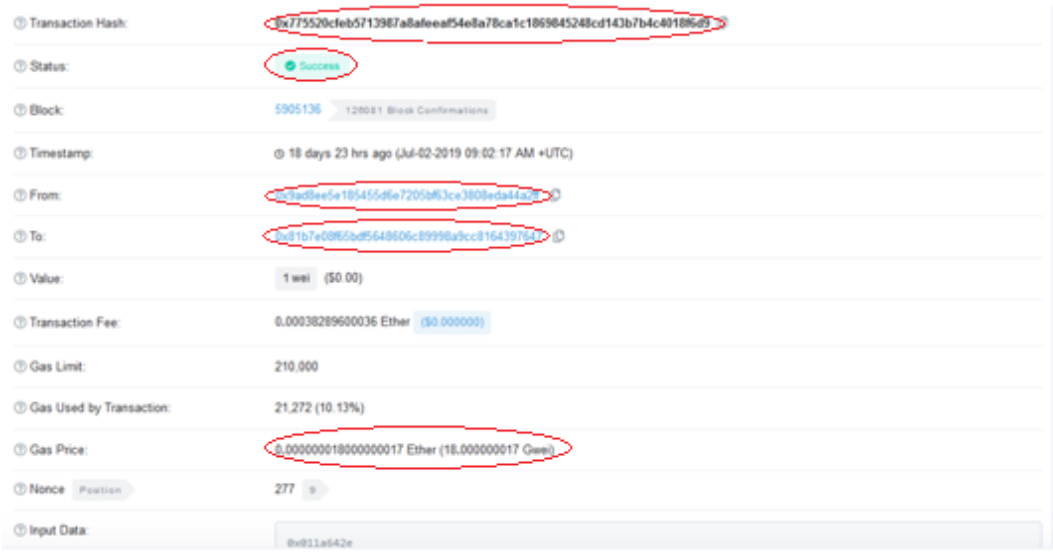
3.7. Perancangan *Smart Contract*

Smart Contract yang dirancang menggunakan Remix dengan bahasa pemrograman Solidity. Kompiler yang digunakan adalah versi 0.4.22. *Smart contract* ini dirancang agar dapat menerima data sensor dari Lolin D32, yaitu berupa data sensor posisi, ketinggian, tekanan, dan temperatur. *Smart contract* ini *live* di jaringan Ropsten. Jaringan ini merupakan jaringan untuk melakukan percobaan sebelum *live* di jaringan utama Ethereum. Dengan menggunakan jaringan Ropsten, dapat dilihat biaya yang mungkin diperlukan dalam melakukan transaksi (dalam GAS yang dapat dikonversi ke nilai ETH atau USD), dan juga untuk menguji apakah *smart contract* mengandung kesalahan atau tidak.

Data-data ini diterima sebagai *state variable*. Nilai variabel tersebut dinyatakan dalam nilai heksadesimal. Dalam kontrak ini juga dinyatakan besarnya nilai GAS (satuan untuk biaya transaksi) maksimum yang perlu dibayarkan untuk menvalidasi transaksi. Transaksi di sini dimaksudkan sebagai pengiriman data-data sensor tersebut dari perangkat keras ke EVM. Contoh transaksi pada *smart contract* ini dapat dilihat pada Gambar 3.16 berikut. Pada gambar ini, terlihat

hash transaksi yang merupakan kode unik untuk mengidentifikasi setiap transaksi yang terjadi. Di sini juga terlihat kolom "Ke" (*To*) ini merupakan kolom yang memperlihatkan alamat dari kontrak tersebut, yaitu 0x81b7e08f65bdf5648606c89998a9cc8164397647. Alamat ini berupa nilai heksadesimal unik yang bisanya dinamakan alamat *wallet*. Selain itu, juga ada kolom "Dari" (*From*). Ini merupakan alamat *wallet* dari mana transaksi ini berasal. Jadi, alamat ini dapat dikatakan sebagai alamat yang spesifik untuk tiap perangkat atau identitas tiap perangkat.

Pada gambar ini juga diperlihatkan biaya transaksi, yaitu biaya yang diperlukan untuk mendaftarkan nilai-nilai sensor ke jaringan Ropsten. Biaya ini adalah sebesar 0.000000018000000017 ETH(berdasarkan nilai referensi dari www.tukarku.com pada tanggal 21 Juli 2019 jam 18.12 adalah sebesar Rp. 3.166.218,04 per 1 ETH). Ini berarti biaya transaksi ini adalah sebesar 0.000000018000000017 x 3.166.218,04 = Rp. 0,057. Kolom terakhir yang tertera pada gambar ini adalah *Input Data*. Ini merupakan data yang dicatatkan dari perangkat. *Smart contract* dapat dilihat secara lengkap pada lampiran. Selain kolom-kolom yang telah dijelaskan di atas, juga terdapat satu kolom yang penting yaitu kolom *Status*. Kolom ini memperlihatkan kondisi apakah transaksi telah berhasil dicatat ke jaringan Ropsten atau tidak. Kondisi untuk kolom ini ada dua, yaitu *pending* dan *success*.



Gambar 3. 16. Transaksi pada *Smart Contract*

3.8. Perancangan Program Transaksi

3.8.1. Klasifikasi Berdasarkan *Use Case*

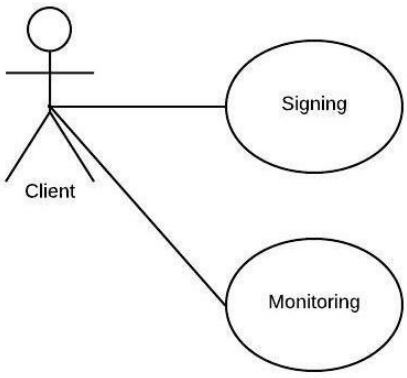
Berdasarkan Subbab 3.6.2 tentang *System Environment*, aktor dalam sistem ini adalah *Author* atau *Client* atau seperti dijelaskan pada subbab tersebut. *Use Case* ditinjau dari sudut pandang aktor ini, dan juga diperluas dari sudut pandang aktor yang berupa subsistem, yaitu *Verifier* dan *Signer*. Berikut adalah aktor-aktor yang ditinjau dalam klasifikasi *Use Case* perancangan sistem:

- *Author* atau *Client*,
- *Verifier*,
- *Signer*.

Use Case dari sudut pandang *Author* memiliki dua jenis kasus, yaitu *Signing* dan *Monitoring*. *Use Case* ini dapat dilihat pada Gambar 3.17 berikut. Kedua *Use Case* dapat dijelaskan sebagai berikut:

- *Signing* merupakan proses enkripsi data dengan menggunakan *private key* yang dimiliki oleh *client/author*.

- *Monitoring* merupakan proses pemonitoran terhadap hasil data yang telah diverifikasi di *Ethereum Virtual Machine* (EVM).



Gambar 3.17. *Author* atau *ClientUse Case*

```
alt_send = int(alt)

pres_send = (pres & 0x000000FF) << 8

temp_send = int(temp)  << 16

ir_send = digitalRead(D4) << 24

data_send = (ir_send | temp_send | pres_send | alt_send)

ethtrans.set_data(data_send)

ethtrans.set_nonce(nt)

ethtrans.sign(config.PRIVATE_KEY)
```

Gambar 3.18. Program *Signing*

Untuk *use caseSigning*, merupakan proses pertama dalam memproses data ke jaringan *Blockchain*. Program *Signing* diperlihatkan pada Gambar 3.18. Keseluruhan baris perintah sebelum yang terakhir memperlihatkan bagaimana objek transaksi dipersiapkan. Perintah **ethtrans.sign(config.PRIVATE_KEY)** merupakan instruksi untuk melakukan *signing* dengan menggunakan *private key*yang telah dikonfigurasikan terlebih dahulu. Ada pun nilai *private key* yang digunakan adalah 0x33627590e0b8751ddabec6d0278c1e5f7da1821c9eb2a7b06f5d0edb3ec579ae, yang merupakan *private key* untuk *smart contract*.

1. *Client/Author* mendaftar *wallet* dengan menginstalasi *plug-in* Metamask.

2. *Client/Author* meminta ETH di Ropsten.

3. *Smart contract* telah di-*deploy* ke EVM dan mendapatkan *address*.

4. *Wallet* dan *Smart contract address* didaftarkan ke program.

Untuk *use caseMonitoring*, merupakan proses terakhir dalam ke seluruhan sistem ini.

