IMPLEMENTASI ALGORITME SHA-256 MENGGUNAKAN PROTOKOL MQTT PADA BUDIDAYA IKAN HIAS

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Yogi Anugrah
135150301111079

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

IMPLEMENTASI ALGORITME SHA-256 MENGGUNAKAN PROTOKOL MQTT PADA BUDIDAYA IKAN HIAS

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Nama: Yogi Anugrah
NIM: 135150301111079

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Januari 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Mochammad Hannats Hanafi Ichsan, S.ST, M.T.	Ari Kusyanti, S.T, M.Sc.
NIK: 201405 881229 1 001	NIP: 19831283 201803 200 2

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Januari 2019

Yogi Anugrah

NIM: 135150301111079

KATA PENGANTAR

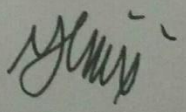
Puji syukur ke hadirat Tuhan Yang Maha Esa atas berkat dan penyertaan-Nya sehingga penulis dapat menyelesaikan penulisan skripsi berjudul "Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias" dengan baik. Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.

Penulis menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak yang telah memberikan bantuan baik lahir maupun batin selama penulisan skripsi ini. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, Bapak Dahnial Syauqy, S.T., M.T., M.Sc dan Bapak M. Tanzil Furqon, S.Kom, M.ComSc selaku Ketua Jurusan Teknik Informatika, Ketua Program Studi Teknik Informatika Keminatan Teknik Komputer dan Sekretaris jurusan Teknik Informatika
2. Mochammad Hannats Hanafi Ichsan, S.ST, M.T dan Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing I dan pembimbing II yang telah dengan sabar membimbing, mengarahkan, serta meluangkan waktu untuk penulis sehingga dapat menyelesaikan skripsi ini.
3. Keluarga dan teman yang telah memberikan motivasi ke pada penulis di tengah kesibukan mengerjakan skripsi.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun senantiasa penulis harapkan. Semoga skripsi ini dapat memberikan manfaat bagi semua pihak membacanya.

Malang, 2 Januari 2019



Penulis

yogianugrah@gmail.com

ABSTRAK

Yogi Anugrah, Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias

Dosen Pembimbing: Mochammad Hannats Hanafi Ichsan, S.ST, M.T dan Ari Kusyanti, S.T, M.Sc

Penggunaan *Internet of things* (IoT) yang dapat mengkomunikasikan mesin dan mesin memunculkan masalah dalam pengirim data yang dikirim. Salah satu contoh arsitektur yang menguhubung mesin dan mesin adalah *publish-subscribe.* Saat *publisher* mengirimkan data hasil *monitoring* sensor ke *subscriber,* data rawan diubah sebelum sampai ke *subscriber.* Oleh karena itu ditambahkanlah pengecekan integritas sehingga data hasil *monitoring* sensor yang dikirim oleh *publisher* agar data tidak dapat dimodifikasi oleh pihak yang tidak berwenang. Pengecekan integritas data dilakukan melalui metode kriptografi berupa *Message Authentication Code* (MAC) , algoritme MAC yang sering digunakan adalah SHA-1 dan MD5 tetapi dalam penggunaanya sering mengalami *brute force,* maka digunakanlah algoritme SHA-2 sebagai pembaruan dari algoritme SHA-1. Berdasarkan pengujian integritas data menggunakan algoritme SHA-256 rata– rata waktu dibutuhkan untuk menghasilkan MAC pada *publisher* 604,958 ms dan 717,344 pada *subscriber.* Peningkatan penggunaan *memory* yang digunakan tanpa dan saat menggunakan MAC algoritme SHA-256 sebesar 0,0078 MB sedangkan pada *subscriber* sebesar 0,0015 MB.

Kata kunci : *internet of things,* MQTT, *publish-subscribe,* SHA-2

ABSTRACT

Yogi Anugrah, The implementation of the SHA-256 algorithm uses the MQTT protocol in ornamental fish farming

Supervisors: Mochammad Hannats Hanafi Ichsan, S.ST, M.T and Ari Kusyanti, S.T, M.Sc

The use of Internet of things (IoT) that can communicate machines and machines raises problems in sending data sent. One example of architecture that connects machines and machines is publish-subscribe. When the publisher sends sensor monitoring data to the subscriber, the vulnerable data is changed before reaching the subscriber. Therefore, integrity checks were added so that the sensor monitoring data sent by the publisher could not be modified by unauthorized parties. Data integrity checking is done through cryptographic methods in the form of Message Authentication Code (MAC), MAC algorithms that are often used are SHA-1 and MD5 but in the use of frequent brute force, the SHA-2 algorithm is used as an update of the SHA-1 algorithm. Based on data integrity testing using the SHA-256 algorithm, the average time needed to produce the MAC on the publisher is 604,958 ms and 717,344 on the subscriber. The increase in memory usage used without and when using the MAC algorithm SHA-256 is 0.0078 MB while in the subscriber is 0.0015 MB.

Keywords: *internet of things,* MQTT*, publish-subscribe,* SHA-2

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Perkembangan teknologi saat ini memungkinkan suatu objek yang memiliki kemampuan untuk mentransfer data melalui jaringan tanpa memerlukan interaksi manusia ke manusia atau manusia ke komputer yang disebut sebagai *Internet of things*. *Internet of things* dapat mengkomunikasikan antara mesin dan mesin tanpa interaksi dari manusia (Datta, Bonnet, & Nikaein, 2014). Interaksi antar mesin kebanyakan menggunakan mikrokontroler. Namun Mikrokontroler memilki kekurangan yaitu sulit untuk mengontrol peralatan dan pengawasan untuk beberapa fungsi secara bersamaan dan komputer memiliki kendala harga yang sangat mahal dan konsumsi daya yang besar, untuk mengatasi hal tersebut digunakanlah Raspberry Pi (Patchava, Kandala, & Babu ,2015).