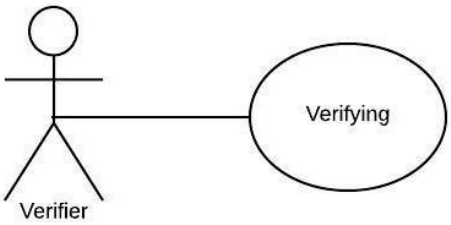
1. Metamask memberikan notifikasi bahwa verifikasi telah berhasil dilakukan minimal oleh satu *node*.

2. *Client/Author* meng-*copylink* yang dicetak oleh program.

3. *Client/Author* membuka *link* tersebut di *browser*.

4. *Client/Author* memonitor data-data sensor yang dikirim.

Use Case ditinjau dari sudut pandang *Verifier* hanya terdiri satu jenis kasus, yaitu *verifying*. Ini dapat dilihat pada Gambar 3.19.*Verifier* melakukan proses subsistem untuk melakukan verifikasi sesuai dengan *private key* yang dikirim oleh *Signer*.



Gambar 3.19. *VerifierUse Case*

Verifier berupa program *smart contract* yang berjalan di EVM di jaringan TestNet Ropsten. *Smart contract* ini telah di-*deploy* dan mendapatkan

alamat.*Verifier* akan menverifikasi integritas data untuk memastikan data yang diterima adalah data yang di-*sign* oleh *Signer*. Berikut adalah tahap dalam melakukan verifikasi:

1. *Verifier* menerima data melalui *state variables*.
2. Data yang diterima merupakan data yang telah diformat oleh proses *Prepairing*.

```
WIFI_SSID = '3Com'
WIFI_PASSWORD = ''
```

Gambar 3.20. *Username* dan *Password* WiFi yang Digunakan

```
# WiFi drivers
from espressif.esp32net import esp32wifi as net_driver # for ESP-32
# from broadcom.bcm43362 import bcm43362 as net_driver # for Particle Photon
from wireless import wifi
```

Gambar 3. 21. Inisiasi *Driver* WiFi

```
CA_CERT = """-----BEGIN CERTIFICATE-----

MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF

ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6

b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL

MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv

b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj

ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM

9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
```

```
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6

VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L

93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm

jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC

AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA

A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI

U5PMCCjjmCXPI6T53iHTflUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs

N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv

o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU

5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy

rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
```

Gambar 3.22. Sertifikat SSL yang Digunakan

```
SSL_CTX = ssl.create_ssl_context(
cacert=config.CA_CERT,
options=ssl.CERT_REQUIRED|ssl.SERVER_AUTH)
```

Gambar 3.23. Validasi Sertifikat SSL

Use Case ditinjau dari sudut *Signer* terdiri dari lima kasus, yaitu *Connecting*, *Initiating*, *Getting Information*, *Prepairing* dan *Sending Transaction*. Untuk *use caseConnecting*, sistem menghubungkan modul ke WiFi yang memiliki koneksi Internet. Secara bertahap, *use case* ini dilakukan sebagai berikut:

1. Inisiasi *driver* WiFi (Gambar 3.20 dan Gambar 3.21.).

2. Inisiasi https (Gambar 3.22 dan Gambar 3.23).

3. Menghubungkan ke WiFi dengan SSID dan *password* yang telah di-*setting*.

Untuk *use caseInitiating*: *use case*(diperlihatkan pada Gambar 3.24) ini memperlihatkan inisiasi ke *node* RPC. Ini dilakukan dengan cara:

1. *Set* url untuk RPC.

2. Inisiasi ke *node* RPC.

```
RPC_URL = 'https://ropsten.infura.io/v3/80859bb1f4c9416f96dffb0e97747e4a'

ADDRESS = '0x9Ad8Ee5E185455D6E7205bF63cE3808EDa44A2Ff'
PRIVATE_KEY =
'0x33627590e0b8751ddabec6d0278c1e5f7da1821c9eb2a7b06f5d0edb3ec579ae'
```

Gambar 3.24. Inisiasi RPC

Untuk *use case Getting Information* (diperlihatkan pada Gambar 3.25), sistem akan mendapatkan informasi mengenai jumlah ETH yang tersedia di *wallet*. Berikut adalah langkahnya:

1. Inisiasi alamat *wallet*.

2. Membaca nilai ETH.

3. Membaca nilai *gas*.

4. Membaca jumlah transaksi yang telah terjadi.

Untuk *use case Prepairing* (perhatikan Gambar 3.26), sistem akan mempersiapkan objek agar transaksi dapat dilakukan ke jaringan TestNet. Langkah-langkah dalam melakukan *Prepairing* adalah sebagai berikut:

1. Inisiasi jumlah *wei*.

2. Inisiasi nilai *gas*.

3. Inisiasi batasan nilai *gas*.

4. Inisiasi *nonce*.

5. Inisiasi alamat *Smart Contract*.

6. Persiapan transaksi.

```
RECEIVER_ADDRESS = '0x81b7e08f65bdf5648606c89998a9cc8164397647'

balance = eth.getBalance(config.ADDRESS)
```

```
print("Balance:", balance)

    if not balance:

        print(eth.last_error)
        raise Exception
```

Gambar 3. 25. Penerapan *Use Case Getting Information*

```
ethtrans = ethereum.Transaction()
```

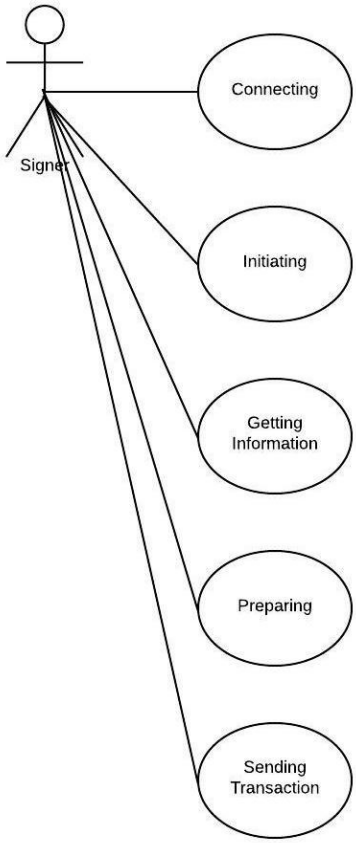
```
ethtrans.set_value(1, ethereum.WEI)

ethtrans.set_gas_price("0x430e23411")

ethtrans.set_gas_limit("0x33450")

ethtrans.set_receiver(config.RECEIVER_ADDRESS)

ethtrans.set_chain(ethereum.ROPSTEN)

ethtrans.set_data(data_send)
ethtrans.set_nonce(nt)
```

Gambar 3.26. Penerapan *Use Case Prepairing*

Untuk *use case Sending Transaction*, sistem mengirim transaksi atau data yang telah diformat ke jaringan TestNet sesuai alamat yang telah diinisiasi. Format data yang digunakan diperlihatkan pada Gambar 3.27, demikian juga penerapan dari *Use Case Sending Transaction* ini. Keseluruhan *Use Case Signer* digambarkan pada Gambar 3.28 ini. Setiap satu transaksi selesai dilakukan, maka modul Perangkat Keras perlu di-*reset*. *Reset* ini berguna untuk mempersiapkan siklus transaksi berikutnya. Sebelum metode *reset* dilakukan, proses perlu dihentikan untuk jangka waktu 30 detik. Nilai 30 detik didapatkan dari hasil percobaan. Untuk penghentian proses di bawah 30 detik mengakibatkan kegagalan dalam siklus transaksi berikutnya, sehingga transaksi gagal dikirimkan ke jaringan EVM.

```
data_send = (ir_send | temp_send | pres_send | alt_send)
print("Sending values... [", hex(data_send), "]")
eth_hash = eth.sendTransaction(ethtrans.to_rlp(True))
print("Monitor your transaction at: https://ropsten.etherscan.io/tx/%s"
% eth_hash)
print("Sent!")
digitalWrite(D5, HIGH)
sleep(30000)
mcu.reset()
```