Protokol *Message Queue Telemetry Transport* (MQTT) adalah protokol berjenis arsitektur *publish-subscribe* yang sering digunakan karena memiliki keunggulan seperti sumber daya dan ukuran data yang ditranmisikan kecil (Kusumawardhana, Ichsan & Primananda, 2018). Protokol MQTT memiliki tiga bagian utama yaitu *topic*, *broker* serta *client* berupa *publisher* dan *consumer* yang lebih dikenal sebagai *subscriber* (Ramos, Benito & Lacuesta, 2018). Tiga bagian dari MQTT tersebut mempunyai tujuan yaitu untuk mengumpulkan data dari sensor *publisher.* Pada protokol MQTT, *topic* memiliki peran yang sangat penting karena pada aplikasi protokol point-to-point data didistribusikan berdasarkan alamat IP jika pada *publish-subscribe* pengumpulan data dikelompokkan berdasarkan *topic* (Ramos, Benito & Lacuesta, 2018).

Pentingnya integritas data yang diterima dari hasil pembacaan sensor oleh publisher dibutuhkan metode pengecekan integritas data. Tiap topik pada protokol MQTT dapat terdiri berbagai data hasil pembacaan sensor. Untuk pengunaan pada budidaya ikan hias yaitu suhu, curah hujan dan transparansi air kolam untuk menentukan kondisi ideal kolam ikan tersebut (Bajaj, 2017). Oleh karena itu protokol MQTT mengirimkan data dalam bentuk plaintext yang membutuhkan pengecekan integritas data yang diterima, sebab data hasil monitoring akan mempengaruhi pertumbuhan dan produksi ikan hias tersebut (Bajaj, 2017). Agar data yang diterima tidak mengalami modifikasi atau pengubahan oleh pihak yang tidak berwenang, maka dibutuhkanlah metode pengecekan integritas data yang diterima oleh subscriber.

Pengecekan integritas data dilakukan melalui metode kriptografi berupa *Message Authentication Code* (MAC) yang dirikirim bersam dengan data yang dikirmkan (Aman, et al., 2018). Algoritme MAC bisa digamarkan melalui fungsi *hash* dengan masukkan berupa data dan *private key* yang digunakan (Mathews & V., 2016). Karena menggunakan *key* sebagai masukan, MAC juga bisa disebut dengan *keyed hash functions.* Dari pernyataan di atas dapat disimpulakan bahwa algoritme MAC dapat menjamin autentikasi yang sudah disediakan oleh MQTT

yang berupa *username* dan *password* ditransmisikan dalam bentuk *plaintext* (Chiwariro & Rajendran, 2018).

Algoritme *hash* atau *symmetric cipher* menjadi dasar dari pembangunan algoritme MAC (Pereira, et al., 2017). MAC yang dibangun dari algoritme *hash* disebut HMAC-X dengan X yang merupakan fungsi *hash* yang digunakan seperti pada HMAC-MD5 atau HMAC-SHA1. Meski dalam pengunaan HMAC MD5 dan SHA-1 yang sering digunakan mengalami kendala tidak aman dan rentan terhadap *brute force* (Chiriaco , Franzen, & Zhang, 2017) dan *collision attack* (FIPS Publication 180-2). Agar penggunaan MAC tidak menyebabkan terganggunya performa maka digunakanlah algoritme SHA-2 sebagai pembaruan dari SHA-1, karena tingkat kompleksitas yang lebih tinggi dan *message digest* yang dihasilkan lebih panjang menyebabkan SHA-2 aman terhadap *brute force* dan *collision attack*. Terdapat 4 hasil besar *digest* pesan dalam algoritme SHA-2 yaitu 224, 256, 384, dan 512 *bits.* Karena perbedaan besar *digest* pesan yang dihasilkan algoritme SHA-2 maka dipilihlah algoritme SHA-2 yang menggunakan 256 *bits* karena pemrosesan dalam pembutan *message digest* lebih cepat dari 512 *bits* (Sebastian, 2007).

Pentingnya integritas data yang ditransmisikan dari *publisher* ke *subscriber*, maka penulis mengambil tema "Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias".

1.2 Rumusan masalah

Sesuai dengan permasalahan yang telah dipaparkan pada bagian sebelumnya, maka rumusan masalah pada penelitian in yaitu:

1. Bagaimana merancang algoritme SHA-256 untuk pengecekan integritas data pada budidaya ikan hias ?
2. Bagaimana cara mengimplementasikan algoritme SHA-256 untuk pengecekan integritas data pada budidaya ikan hias ?
3. Bagaimana hasil dari algoritme SHA-256 untuk pengecekan integritas data pada budidaya ikan hias ?

1.3 Tujuan

Tujuan dari penelitian yang dilakukan dari permasalahan yang telah di kemukakan yaitu :

1. Algoritme SHA-256 dapat dirancang untuk melakukan implemtasi pada budidaya ikan hias untuk pengecekan integritas data.
2. Algoritme SHA-256 dapat di implementasikan pada budidaya ikan hias untuk pengecekan integritas data.
3. Algoritme SHA-256 dapat di implementasikan yang memberi performa baik untuk pengecekan integritas data pada budidaya ikan hias.

1.4 Manfaat

Manfaat yang dapat diperoleh dari penelitian ini adalah:

1. Penelitian ini diharapkan bisa menjadi referensi maupun acuan untuk melakukan pengecekan integritas data dari beberapa sensor dan integritas data yang diterima, terutama pada protokol MQTT.
2. Penelitian ini diharapkan dapat menjadi rujukan sebagai keamanan tambahan pada protokol MQTT jika SSL tidak memungkinkan untuk diterapkan karena keterbatasan sumber daya pada kebanyakan perangkat yang digunakan untuk kepentingan sistem IoT.

1.5 Batasan masalah

Agar penelitian dapat memberi hasil yang sesuai serta pembahasan penelitian dilakukan secara terarah, maka membutuhkan batasan permasalahan antara lain algoritme MAC yang diterapkan adalah SHA-256, protokol komunikasi yang digunakan yaitu MQTT, tidak ada pertukaran *key* ,serta pengecekan integritas data dilakukan pada *subscriber.*

1.6 Sistematika pembahasan

Uraian singkat mengenai metodologi penelitian pada masing-masing bab adalah sebagai berikut:

BAB I Pendahuluan

Bab ini terdiri dari latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan dari "Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias".

BAB II Landasan Kepustakaan

Membahas landasan pustaka dan penelitian sebelumnya yang berhubungan dengan latar belakang serta metode yang digunakan pada skripsi ini. Kajian pustaka ini berasal dari referensi-referensi berkaitan dan mendukung dalam penelitian ini seperti dokumen asli dari pembuat algoritme SHA-2, penerapan ptokol MQTT untuk pengiriman data sensor, dan penelitian tentang HMAC SHA-1 untuk integritas data.

BAB III Metodologi

Menguraikan mengenai metode serta langkah kerja penelitian yang terdiri atas studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian dan analisis serta pengambilan kesimpulan dan saran dari penelitian yang telah dilakukan tentang algoritme SHA-256 .

BAB IV Rekayasa Kebutuhan

Membahas tentang Gambaran umum sistem, dan kebutuhan sistem. Kebutuhan sistem meliputi kebutuhan fungsional , kebutuhan perangkat

keras, dan kebutuhan perangkat lunak yang digunakan untuk perancangan dan implementasi algoritme SHA-2 untuk integritas data.

BAB V Perancangan dan Implementasi

Menjelaskan tentang perancangan algoritme SHA-256 , perancangan sistem, perancangan pengujian , implementasi sistem dan implementasi sesuai dengan rekayasa kebutuhan.

BAB VI Pengujian dan Analisa

Pengujian dilakukan jika perancangan dan implementasi telah selesai dilakukan. Pengujian bertujuan untuk melakukan uji coba sistem, serta mengetahui kinerja sistem bahwa sistem sudah berjalan sesuai keinginan. Pengujian didalam sistem ini meliputi pengujian validitas algoritme SHA-256 , pengujian waktu esekusi algoritme SHA-256 , pengujian fungsional, pengujian performa sistem, dan pengujian keamanan.

BAB VII Penutup

Memuat kesimpulan dari pembuatan dan pengujian sistem, dan saran untuk pengembangan atau penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi kajian pustaka yang memuat berbagai teori dari beberapa penelitian terdahulu dan relevan dengan penelitian yang akan dilakukan. Serta dasar teori yang berisi teori penunjang dalam pengerjaan penelitian seperti dasar teori algoritme SHA-2, protokol MQTT, mikrokomputer Raspberry Pi 3, serta fungsi dari 3 sensor yang digunakan.

2.1 Tinjauan Pustaka

Kajian pustaka membahas penelitian yang pernah di lakukan yang berkaitan dengan penelitian yang di ajukan. Pada penelitian ini, kajian pustaka di ambil dengan menelaah beberapa penelitian yang berkaitan dan pernah dilakukan.

Tabel 2.1 Tinjauan Pustaka

No	Judul	Nama Peneliti	Perbedaan	
			Kajian Pustaka	Skripsi Penulis
1	FIPS Publication 180-2	National Security Agency.2002	Dokumen asli dari pembuat algoritme SHA-2 sebagai penyempurnaan dari SHA-1 yang sering terjadi *collison attack* pada saat proses *hashing* terjadi.	Penulis menggunakan Algoritme SHA-2 untuk integritas data dengan menggunakan metode *Message Authentication Code* (MAC).
2	Implementasi Penyimpanan Data Sensor Nirkabel dengan MongoDB pada Lingkungan IOT Menggunakan Protokol MQTT	Pramudya Mahardika Kusumawardhana	Penerapan protokol MQTT untuk media transmisi data dari sensor	Penulis menggunakan media transmisi data dari sensor yang diberi tambahan fitur MAC

Lanjutan Tabel 2.1 Tinjauan Pustaka

No	Judul	Nama Peneliti	Perbedaan	
			Kajian Pustaka	Skripsi Penulis
3	Enhancing the Security of MANETs Using *Hash* Algorithms	Dilli Ravillaa dan Chandra Shekar Reddy Putta	Penerapan HMAC SHA-1 untuk pengecekan integritas data pada MANETs	Penulis menggunakan HMAC SHA-2 untuk pengecekan integritas data data dari sensor yang diterima oleh *subscriber*.

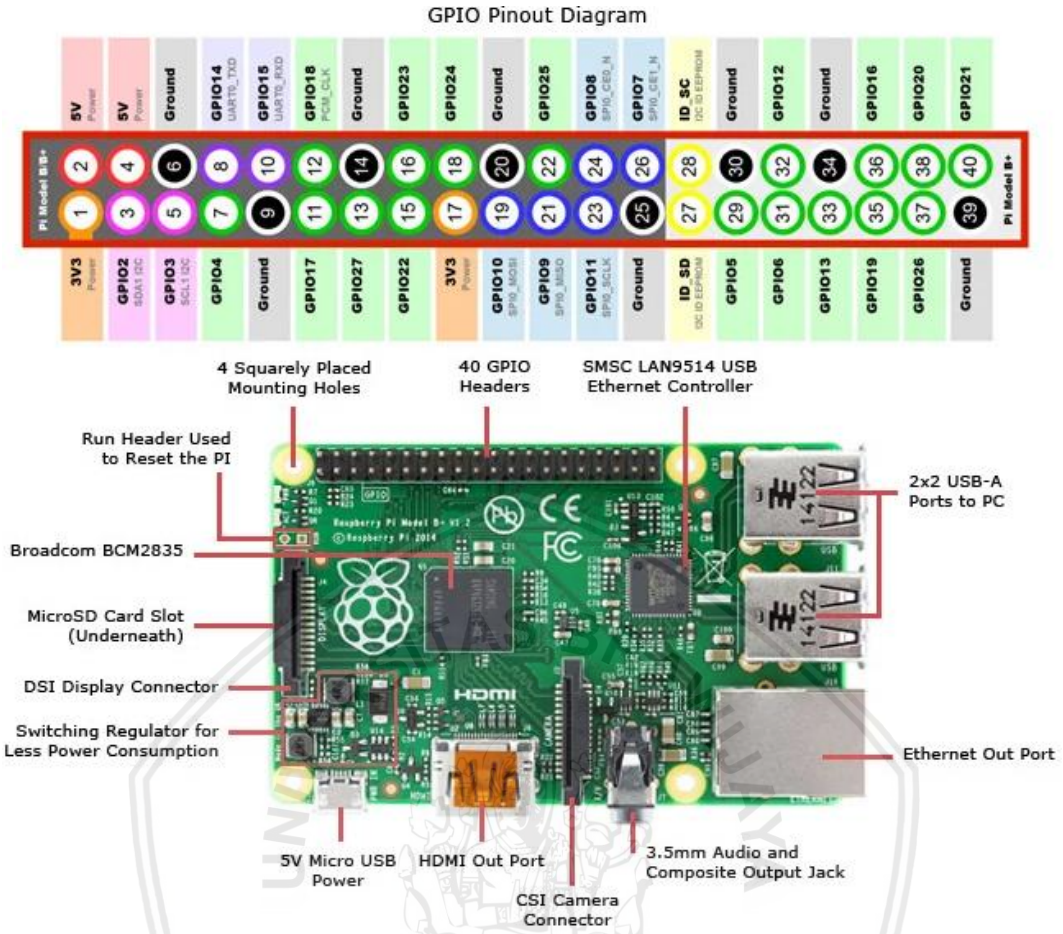
Penelitian ini membutukan tinjauan pustaka yang berisikan dokumen asli dari pembuat algoritme SHA karena di dalam penelitain ini menggunakan algoritme SHA-2 sebagai integritas data dengan metode *Message Authentication Code* (MAC), penerapan protokol MQTT sebagai media transmisi dari 3 sensor yang digunakan, dan algoritme SHA-2 yang tidak menyediakan fitur *Message Authentication Code* (MAC) maka diterpakan HMAC agar algoritme SHA-2 dapat menerima *key*.