Gambar 3.27. *Format* dan Pengiriman Data

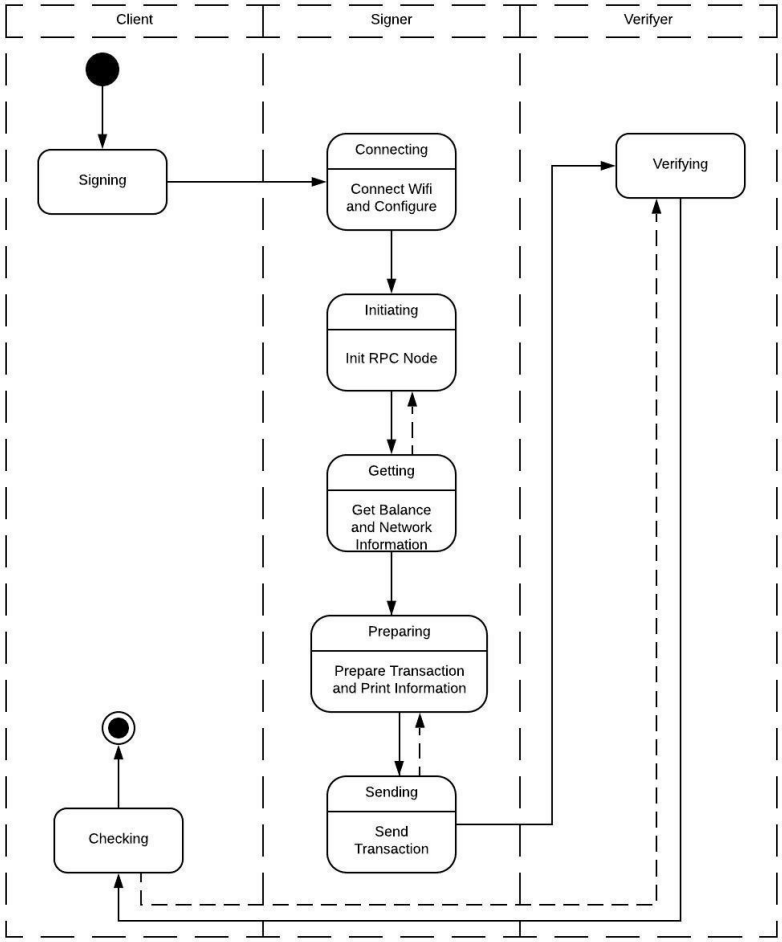


Gambar 3.28. *Use Case Signer*

3.8.2. Diagram Aktivitas

Diagram aktivitas merupakan diagram sejenis diagram alir yang memperlihatkan urutan proses dilakukan. Pada perancangan sistem ini, diagram aktivitas digunakan bersama dengan *swim lane*. Ini berarti diagram mampu menggambarkan urutan proses dan aktor yang berkaitan dengan proses tersebut. Gambar 3.29 memperlihatkan diagram aktivitas sistem yang dirancang. Proses dimulai oleh *Author* atau *Client* yang melakukan proses *Signing*. Kemudian dilanjutkan oleh subsistem *Signer* yang akan melakukan kelima prosesnya, yaitu *Connecting*, *Initiating*, *Getting Information*, *Prepairing* dan *Sending Transaction*.

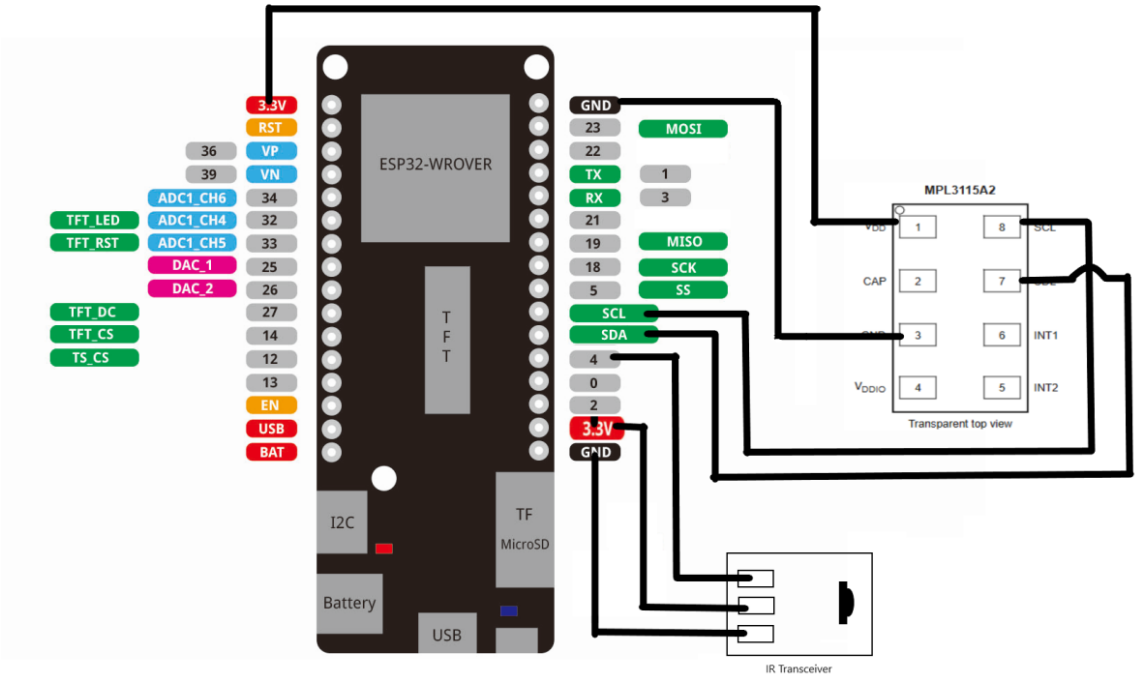
Selanjutnya adalah proses pada subsistem *Verifier* yang akan melakukan verifikasi terhadap transaksi. Hasilnya akan dikembalikan untuk dapat diperiksa oleh *Author* atau *Client*. Siklus ini melengkapi seluruh proses pada sistem.



Gambar 3.29. Diagram Aktivitas Sistem

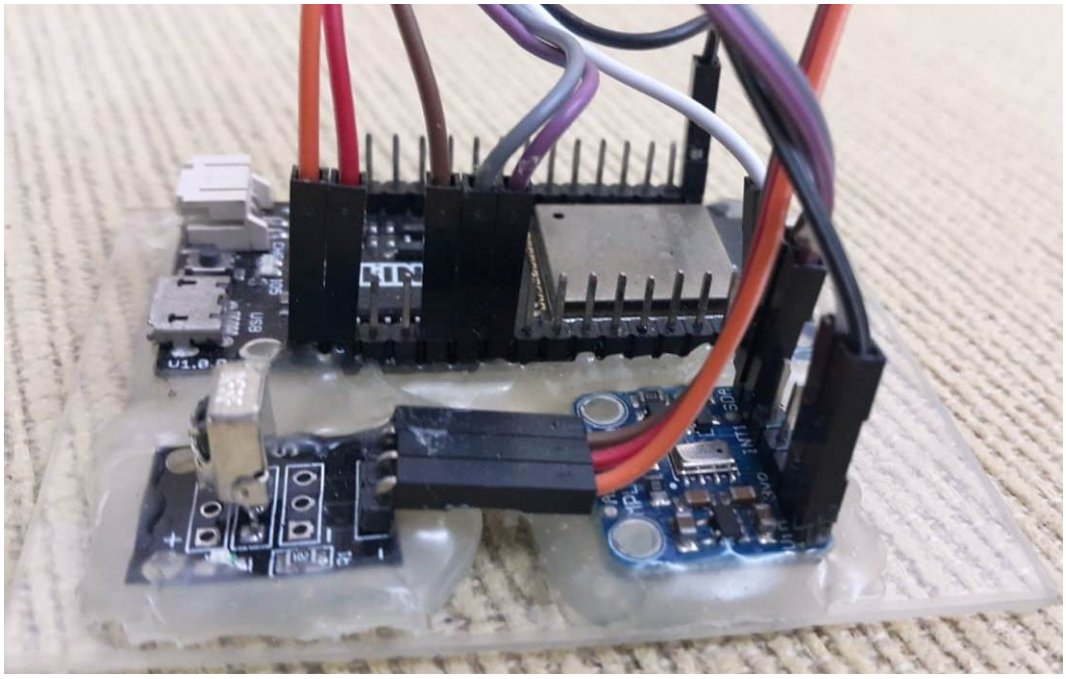
3.9. Perancangan Perangkat Keras

Seperti yang dijelaskan pada Subbab III.5, perangkat keras yang digunakan dalam perancangan sistem ini adalah perangkat dalam bentuk modul. Ini terdiri dari modul Lolin D32, modul MPL3115A2 serta kabel penghubung perangkat dengan komputer. Perangkat keras utama pada sistem ini adalah dua modul yang disebutkan pertama. Untuk kabel, diperlukan guna mengamati transaksi yang terjadi. Jadi, hanya diperlukan saat perancangan sistem saja.Gambar 3.30 memperlihatkan diagram skematik rancangan perangkat keras.Gambar 3.31 dan Gambar 3.32 memperlihatkan perangkat hasil rancangan.