2.2 Dasar Teori

Dasar teori membahas berbagai teori yang diperlukan dalam menyusun penelitian yang diusulkan. Dasar teori akan dibahas pada subbab 2.2.1 sampai 2.2.11.

2.2.1 Mikrokomputer Raspberry Pi 3

Raspberry Pi adalah komputer papan tunggal (*single-board circuit*; SBC) yang seukuran dengan kartu kredit yang dapat digunakan untuk menjalankan program perkantoran, permainan komputer, dan sebagai pemutar media hingga *video* beresolusi tinggi. Raspberry Pi bisa digunakan untuk pengontrolan lebih dari satu *device*, baik jarak dekat ataupun jarak jauh. Berbeda dengan mikrokontroler, Raspberry Pi dapat mengontrol lebih dari 1 unit *device* yang ingin dikontrol. Raspberry Pi memiliki pin GPIO (*General-purpose input/output*) yang berfungsi sebagai penghubung dengan perangkat yang akan dijadikan berbagai *project* penggabungan di *Raspberry Pi*seperti modul *relay*. Saat ini model *Raspberry Pi*mempunyai 3 versi model yaitu model A, B, dan B+ (Kulkarni, et al., 2017). Model Raspberry Pi versi 3 B ditunjukkan pada Gambar 2.1.



Gambar 2.1 Rasberry Pi 3 Model B
Sumber: (Jameco, 2018)

Raspberry Pi 3 model B memiliki spesifikasi prosesor ARMv8 64 bit 1.2Ghz quadcore Cortex A53, kapasitas RAM 1GB dengan kecepatan 900Mhz, 802.11n *wireless* LAN, Bluetooth v4.1, 1 *port* Ethernet, 4 *port* USB, 1 *port* HDMI, 1 *port* audio I/O, 40 pin GPIO yang dapat dikonfigurasikan sebagai digital *input* atau *output* (Imteaj, et al., 2017). Raspberry Pi memiliki sistem operasi yang dikembangkan khusus yaitu Raspbian, yang berbasis Debian Linux ARM. Selain sistem operasi Raspbian, beberapa sistem operasi seperti Windows IoT dan Linux untuk prosesor ARM juga dapat digunakan. Raspberry Pi mendukung bahasa pemrograman C++, Python, SQL, *Hyper Text Structured Query Language* (HTSQL), Java, Javascript dan lain-lain (Kulkarni, et al., 2017).

2.2.2 Sensor Suhu DS18B2O

Sensor suhu adalah alat yang digunakan untuk mengubah besaran panas menjadi besaran listrik yang dapat dengan mudah dianalisis besarnya. Ada beberapa metode yang digunakan untuk membuat sensor ini, salah satunya dengan cara menggunakan material yang berubah hambatannya terhadap arus listrik sesuai dengan suhunya.

Gambar 2.2 Sensor Suhu DS18B2O
Sumber : (Potentiallabs, 2018)

2.2.3 Sensor Ultrasonik HC-SR04

Sensor Ultrasonik HC-SR04 adalah sensor yang dirancang untuk mengubah energi listrik menjadi mekanik dalam bentuk gelombang suara ultrasonik. Sensor ini memiliki rangkaian pemancar gelombang ultrasonik (*transmitter*) dan rangkaian penerima gelombang ultrasonik (*receiver*). Sensor ini bekerja dengan memancarkan sinyal melalui pemancar gelombang ultrasonik, kemudian sinyal yang dipancarkan akan merambat sebagai bunyi dengan kecepatan 344 m/s. Dan yang terakhir sinyal yang diterima oleh *receiver* akan diproses untuk menghitung jaraknya dalam satuan *centimeter*(cm).

Gambar 2.3 Sensor Ultra Sonik
Sumber : (Peers, 2017)

2.2.4 Sensor Cahaya Photoresistor LDR GL5528

Sensor cahaya adalah alat yang digunakan dalam bidang elektronika yang berfungsi untuk mengubah besaran cahaya menjadi besaran listrik. Sensor cahaya LDR (*Light Dependent Resistor*) merupakan suatu jenis resistor yang peka terhadap cahaya. Nilai resistansi LDR akan berubah-ubah sesuai dengan intensitas cahaya yang diterima. Jika LDR tidak terkena cahaya maka nilai tahanan akan menjadi besar (sekitar 10MΩ) dan jika terkena cahaya nilai tahanan akan menjadi kecil (sekitar 1kΩ*)* pabrikasi modern.

Gambar 2.4 Sensor Cahaya Photoresistor LDR GL5528
Sumber : (Sparkfun, 2018)

2.2.5 Kriptografi

Kriptografi adalah sebuah bidang studi matematika yang berkaitan tentang teknik pengamanan informasi dengan menggunkan rumus matetmatika yang telah dirancang sedemikian rupa agar dapat menjamin keamanan informasi. Kriptografi sendiri memiliki arti tulisan rahasia, diambil dari bahasa yunani yaitu *cryptós* (rahasia) dan *gráphein* (tulisan). Pada awalnya kriptografi bekerja dengan cara menjadikan sandi dari pesan asli yang dapat dimengerti kedalam bentuk yang tidak lagi dapat dimengerti, hal ini untuk menunjang komunikasi antar suatu golongan agar tidak dimengerti oleh golongan lain terlebih oleh musuh, namun saat ini kriptografi tidak lagi digunakan untuk sekedar menjaga kerahasiaan (*Confidentiality*) namun juga untuk menjaga integritas data (*Integrity*) (Menezes, et al., 1996).

2.2.6 Integritas Data

Integritas data adalah bagian tujuan kriptografi yang berfungsi untuk pemeliharaan , kepastian akurasi dan konsistensi data. Integritas data sangat penting dalam berbagai aspek seperti desain, implementasi dan penggunaan sistem yang menyimpan, memproses, atau mengambil data. Teknik Integritas data adalah memastikan data yang di catat persis seperti pengambilan data dan memastikan data yang sama saat data direkam, integritas data bertujuan untuk mencegah perubahan terhadap informasi yang disampaikan. Metode kriptografi untuk mendeteksi manipulasi data ialah *hashing*. *Hashing* bekerja dengan cara mengubah sebuah pesan atau dalam kriptografi disebut *message* menggunakan perhitungan matematika menjadi beberapa karakter tidak bisa dibaca yang mewakili nilai dari sebuah pesan yang disebut sebagai *digest* (Menezes, et al., 1996).

2.2.7 Algoritme SHA-2

SHA-2 (*Secure Hash Algorithm 2*) adalah seperangkat fungsi *hash* kriptografi yang dirancang oleh *United States National Security Agency* (NSA) . Mereka dibangun menggunakan struktur Merkle-Damgård , dari fungsi kompresi *One-way* yang dibangun menggunakan struktur Davies-Meyer dari *block cipher* khusus (diklasifikasikan). Algoritme SHA-2 merupakan algoritme pengembangan

dari algoritme SHA-1. Jenis SHA-2 dibagi menjadi 4 berdasarkan hasil keluaran yaitu 224, 256,384, dan 512.

Setiap algoritme dibagi menjadi 2 tahap yaitu *preprocessing* dan *hash computation*. Pada tahap *preprocessing* dimulai dari dengan memberikan *padding* pada pesan, membagi pesan yang sudah dibagi menjadi m-*bit* blok dan inisialisasi nilai yang akan digunakan saat *hash computation. Hash computation* membuat *message schedule* dari pesan yang telah diberi *padding* dan menggunakan *message schedule* tersebut bersama dengan fungsi *hash,* konstan dan operasi kalimat secara literatif untuk menghasilkan *hash value* atau *digest.*

2.2.8 Algoritme SHA-256

Algoritme SHA-256 digunakan untuk membuat *message digest* dimana panjang *message* harus diantara $0 \leq \ell \leq 2^{64}$. Algoritme ini menggunakan sebuah *message schedule* yang terdiri dari 64 elemen 32-*bit word* yang diberi label *W0, W1…W63.* Delapan buah variabel 32-*bit* yang diberi label *a, b, c, d, e, f, g, h.* Variabel penyimpanan nilai *hash* berjumlah 8 buah *word* 32-*bit* yang diberi label H_0^i, H_1^i,… H_7^i.

Algoritme SHA-256 mengubah masukkan ke dalam *message digest* 256 *bit.* Berdasarkan *Secure Hash Signature Standard,* pesan masukkan yang panjangnya lebih pendek dari 2^{64} , harus dioperasikan oleh 512 *bit* dalam kelompok dan menjadi sebuah *message digest* 256 *bit.*

2.2.9 *Keyed-Hash Message Authentication Code* (HMAC)

Keyed-Hash Message Authentication Code (HMAC) adalah jenis spesifik dari kode autentikasi pesan (MAC) yang melibatkan fungsi *hash* kriptografi dan kunci kriptografi rahasia . Ini dapat digunakan untuk secara bersamaan memverifikasi baik integritas data dan autentikasi dari pesan , karena dengan MAC. Setiap fungsi *hash* kriptografi, seperti MD5 atau SHA-2, dapat digunakan dalam perhitungan HMAC. Algoritme MAC yang dihasilkan disebut HMAC-X, di mana X adalah fungsi *hash* yang digunakan (misalnya HMAC-MD5 atau HMAC-SHA-2). Kekuatan kriptografi HMAC bergantung pada kekuatan kriptografi dari fungsi *hash* yang mendasarinya, ukuran keluaran *hash*-nya, dan ukuran dan kualitas kuncinya. HMAC memberikan nilai integritas data dan autentikasi dalam kriptografi tetapi HMAC tidak memberikan nilai kerahasiaan pesan karena HMAC tidak mengenkripsi pesan.

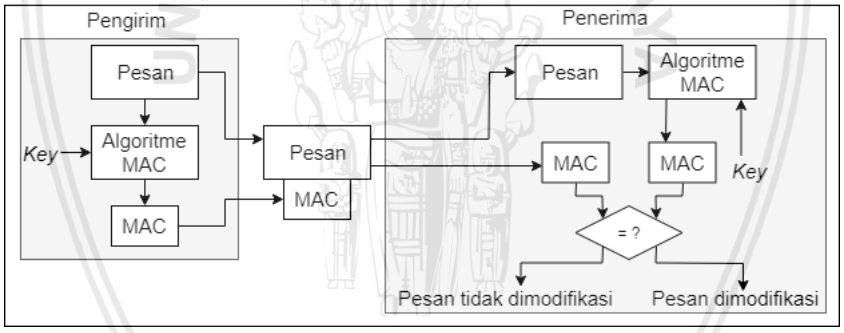
2.2.10 *Message Authentication Code* (MAC)

Message Authentication Code (MAC) merupakan metode kriptografi yang digunakan untuk melakukan verifikasi bahwa pesan yang diterima tidak mengalami perubahan maupun modifikasi saat ditransmisikan. Pada saat proses pembuatan MAC oleh pengirim pesan, dibutuhkan sebuah *key* atau kunci yang juga harus diketahui oleh penerima pesan agar dapat dilakukan pengecekan ulang integritas data yang diterima. Oleh sebab penggunaan kunci yang sama oleh pengirim maupun penerima, maka MAC termasuk *symmetric-key cryptography*.