Gambar 3. 30. Diagram Skematik Perangkat Keras

Gambar 3.31. Perangkat Keras Hasil Rancangan (Tampak Atas)



Gambar 3.32. Perangkat Keras Hasil Rancangan (Tampak Samping)

BAB 4
PEMBAHASAN

4.1. Pembahasan Modular Perangkat Lunak

4.1.1. Pengujian dan Analisis Modul *Signing*

Proses *Signing* berhasil apabila enkripsi data berhasil dilakukan dengan menggunakan *private_key* yang telah disimpan. Untuk sistem ini, *private_key* yang digunakan adalah 0x33627590e0b8751ddabec6d0278c1e5f7da1821c9eb2a7b06f5d0edb3ec579ae. Untuk mengetahui apakah proses ini berhasi atau tidak, pada program ditambahkan baris perintah setelah perintah untuk *signing* yang dapat dilihat pada Gambar 4.1.

```
Print ("Begin of Signing Process..")
Print ("Signing Completed..")
```

Gambar 4.1. Pemeriksaan Keberhasilan Proses *Signing*

Apabila proses ini berhasil maka pada layar *console* proses akan muncul kalimat "Signing Completed..". Awal proses *Signing* ditandai dengan kalimat "Begin of Signing Process..". Di antaranya adalah proses *signing* tersebut. Hasil pengujian berhasil apabila kedua kalimat ini tampil saat pengujian diadakan. Gambar 4.2 memperlihatkan hasil pengujian untuk transaksi yang ke-294.

```
Begin of Signing Process..
Signing Completed...
```

Gambar 4.2. Hasil Pengujian Proses *Signing*

4.1.2. Pengujian dan Analisis Modul *Monitoring*

Proses*monitoring* merupakan proses pemeriksaan validitas transaksi. Ini berarti proses untuk memastikan transaksi telah tercatat di *node*. Dalam sistem ini,

transaksi telah tercatat di alamat tujuan pada jaringan Ropsten. Proses *monitoring* sistem dilakukan dengan mencetak *link* pemeriksaan dengan instruksi yang ditampilkan pada Gambar 4.3.

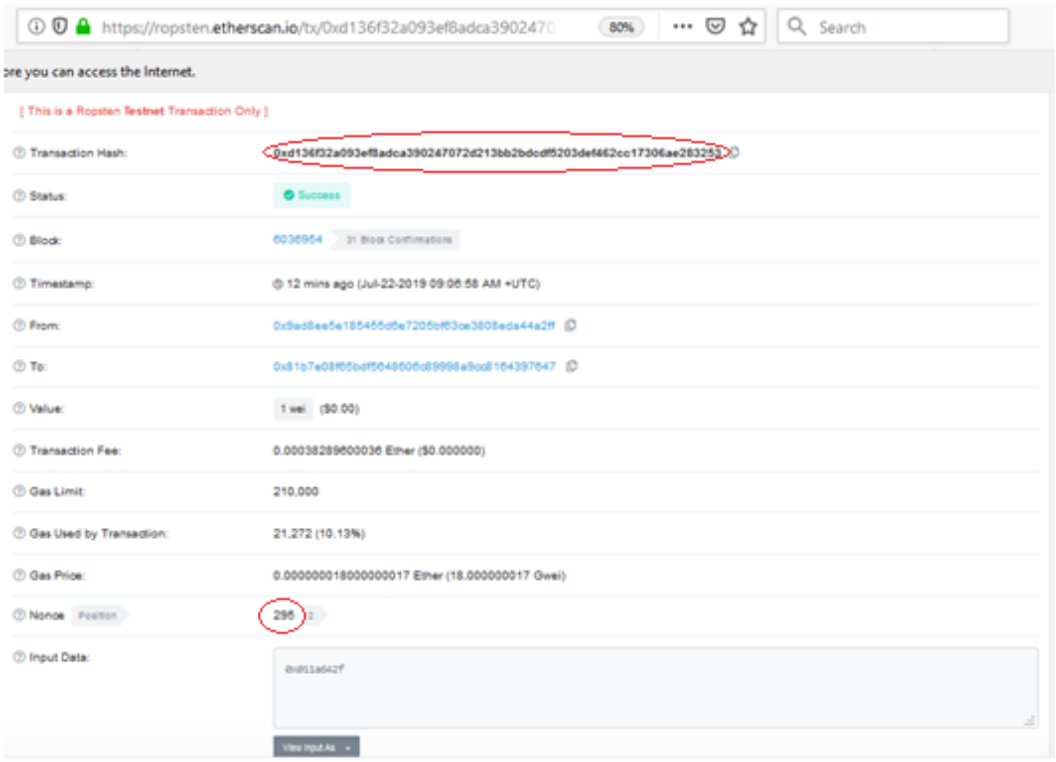
```
Print ("Monitor your transaction at: https://ropsten.etherscan.io/tx/%s" %eth_hash)
```

Gambar 4.3. Proses *Monitoring* dengan *Link* Transaksi

Hasil pengujian proses dapat dilihat pada Gambar 4.4. Ini adalah hasil proses untuk transaksi yang ke-295. Data Gambar 4.4 diperoleh dari *console* Zerynth Studio. Validitas *link* diuji langsung dengan meng-*copy* ke *browser* dan memeriksa langsung ke **etherscan.io**. Hasil pemeriksaan dapat dilihat pada Gambar 4.5. Di Gambar 4.5, dapat dipastikan bahwa *Transaction Hash* adalah sama dengan *link* yang diberikan pada Gambar 4.4. Nomor transaksi juga dapat dipastikan pada kolom *Nonce* (295). Ini membuktikan bahwa proses *monitoring* berhasil dilakukan.

```
Monitor your transaction at: https://ropsten.etherscan.io/tx/0xd136f32a093ef8adca390247072d213bb2bdcdf5203def462cc17306ae283253
```

Gambar 4.4. *Link* Hasil Pengujian Proses *Monitoring*



Gambar 4.5. Validitas *Link* Proses *Monitoring*

4.1.3. Pengujian dan Analisis Modul *Verifying*

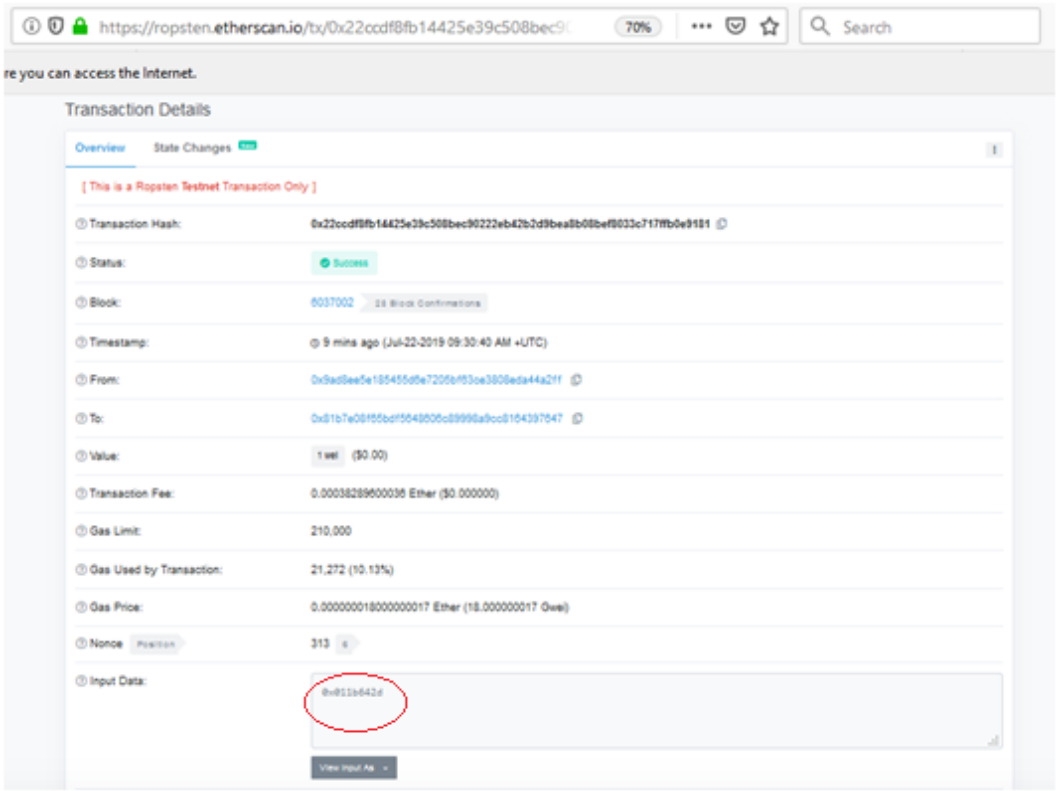
Melalui proses *Verifying*, data yang dikirim oleh sistem rancangan dipastikan sama dengan data yang tercatat dalam jaringan *blockchain*, yaitu Ropsten. Pengujian ini dilakukan dengan menambahkan baris instruksi seperti yang terlihat dari Gambar 4.6. Dengan instruksi tersebut, dapat dipastikan awal pengiriman data dilakukan, data yang dikirim, dan akhir dari pengiriman data tersebut.

```
Print ("Begin of Sending Value..")

Print ("Sending Values… [", hex(data_send), "]")
Print ("Sending Completed..")
```