Penggunaan kunci yang sama antara penerima dan pengirim pesan ini membedakan MAC dengan *digital signature*. Pada *digital signature*, terdapat *private key* yang hanya diketahui oleh masing-masing pihak dan *public key* yang harus diketahui baik oleh pengirim maupun penerima. Algoritme MAC menghasilkan *output* MAC yang berbeda untuk setiap pesan berbeda pula karena MAC dibuat dari pesan tersebut dengan penambahan kunci yang diketahui oleh pengirim maupun penerima. Persamaan proses pada saat pembuatan MAC ditunjukkan oleh Persamaan 2.1 dimana MAC adalah *output* yang dihasilkan, C adalah algoritme yang digunakan dan M adalah pesan yang dicari nilai MAC-nya.

$$MAC = C_k(M) \tag{2.1}$$

Setiap mengirim pesan, hasil *output* MAC akan ditambahkan dengan pesan yang dikirim oleh pengirim. Sedangkan penerima pesan juga akan mencocokan MAC yang diterima dari pengirim dengan MAC yang dihasilkan. Jika MAC dari pengirim cocok dengan MAC yang dihasilkan penerima, maka pesan tersebut merupakan pesan yang tidak terjadi modifikasi saat pesan ditransmisikan. seperti yang ditunjukkan pada Gambar 2.5. Proses ini memungkinkan karena MAC yang aman tidak dapat dipalsukan dan tidak dapat dilakukan komputasi untuk menghasilkan MAC.



Gambar 2.5 Cara Kerja MAC

2.2.11 Protokol MQTT

Protokol Message Queue Telemetry Transport (MQTT) adalah salah satu protokol komunikasi yang banyak digunakan untuk pertukaran data antar perangkat IoT. MQTT merupakan protokol berbasis arsitektur publish-subscribe yang sangat sederhana penerapanya serta rendah dalam konsumsi daya maupun penggunaan sumber dayanya (Ramos, Benito & Lacuesta, 2018). MQTT juga merupakan protokol yang memberi reliabilitas dan efisiensi dalam berbagai macam kondisi, hal inilah yang menyebabkan pertumbuhan pengguna MQTT kian meningkat.

Berdasarkan pengelompokan Faktornya, protokol MQTT memiliki tiga kelompok aktor perangkat yaitu *publisher* sebagai pengirim data, *consumer* atau yang lebih dikenal sebagai *subscriber*, dan *broker* yang menghubungkan antara *publisher* dengan *subscriber* berdasarkan *topic*. Peran *topic* pada MQTT yaitu

sebagai alamat identifikasi kepada siapa data dikirimkan oleh *publisher* dan di mana data dapat diakses oleh *subscriber*. Data yang diteruskan berdasarkan *topic* pada MQTT sendiri diatur oleh perangkat yang berperan sebagai MQTT *broker*. Selain *topic*, MQTT juga memiliki bermacam-macam tipe pesan yang ditunjukkan oleh Tabel 2.2.

Tabel 2.2 Tipe Pesan MQTT

Paket	Deskripsi
CONNECT	*Request* koneksi ke *broker*
CONNACK	ACK dari tipe pesan *connect*
PUBLISH	Publish kepada suatu *topic*
PUBACK	ACK dari tipe pesan *publish*
PUBREC	Pesan *publish* telah diterima
PUBREL	Pesan *publish* telah terkirim
PUBCOMP	Pesan *publish* telah diselesaikan
SUBSCRIBE	*Request subscribe* terhadap suatu *topic*
SUBACK	ACK dari tipe pesan *subscribe*
UNSUBSCRIBE	*Request* menghentikan *subscribe terhadap suatu topic*
UNSUBACK	ACK dari tipe pesan *unsubscribe*

BAB 3 METODOLOGI

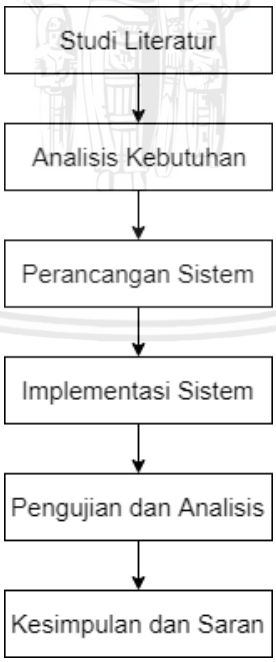
Bab ini membahas metodologi yang dilakukan agar dapat melakukan Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias yang meliputi implementasi keseluruhan dari komponen perangkat keras dan perangkat lunak.

3.1 Jenis Penelitian

Jenis penelitian yang dilakukan pada penelitian ini adalah implementasi perancangan sistem. Hasil akhir dari penelitian ini adalah sebuah *prototype* sesuai dengan tujuan penelitian, yaitu algoritme SHA-2 yang diterapkan sebagai mekanisme pengecekan integritas data yang diperoleh dari *publisher* serta rangkaian komponen lainnya yang mendukung berjalannya sistem.

3.2 Metodologi Penelitian

Metodologi penelitian membahas mengenai langkah-langkah sistematis yang dilakukan pada penelitian. Alur pelaksanaan metode penelitian untuk menyelesaikan permasalahan yang dibahas pada penelitian ini direpresentasikan melalui diagram alir metodologi penelitian. Terdapat lima proses yang dilakukan secara bertahap sesuai pada Gambar 3.1, yaitu studi literatur, analisis kebutuhan, perancangan, implementasi, pengujian dan analisis, serta pengambilan kesimpulan dan saran.



Gambar 3.1 Metodologi

3.3 Studi Literatur

Studi literatur adalah tahap untuk mencari dan menyusun teori dasar serta referensi penunjang yang mendukung implementasi sistem pada penelitian ini. Studi literatur pada penelitian ini diperoleh dari jurnal, *paper*, dan sumber lain yang meliputi:

1. Teori mengenai algoritme SHA-2 untuk pengecekan integritas data.
2. Teori mengenai cara kerja protokol MQTT yang memiliki arsitektur *publish-subscribe*.
3. Teori mengenai ketinggian air, suhu air, keasaman air, dan keruh tidaknya air.

3.4 Analisis Kebutuhan

Analisis kebutuhan dilakukan dengan tujuan untuk melakukan analisis kebutuhan yang diperlukan sistem meliputi *software* dan *hardware* berdasarkan literatur agar memudahkan untuk mendesain dan mengimplementasikan sistem yang dibuat.

3.5 Perancangan

Perancangan sistem merupakan tahap yang dilakukan setelah melakukan studi literature dan juga analalisis kebutuhan. Di dalam perancangan dibagi menjadi 3 yaitu perancangan sistem, perancangan algoritme SHA-256 dan perancangan pengujian. Perancangan sistem meliputi perancangan *publisher* dan perancangan *subscriber*. Perancangan algoritme SHA-256 meliputi Perancangan HMAC SHA-256 *,* Perancangan SHA-256 , dan Perancangan *preprocessing* SHA-256 . Perancangan pengujian validitas , pengujian waktu esekusi, pengujian fungsional, pengujian performa, dan pengujian keamanan.

3.6 Implementasi

Tahap implementasi sistem akan dilakukan penerapan penggunaan algoritme SHA-2 untuk pengecekan integritas data sensor yang diterima oleh *subscriber* sesuai dengan rekayasa kebutuhan dan perancangan. Adapun sistem yang akan implementasikan meliputi:

1. Perancangan diagram cara kerja sistem.
2. Pengimplementasian pembacaan data dari sensor.
3. Pengimplementasian MQTT *publisher* pada sistem.
4. Pengimplementasian MQTT *broker* pada *Guest* OS Ubuntu Server.
5. Pengimplementasian MQTT *subscriber* pada *Guest* OS Ubuntu Server.
6. Pengimplementasian algoritme MAC SHA-256 baik pada *publisher* maupun *subscriber*.

3.7 Pengujian dan Analisis

Pengujian dan analisis sistem yang dilakukan meliputi pengujian validitas algoritme, pengujian waktu eksekusi algoritme, pengujian fungsional sistem,

pengujian performa sistem, serta pengujian keamanan sistem berdasarkan skenario pengujian. Pengujian validitas algoritme dilakukan untuk menguji kesesuaian algoritme SHA-2 yang diimplementasikan menggunakan *test vector* yang sudah ditetapkan oleh pembuat algoritme SHA-2. Pengujian fungsional sistem dilakukan untuk memastikan bahwa sistem yang diterapkan dapat berjalan dari awal hingga akhir. Pengujian performa sistem merupakan pengujian penggunaan *memory* pada *publisher* dan *subscriber*. Sedangkan pengujian keamanan sistem merupakan pengujian dengan melakukan serangan MITM pada sistem.

3.8 Kesimpulan dan Saran

Pengambilan kesimpulan dilaksanakan setelah tahap perancangan, implementasi, pengujian dan analisis sistem dilakukan. Kesimpulan diambil berdasarkan hasil dari pengujian dan analisis sistem yang dibuat untuk menjawab masalah yang telah dirumuskan. Sedangkan Saran diambil berdasarkan Kesimpulan yang memuat masukkan atas kekurangan terhadap sistem, dengan harapan dapat menjadi acuan untuk pertimbangan penelitian mendatang.

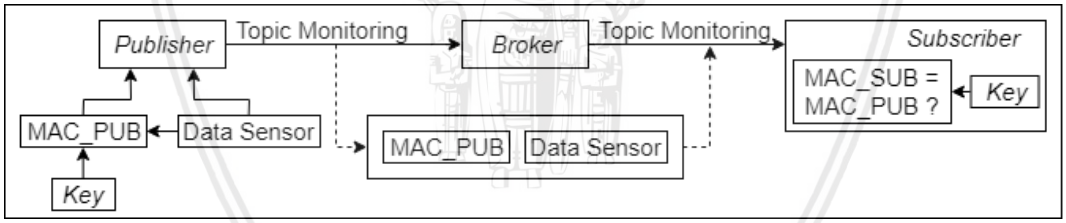
BAB 4 REKAYASA KEBUTUHAN

Bab ini menjelaskan tentang kebutuhan yang akan digunakan untuk membuat sisitem yang di buat. Rekayasa kebutuhan sistem dibagi menjadi dua yaitu Gambaran umum sistem, dan kebutuhan sistem.

4.1 Gambaran Umum Sistem

Berdasarkan Gambaran umum sistem, alur dari jalanya sistem diawali dengan pengambilan data dari 3 sensor pada *publisher* lalu melakukan *publish* data sensor tersebut ke *topic* Monitoring. Oleh karena penelitian ini berfokus pada pengecekan integritas data, maka *publisher* menghitung nilai MAC berdasarkan *key* dan data sensor terlebih dahulu. Setelah nilai MAC didapatkan, barulah *publisher* dapat melakukan *publish* data dengan format pengiriman (*payload)* berupa MAC publisher dan data sensor. Data yang di-*publish* tersebut selanjutnya akan diteruskan ke *subscriber* oleh *broker*.

Kondisi awal yang harus dipenuhi *subscriber* yaitu harus mengetahui dan menyimpan *key* milik *publisher*. Jika *subscriber* telah memiliki *key publisher*, maka pengecekan integritas data sensor yang diterima baru dapat dilakukan. Hanya data sensor yang tidak mengalami pengubahan saja yang ditampilkan oleh subscriber. Integritas data data sensor didapatkan dengan membandingkan nilai MAC yang dihasilkan oleh *subscriber* dengan nilai MAC publisher yang disertakan dalam *payload*. Keseluruhan dari Gambaran umum sistem ditunjukkan pada Gambar 4.1.