Gambar 4.6. Proses Pengujian Modul *Verifying*

```
Begin of Sending Value..
Sending values... [ 0x011B642D ]
Sending Completed..
```

Gambar 4.7. Hasil Pengujian Modul *Verifying*

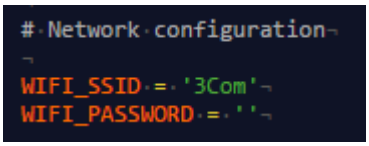


Gambar 4.8. Pemeriksaan Data Hasil *Verfying*

Hasil pengujian modul dapat dilihat pada Gambar 4.7. Pengujian dilakukan pada transaksi ke-313. Data hasil pengujian diperiksa dengan *link* transaksi pada **etherscan.io**. Hasil pemeriksaan ditampilkan pada Gambar 43. Pada Gambar 4.8, dapat dipastikan bahwa transaksi yang diperiksa adalah transaksi ke-313 (nilai *Nonce* adalah 313. Nilai data adalah 0x011b642d, juga dipastikan sama dengan data Gambar 4.7.

4.1.4. Pengujian dan AnalisisModul *Connecting*

Pengujian ini untuk memeriksa apakah modul berhasil melakukan koneksi ke *router* WiFi yang digunakan untuk terhubung ke Internet. Dalam pengujian ini, *router* yang digunakan adalah merk 3Com, dengan nama koneksi 3Com dan berlokasi di Sekretariat Program Studi Teknik Elektro Universitas Tarumanagara. Konfigurasi yang digunakan dapat dilihat pada Gambar 4.9.



Gambar 4.9.Konfigurasi Jaringan

Pengujian dilakukan dengan memonitor status koneksi ke WiFi dengan baris perintah seperti diperlihatkan pada Gambar 4.10. Pengujian dimulai dengan memeriksa apakah *driver* telah terinisiasi dengan baik. Apabila sudah, maka koneksi ke *router* WiFi dilakukan. Hasil pengujian tanpa kesalahan diperlihatkan pada Gambar 4.11. Apabila terjadi kesalahan, maka akan muncul pesan kesalahan seperti yang diperlihatkan pada Gambar 4.12. Kesalahan terutama terjadi karena prosedur *reset* setelah setiap proses dilakukan.

```
#Connect to WiFi Network

Print (“Begin to Initiating Network Driver…”)

Net_driver.auto_init()

Print (“Driver Initialized…”)

Print (“Conneting to WiFi…”)

Wifi.link (config.WIFI_SSID, wifi.WIFI_WPA2, config.WIFI_PASSWORD)
Print (“WiFi Connected to “, config.WIFI_SSID)
```

Gambar 4.10. Proses Pengujian Modul *Connecting*

```
Begin to Initiating Network Driver...
Driver initialized...
Connecting to WiFi...
WiFi Connected to  3Com
```

Gambar 4.11. Hasil Pengujian Tanpa Kesalahan Terjadi

```
E (75380) wifi: esp_wifi_connect 952 wifi not start
```

Gambar 4.12. Hasil Pengujian Dengan Kesalahan

4.1.5. Pengujian dan Analisis Modul *Initiating*

Modul ini melakukan inisiasi ke RPC yang disediakan **infura.io**. Tanpa RPC, transaksi tidak dapat diteruskan ke EVM di jaringan Ropsten. URL RPC dan validasi sertifikat SSL yang digunakan diperlihatkan pada Gambar 4.13. Proses pengujian diperlihatkan pada Gambar 4.14. Proses pengujian dimulai dengan instruksi untuk mengetahui bahwa prosedur inisiasi akan dilakukan. Bila berhasil, maka akan ditampilkan pesan bahwa inisiasi telah berhasil dilakukan dan proses dapat berlanjut ke prosedur berikutnya. Hasil pengujian ditunjukkan pada Gambar 4.15. Hasil pengujian tidak pernah memperlihatkan kegagalan yang terjadi dalam proses tersebut.

```
RPC_URL = 'https://ropsten.infura.io/v3/80859bb1f4c9416f96dffb0e97747e4a'
SSL_CTX = ssl.create_ssl_context(
    cacert=config.CA_CERT,
    options=ssl.CERT_REQUIRED|ssl.SERVER_AUTH
)
```

Gambar 4.13. URL RPC dan Validasi SSL

Print (“Begin of Initiating of the RPC Node..”)

Eth = rpc.RPC(config.RPC_URL, ssl_ctx=SSL_CTX)
Print (“Initiating RPC Node Success..”)

Gambar 4.14. Proses Pengujian Modul *Initiating*

```
Begin of Initiating of the RPC Node..
Initiating RPC Node Success..
```

Gambar 4.15. Hasil Pengujian Modul *Initiating*

4.1.6. Pengujian dan Analisis Modul *Getting Information*

Modul *Getting Information*berguna untuk mendapatkan informasi perihal jumlah ETH yang tersedia di akun, juga informasi tentang biaya per transaksi yang dinyatakan dalam satuan GAS, serta kode unik transaksi (*nonce*). Pengujian dilakukan dengan cara seperti diperlihatkan pada Gambar 4.16. Informasi jumlah ETH (*balance*) didapatkan dengan melakukan koneksi ke alamat *wallet* yang telah di-*set*. Nilai ini didapatkan dengan me-*retrieve* dari jaringan Ropsten. Nilai GAS didapatkan dari program pada *smart contract*. Di sini telah diatur nilai maksimum GAS yang ingin dibayarkan untuk setiap proses. Nilai ini akan menentukan apakah proses terjadi dengan cepat atau lambat. Untuk nilai *Transaction Count* atau *Nonce* didapatkan dari urutan blok tempat transaksi dicatat.

Apabila modul ini berjalan dengan baik, maka akan didapatkan hasil dengan tampilan yang menyatakan bahwa *balance* berhasil diambil dan juga ditampilkan jumlahnya (dalam format kode heksadesimal). Demikian juga dengan *nonce* (hasil pengujian ini adalah 339) dan juga ditampilkan nilai GAS yang dibutuhkan dalam menyelesaikan transaksi.Hasil pengujian diperlihatkan pada Gambar 4.17.

```
# Get our current balance
    print ("Begin of Getting Information..")
    print ("Getting of the Ethereum Balance in Wallet..")
    balance = eth.getBalance(config.ADDRESS)
print("Balance:", balance, " in Wallet Address ", config.ADDRESS)
    if not balance:
        print(eth.last_error)
        raise Exception
    print ("Success in Retrieving Balance..")
    print ("Getting of Gas Price to Process a Transaction..")
print("Gas Price:", eth.getGasPrice())
    print ("Gas Price Acquired..")
    print ("Getting of the Transaction Count or Nonce..")
```

```
    nt = eth.getTransactionCount(config.ADDRESS)
print("TCount:", nt)
print("Chain:", eth.getChainId())
    print ("Success in Retrieving Transaction Count..")


    print ("Getting Information is Done Successfully..")
```

Gambar 4.16. Proses Pengujian Modul *Getting Information*

```
Begin of Getting Information..
Getting of the Ethereum Ballance in Wallet..
Balance: 0x2788e4ed3a9c4ce7  in Wallet Address  0x9Ad8Ee5E185455D6E7205bF63cE3808EDa44A2Ff
Success in Retrieving Ballance..
Getting of Gas Price to Process a Transaction..
Gas Price: 3000000000
Gas Price Acquired..
Getting of the Transaction Count or Nonce..
TCount: 397
Chain: 3
Success in Retrieving Transaction Count..
```

Gambar 4.17. Hasil Pengujian Modul *Getting Information*

```
print ("Begin of Prepairing Transaction Object..")
ethtrans = ethereum.Transaction()
print ("Setting wei..")
ethtrans.set_value(1, ethereum.WEI)
print ("WEI Price: ", ethereum.WEI)
print ("WEI Value Set..")
print ("Setting GAS Price..")
ethtrans.set_gas_price("0x430e23411")
ethtrans.set_gas_limit("0x33450")
print ("GAS Set..")
print ("Setting Smart COntract Address..")
ethtrans.set_receiver(config.RECEIVER_ADDRESS)
ethtrans.set_chain(ethereum.ROPSTEN)
print ("Wallet Address for Transaction: ", config.RECEIVER_ADDRESS)
print ("Address Set..")
print ("Setting Up Data for Transaction..")
alt_send = int(alt)
pres_send = (pres & 0x000000FF) << 8
temp_send = int(temp)  << 16
ir_send = digitalRead(D4) << 24
data_send = (ir_send | temp_send | pres_send | alt_send)
print ("Data to be sent: ", hex(data_send))
```

```
ethtrans.set_data(data_send)
print ("Data Set Up Done..")
print ("Setting Nonce..")
ethtrans.set_nonce(nt)
print ("Nonce ", nt)
print ("Nonce Set..")


print ("Object Prepairing Done..")
```

Gambar 4.18. Proses Pengujian Modul *Prepairing*

4.1.7. Pengujian dan Analisis Modul *Prepairing*

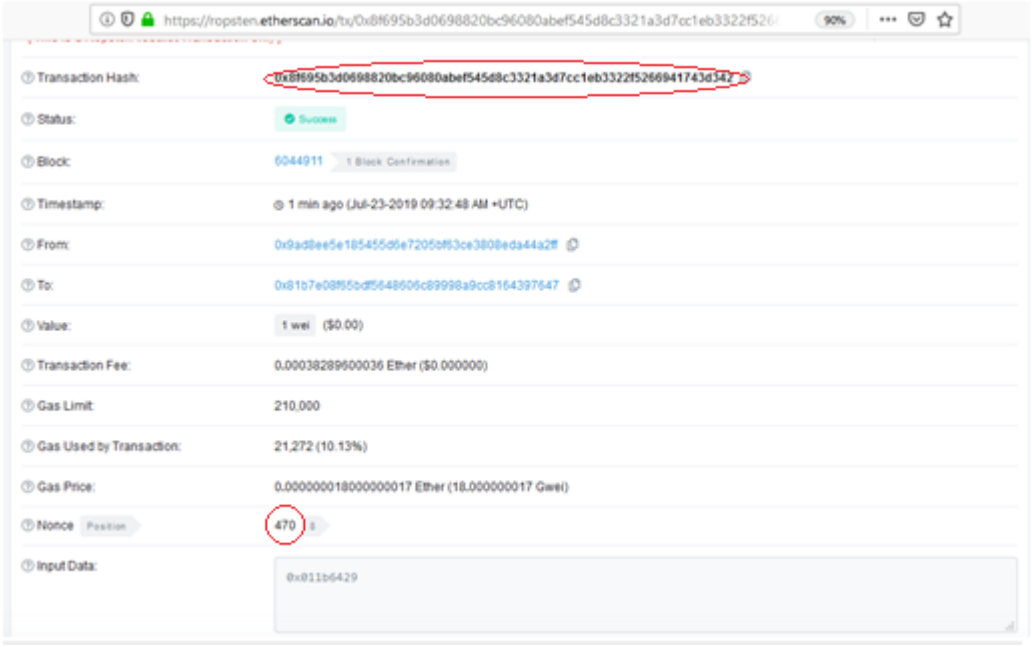
Modul *Prepairing* merupakan modul untuk persiapan sebelum pengiriman transaksi dilakukan. Modul ini akan mengatur nilai WEI, batasan nilai GAS, alamat *walletsmart contract*, nama jaringan yang digunakan dan format data. Data diformat dengan pola **ir_send | temp_send | pres_send | alt_send**. Format data ini memungkinkan pengiriman data yang kompak, karena tiap proses data ke node EVM akan dikenai biaya GAS. Ini menjadikan format data yang kompak menjadi penting.

Gambar 4.18 memperlihatkan proses pengujian terhadap modul ini, dengan memastikan tiap nilai telah dipersiapkan dengan baik. Gambar 4.19 memperlihatkan hasil pengujian terhadap modul *Prepairing* pada layar konsol Zerynth Studio. Gambar 4.20 memperlihatkan bahwa hasil pengujian pada Gambar 4.19 adalah benar melalui pemeriksaan *Transaction Hash* di **etherscan.io**.

Gambar 4.19 dan Gambar 4.20, dapat dilihat bahwa alamat *wallet* adalah benar, yaitu 0x81b7e08f65bdf5648606c89998a9cc8164397647. Nilai *nonce* adalah 470. Data yang dikirim setelah diformat juga sama, yaitu 0x011B6429. Ini membuktikan bahwa pengujian modul *Prepairing* berhasil.

```
Begin of Prepairing Transaction Object..
Setting wei..
WEI Price:  0
WEI Value Set..
Setting GAS Price..
GAS Set..
Setting Smart COntract Address..
Wallet Address for Transaction: 0x81b7e08f65bdf5648606c89998a9cc8164397647
Address Set..
Setting Up Data for Transaction..
Data to be sent: 0x 0x011B6429
Data Set Up Done..
Setting Nonce..
Nonce 470
Nonce Set..
Object Prepairing Done..
```

Gambar 4.19. Hasil Pengujian Modul *Prepairing*



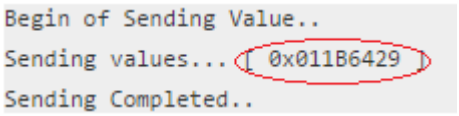
Gambar 4.20. Pemeriksaan Hasil Modul *Prepairing* di **Ehterscan.io**

4.1.8. Pengujian dan Analisis Modul *Sending Transaction*

Modul terakhir dari sistem ini adalah modul *Sending Transaction*. Modul mengirim data yang telah diformat untuk dicatatkan ke *smart contract*. Pengujian modul ini dilakukan dengan layar konsol Zerynth Studio untuk memastikan data yang dikirim, dan kemudian memeriksa hasilnya di **etherscan.io**. Pengujian ini menggunakan data yang sama dengan pengujian pada modul *Prepairing*, dengan *nonce* 470 dan nilai data 0x011B6429. Gambar 4.21 memperlihatkan proses pengujian modul *Sending Transaction*. Hasil pada layar konsol dapat dilihat pada Gambar 4.22 dan pengecekan pada **etherscan.io** dapat dilihat pada Gambar 4.20 di atas. Dengan samanya data yang terlihat pada Gambar 4.22 dan Gambar 4.20, dapat disimpulkan bahwa pengujian modul berhasil.

```
print ("Begin of Sending Value..")
print("Sending values... [", hex(data_send), "]")
eth_hash = eth.sendTransaction(ethtrans.to_rlp(True))
print ("Sending Completed..")
print("Monitor your transaction at: https://ropsten.etherscan.io/tx/%s" %
eth_hash)

print("Sent!")
```

Gambar 4.21. Pengujian Modul *Sending Transaction*



Gambar 4.22. Hasil Pengujian Modul *Sending Transaction* di Konsol

4.2. Pembahasan Modular Perangkat Keras

4.2.1. Pengujian dan Analisis Modul Sensor MPL3115A2

Pengujian modul perangkat keras sensor ini dilakukan sekaligus karena seluruh perangkat berada pada satu modul yang sama yaitu modul MPL3115A2. Pengujian secara fisik dipastikan melalui perangkat lunak. Gambar 4.23

memperlihatkan proses pengujian modul MPL3115A2. Hal yang dilakukan dalam pengujian ini adalah menguji apakah koneksi MPL3115A2 dapat dilakukan dengan menggunakan I2C. Apabila berhasil dilakukan, maka dapat dipastikan modul bekerja dengan baik.Gambar 4.24 memperlihatkan hasil pengujian modul tersebut. Gambar 4.24 membuktikan bahwa modul sensor MPL3115A2 bekerja dengan baik.

```
print ("Begin of Sensor Initialization for modul MPL3115A2..")
mpl = mpl3115a2.MPL3115A2(I2C0)
mpl.start()
mpl.init()


print("Sensor Initialized and Ready to Use...")
```

Gambar 4.23. Pengujian Modul Perangkat Keras MPL3115A2

```
Begin of Sensor Initialization for modul MPL3115A2..
Sensor Initialized and Ready to Use...
```

Gambar 4.24.Hasil Pengujian MPL3115A dari Konsol

4.2.2. Pengujian dan Analisis Modul WiFi

Pengujian modul perangkat keras WiFi menggunakan metode yang sama dengan pengujian modul perangkat lunak modul *Connecting* (Subbab 4.1.4). Modul WiFi ini merupakan modul *built-in* pada modul Lolin D32. Jadi pengujian pada Subbab 4.1.4 dapat dijadikan acuan keberhasil pengujian modul perangkat kerasnya. Subbab 4.1.4 menunjukkan bahwa perangkat keras dapat terhubung dengan baik ke *router* dengan SSID 3Com.

4.3. Pembahasan Sistem Terintegrasi

Pengujian sistem terintegrasi dilakukan dengan menguji sistem secara keseluruhan dalam satu kali proses. Gambar 4.25 memperlihatkan hasil pengujian

terintegrasi pada layar konsol Zerynth Studio. Kode *Transaction Hash* adalah 0x85c012cc0c43a49331ea523a3d29deffe9c41de5d4ea604f28c5498fbfb4f47d.

Begin of Sensor Initialization for modul MPL3115A2..

Sensor Initialized and Ready to Use...

Begin to Initiating Network Driver...

Driver initialized...

Connecting to WiFi...

WiFi Connected to 3Com

Asking ethereum...

Begin of Initiating of the RPC Node..

Initiating RPC Node Success..

Begin of Getting Information..

Getting of the Ethereum Ballance in Wallet..

Balance: 0x26e704247c3389c8 in Wallet Address 0x9Ad8Ee5E185455D6E7205bF63cE3808EDa44A2Ff

Success in Retrieving Ballance..

Getting off Gas Price to Process a Transaction..

Gas Price: 1000000000

Gas Price Acquired..

Getting of the Transaction Count or Nonce..

TCount: 516

Chain: 3

Success in Retrieving Transaction Count..

Getting Information is Done Successfully..

Altitude: 38.5625 meter [0x26]

Pressure: 100 kPa [0x64]

Temperature: 28.125 Celcius [0x1C]

Begin of Prepairing Transaction Object..

Setting wei..

WEI Price: 0

WEI Value Set..

Setting GAS Price..

GAS Set..

Setting Smart COntract Address..

Wallet Address for Transaction: 0x81b7e08f65bdf5648606c89998a9cc8164397647

Address Set..

Setting Up Data for Transaction..

Data to be sent: 0x011C6426

Data Set Up Done..

Setting Nonce..

Nonce 516

Nonce Set..

Object Prepairing Done..

Signing Completed...

Begin of Sending Value..

Sending values... [0x011C6426]

Sending Completed..

Monitor your transaction at:
https://ropsten.etherscan.io/tx/0x85c012cc0c43a49331ea523a3d29deffe9c41de5d4ea604f28c5498fbfb4f47d

Sent!

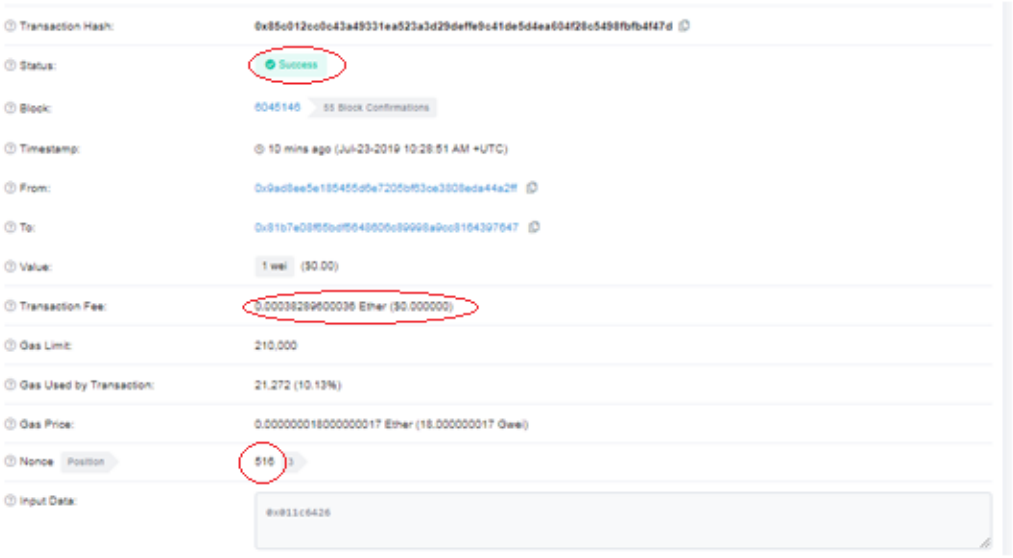
Gambar 4.25. Hasil Pengujian Terintegrasi Pada Konsol

Dengan data yang dikirim adalah:

- Posisi : 1 (terhubung),
- Ketinggian : 38,5625 m (0x26),
- Tekanan : 100 kPa (0x64),
- Suhu : 28,125°C (0x1C).

Data setelah diformat adalah 0x011C6426. Dikirim ke alamat *wallet*0x81b7e08f65bdf5648606c89998a9cc8164397647. Hasil pengujian ini divalidasi di **etherscan.io**, dengan hasil pada Gambar 61. Hasil validasi pada Gambar 4.26 memperlihatkan bahwa transaksi telah sukses. Ini berarti data berhasil dikirim ke jaringan *blockchain*. Di gambar ini juga terlihat biaya untuk transaksi ini yaitu sebesar 0.000000018000000017 ETH. Bila dikonversi ke rupiah (dengan nilai konversi sebesar Rp. 2.949.349 per ETH, acuan harga dari Tukarku.com pada tanggal 23 Juli 2019 jam 5:44 WIB) akan senilai Rp. 0,053. Pada Gambar 4.26 juga dapat dilihat bahwa *Transaction Count* atau *Nonce* adalah sebesar 516. Ini artinya pengujian telah dilakukan sebanyak 516 kali dan tercatat di jaringan *blockchain*.

Perekaman dilakukan tiap 30 detik, maka dalam satu hari akan dilakukan perekaman sebanyak 2.880 kali (= 2 x 60 x 24). Ini akan memerlukan biaya sebesar Rp. 152,64.



Gambar 4.26. Hasil Validasi Pengujian Terintegrasi di etherscan.io

BAB 5
KESIMPULAN DAN SARAN

5.1. Kesimpulan

1. Perancangan sistem ini berhasil membuktikan bahwa data IoT dapat disekuritisasi dengan teknologi *blockchain*. Pengujian telah dilakukan sebanyak 516 kali (berdasarkan data *nonce* terakhir yang tercatat pada **etherscan.io**). Data berhasil direkam ke *state variable* pada jaringan *blockchain*.
2. Walau pun perancangan sistem ini memperlihatkan hasil yang memuaskan, demikian juga dengan pengujian terhadap sistem, tetapi sistem ini masih memiliki kekurangan, yaitu perlu biaya tambahan (selain biaya konektivitas Internet). Biaya tambahan ini adalah berupa biaya proses oleh tiap *node* pada jaringan *Blockchain*. Pada pengujian yang telah dilakukan pada jaringan Ropsten (yang merupakan jaringan TestNet) menunjukkan bahwa diperlukan Rp. 0,053 untuk melakukan perekaman tiap transaksi. Jadi, per hari memerlukan Rp.152,64.
3. Nilai sebesar Rp. 152,64 ini tentu saja tidak begitu signifikan, tetapi perlu diingat bahwa pengujian dilakukan di jaringan TestNet yang dapat dianggap kondisi ideal. Juga, perekaman dianggap valid dengan hanya perlu satu kali proses validasi. Apa bila ini dilakukan di jaringan MainNet, mungkin akan memberikan hasil yang berbeda. Oleh karena, nilai Gas akan ditentukan oleh nilai tukar ETH dan juga kepadatan lalu lintas tranksaksi serta validasi oleh lebih dari satu *node*.

5.2. Saran

Untuk saran terhadap sistem ini, dalam upaya penerapan sistem ini, dapat dipikirkan untuk membangun jaringan *Blockchain* sendiri. Dengan demikian, biaya transaksi dapat ditiadakan. Keamanan data tetap terjamin.

DAFTAR PUSTAKA

[1] S. Smith, *Internet of Things connected devices to triple by 2021, reaching over 46 billion units*, in Juniper Research, 2016.

[2] K. Rawlinson. *Hp study reveals 70 percent of internet of things devices vulnerable to attack. [Online]*. Available: http://www8.hp.com/us/en/ hp-news/press-release.html?id=1744676#.WUrrwWgrKM8, diakses pada tanggal 29 Agustus 2018 pada jam 21.00 WIB.

[3] K. Christidis and M. Devetsikiotis, *Blockchains and Smart Contracts for the Internet of Things*, IEEE Access, vol. 4, 2016, pp. 2292 – 2303.

[4] P. Danzi, et. al., *Analysis of the Communication Traffic for Blockchain Synchronization of IoT Devices*, IEEE International Conference on Communications (ICC), 2018.

[5] J. Pan and E. Alqrem, *EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts*, https://arxiv.org/abs/1806.06185, 2018, diakses pada tanggal 29 Agustus 2018 pada jam 22.00 WIB.

[6] K. K. Patel and S. M. Patel, *Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges*, IJSEC, vol. 6, no. 5, ISSN: 2321 3361, 2016, pp.6122-6131.

[7] L. Atzori, A. Iera, and G. Morabito, *Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm*, Ad Hoc Networks, vol. 56, pp. 122-140, 2017.

[8] L. Srivastava and T. Kelly, *The internet of things*, International Telecommunication Union, Tech. Rep, vol. 7, 2005.

[9] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, *IoT Middleware: A Survey on Issues and Enabling Technologies*, IEEE Internet of Things Journal, vol. 4, pp. 1-20, 2017.

[10] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*,https://bitcoin.org/bitcoin.pdf, 2008, diakses pada tanggal 20 September 2018 pada jam 22.00 WIB.

[11] A. Narayanan et al., *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, 2016.

[12] R. C. Merkle, *A digital signature based on a conventional encryption function*, in Conference on the Theory and Application of Cryptographic Techniques. Springer, 1987, pp. 369–378.

[13] U. Guin, P. Cui and A. Skjellum, *Ensuring Proof-of-Authenticity of IoT Edge Devices using Blockchain Technology*, The 2018 IEEE International Conference on Blockchain, 2018.

[14] A. Dorri, et. al., *Blockchain for IoT Security and Privacy: The Case Study of a Smart Home*, IEEE Percom Workshop on Security Privacy and Trust in The Internet of Thing, 2017.

[15] S. F. T. O. Mendonca, J. F. S. Junior and F. M. R. Alencar, *The Blockchain-based Internet of Things Development: Initiatives and Challenges*, ICSEA, 2017, pp. 28-33.

[16] J. Kogure, et. al., *The Blockchain-based Internet of Things Development: Initiatives and Challenges*, Fujitsu Scientific & Technical Journal, vol. 53, no. 5, pp 56-61, 2017.

[17] A. Dorri, et. al., *LSB: A Lightweight Scalable BlockChain for IoT Security and Privacy*, https://arxiv.org/abs/1712.02969, 2017, diakses pada tanggal 20 September 2018 pada jam 23.00 WIB.

LAMPIRAN

Config.py

```
"""
    Configuration files. Please double check all the informations for the
    project to run correctly.

    The included CA certificate is for `Amazon Root CA 1`, and it's being used
    for *.infura.io RPC node. Omit it if your node is not using https, or
    change it with the one of your CA.
"""

# Network configuration

WIFI_SSID = '3Com'
WIFI_PASSWORD = ''


# Ethereum configuration

RPC_URL = 'https://ropsten.infura.io/v3/80859bb1f4c9416f96dffb0e97747e4a'

ADDRESS = '0x9Ad8Ee5E185455D6E7205bF63cE3808EDa44A2Ff'
PRIVATE_KEY =
'0x33627590e0b8751ddabec6d0278c1e5f7da1821c9eb2a7b06f5d0edb3ec579ae'
# Note the '0x' prefix - it must be used

# Do not edit below if not sure
RECEIVER_ADDRESS = '0x81b7e08f65bdf5648606c89998a9cc8164397647' #
MetaMask faucet

WEI_AMOUNT = 5 # The amount to be sent, in Wei

CA_CERT = """-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0B
AQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwF
wYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNz
AwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxM
QQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBA
LJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHF
zZM
```

9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/q
w
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwh
mahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8k
Di1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+O
yQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVH
Q8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSI
b3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rb
xenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wU
s
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJb
YK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6I
Q6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXe
GADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
\x00"""

Main.py

```
import streams
import mcu

# Ethereum modules
from blockchain.ethereum import ethereum
from blockchain.ethereum import rpc

# Sensor modules
from nxp.mpl3115a2 import mpl3115a2

# WiFi drivers
from espressif.esp32net import esp32wifi as net_driver # for ESP-32
# from broadcom.bcm43362 import bcm43362 as net_driver # for Particle Photon
from wireless import wifi

# SSL module for https
import ssl
```

```
# Configuration file
import config

# The SSL context is needed to validate https certificates
SSL_CTX = ssl.create_ssl_context(
   cacert=config.CA_CERT,
   options=ssl.CERT_REQUIRED|ssl.SERVER_AUTH
)

pinMode(D15, OUTPUT)
pinMode(D5, OUTPUT)
pinMode(D4, INPUT)
digitalWrite(D15, HIGH)

try:
   streams.serial()

   # Setup sensor
   print ("Begin of Sensor Initialization for modul MPL3115A2..")
   mpl = mpl3115a2.MPL3115A2(I2C0)
   mpl.start()
   mpl.init()
   print("Sensor Initialized and Ready to Use...")

   # Connect to WiFi network
   print ("Begin to Initiating Network Driver...")
   net_driver.auto_init()
   print ("Driver initialized...")
   print("Connecting to WiFi...")
   wifi.link(config.WIFI_SSID, wifi.WIFI_WPA2, config.WIFI_PASSWORD)
   print("WiFi Connected to ",config. WIFI_SSID)

   digitalWrite(D5, LOW)
   print("Asking ethereum...")

   # Init the RPC node
   print ("Begin of Initiating of the RPC Node..")
   eth = rpc.RPC(config.RPC_URL, ssl_ctx=SSL_CTX)
   print ("Initiating RPC Node Success..")

   # Get our current balance
   print ("Begin of Getting Information..")
   print ("Getting of the Ethereum Ballance in Wallet..")
   balance = eth.getBalance(config.ADDRESS)
   print("Balance:", balance, " in Wallet Address ", config.ADDRESS)
   if not balance:
      print(eth.last_error)
      raise Exception
```

```
print ("Success in Retrieving Ballance..")


print ("Getting off Gas Price to Process a Transaction..")
print("Gas Price:", eth.getGasPrice())
print ("Gas Price Acquired..")
print ("Getting of the Transaction Count or Nonce..")
nt = eth.getTransactionCount(config.ADDRESS)
print("TCount:", nt)
print("Chain:", eth.getChainId())
print ("Success in Retrieving Transaction Count..")
print ("Getting Information is Done Successfully..")


while True:
   digitalWrite(D5, LOW)
   # Read sensor data
   alt = mpl.get_alt() # Read altitude
   print("Altitude: ", alt, "meter", " [", hex(int(alt)), "]")
   pres = int(mpl.get_pres()/1000) # Read pressure
   print("Pressure: ", pres, "kPa", " [", hex(int(pres)), "]")
   temp = mpl.get_temp() # Read temperature
   print("Temperature: ", temp, "Celcius", " [", hex(int(temp)), "]")


   # Prepare a transaction object
   print ("Begin of Prepairing Transaction Object..")
   ethtrans = ethereum.Transaction()
   print ("Setting wei..")
   ethtrans.set_value(1, ethereum.WEI)
   print ("WEI Price: ", ethereum.WEI)
   print ("WEI Value Set..")
   print ("Setting GAS Price..")
   ethtrans.set_gas_price("0x430e23411")
   ethtrans.set_gas_limit("0x33450")
   print ("GAS Set..")
   print ("Setting Smart COntract Address..")
   ethtrans.set_receiver(config.RECEIVER_ADDRESS)
   ethtrans.set_chain(ethereum.ROPSTEN)
   print ("Wallet Address for Transaction: ", config.RECEIVER_ADDRESS)
   print ("Address Set..")
   print ("Setting Up Data for Transaction..")
   alt_send = int(alt)
   pres_send = (pres & 0x000000FF) << 8
   temp_send = int(temp)  << 16
   ir_send = digitalRead(D4) << 24
   data_send = (ir_send | temp_send | pres_send | alt_send)
   print ("Data to be sent: 0x", hex(data_send))
```

```
        ethtrans.set_data(data_send)
        print ("Data Set Up Done..")
        print ("Setting Nonce..")
        ethtrans.set_nonce(nt)
        print ("Nonce ", nt)
        print ("Nonce Set..")
        print ("Object Prepairing Done..")

        ethtrans.sign(config.PRIVATE_KEY)
        print("Signing Completed...")

        # Send the transaction
        print ("Begin of Sending Value..")
        print("Sending values... [", hex(data_send), "]")
        eth_hash = eth.sendTransaction(ethtrans.to_rlp(True))
        print ("Sending Completed..")
            print("Monitor your transaction at: https://ropsten.etherscan.io/tx/%s" %
eth_hash)
        print("Sent!")
        digitalWrite(D5, HIGH)
        sleep(30000)
        mcu.reset()
except Exception as e:
    print(e)
    digitalWrite(D5, HIGH)
    sleep(3000)
    mcu.reset()
```

DAFTAR ISI

DAFTAR GAMBAR