Gambar 4.1 Gambaran Umum Sistem

4.2 Kebutuhan Sistem

Kebutuhan sistem menjelaskan tentang kebutuhan fungsional, kebutuhan perangkat keras (*hardware*) dan perangkat lunak (software). Kebutuhan sistem dianalisis agar memudahkan untuk mendesain dan mengimplementasikan sistem.

4.2.1 Kebutuhan Fungsional

Analisis kebutuhan fungsional dilakukan untuk menganalisa kebutuhan yang harus dimiliki oleh sistem. Kebutuhan fungsional sistem yang akan dicapai sesuai penggunaan algoritme SHA-2 sebagai metode pengecekan integritas data :

1. Data dari masing- masing sensor dapat dibaca oleh Raspberr Pi.
2. Raspberry Pi 3 yang digunakan sebagai *publisher* dapat mem-*publish* data dari ke 3 sensor yang digunakan.

3. *Publisher* dan *subscriber* dapat menghasilkan kode MAC yang diperoleh dari algoritme SHA-256 .
4. *Publisher* dapat mengirim kode MAC dan data sensor dalam format JSON.
5. *Subcriber* dapat membaca data JSON yang diterima yang berupa kode MAC dan data sensor.
6. *Subcriber* dapat melakukan pengecekan integritas data sensor yang diterima berdasarkan kode MAC *publisher* yang didapatkan.

4.2.2 Kebutuhan Perangkat Keras

Berikut ini adalah kebutuhan perangkat keras yang harus dimiliki oleh sistem, yaitu :

1. Personal Computer (PC)
   Host OS yang terdiri dari guest OS *broker* dan *subscriber* dengan spesifikasi :
   Model Perangkat : ASUS A46CB
   Prosesor : prosesor i3 3217U dan DDR3 4GB
2. Raspberry Pi 3 Model B
   Mikrokomputer yang berfungsi sebagai *publisher* dan membaca *input* data dari sensor yang digunakan.
3. MicroSD 16GB
   Media penyimpanan untuk Raspberry Pi 3 yang berisikan OS beserta data lainnya seperti kode program.
4. Sensor Suhu DS18B2O
   Sensor suhu DS18B2O digunakan untuk mendapatkan data nilai temperatur suhu di sekitar jangkauan sensor.
5. Sensor Ultrasonik HC-SR04
   Sensor ultrasonik HC-SR04 digunakan untuk mendapatkan data nilai ketinggian pada jangkauan sensor.
6. Sensor Cahaya Photoresistor LDR GL5528
   Sensor cahaya Photoresistor LDR GL5528 digunakan untuk mendapatkan nilai gelap dan terang yang dibaca oleh sensor tersebut.

4.2.3 Kebutuhan Perangkat Lunak

Berikut ini adalah kebutuhan perangkat lunak yang harus dimiliki oleh sistem, yaitu :

1. Sistem operasi Windows 10 Pro 64-bit
   Host OS untuk virtualisasi virtualbox.
2. Sistem operasi Raspbian Stretch
   Sistem operasi asli dari Raspberry Pi.
3. VirtualBox (Versi 5.1.34)
   Software virtualisasi untuk *guest* ubuntu server.
4. Ubuntu Server (Versi 17.10.1)
   OS yang digunakan sebagai *guest* OS yang berfungsi sebagai *broker* dan *subscriber*.

5. Sublime Text 3
   Sublime Text digunakan untuk membuat kode program yang akan di implementasikan pada *publisher* dan *subscriber*.
6. Python3
   Bahasa pemrograman yang digunakan unutk implemtasi algoritme, *publisher*, dan *subscriber*.
7. MobaXtem (Versi 10.5)
   SSH Client berfungi untuk melakukan koneksi *remote shell* ke *publisher*, *broker*, dan *subscriber*.
8. Mosquitto (Versi 1.4.12)
   Perangkat lunak MQTT *broker*.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan tentang peracangan dan implementasi dari sistem yang terdiri dari perancangan perangkat keras, perancangan algoritme SHA-256 *,* perancangan sistem, dan perancangan pengujian. Implementasi berisi tentang implementasi algoritme SHA-256 , implementasi sistem.

5.1 Perancangan

Perancangan dalam penelitian ini berisikan perancangan sistem, perancangan algoritme SHA-256 dan perancangan pengujian.

5.1.1 Perancangan Sistem

Perancangan sistem berisi tentang alur kerja sistem SHA-2 dalam melakukan integritas datanya. Perancangan sistem terdiri dari perangkat keras dan perangkat lunak yang sudah di analisis untuk kebutuhan sistem tersebut. Perancangan sistem di bagi berdasarkan antar komponen saat berinteraksi, yaitu *publisher* dan *subscriber.* Setiap komponen yang berinteraksi tersebut mempunyai fungsi dan perintah yang dijelaskan pada Tabel 5.1.

Tabel 5.1 Pengertian Fungsi dan Perintah di Perancangan Sistem

Fungsi / Perintah	Parameter	Deskripsi
MQTT CONNECT	1. *clientId* 2. *cleanSession* 3. *lastWillTopic* 4. *lastWillQoS* 5. *lastWillMessage* 6. *lastWillRetain* 7. *keepAlive*	1. *Request* koneksi dari *client*, yaitu *publisher* atau *subscriber* ke *broker*. 2. Parameter *lastWillTopic*, *lastWillQoS*, maupun *lastWillMessage*, dan *lastWillRetain* bersifat opsional. 3. *clientId* harus unik untuk setiap *broker*, karena dijadiakn identifikasi dengan tipe *string* untuk *publisher* dan *subscriber*. 4. *cleanSession* merupakan tipe data *boolean* yang mengindikasi bahwa *publisher* atau *subscriber* ingin membentuk sesi persisten atau tidak. 5. *keepAlive* merupakan parameter *time interval* agar *client* tetap terkoneksi dengan mengirimkan *ping* ke *broker*.

Lanjutan Tabel 5.1 Pengertian Fungsi dan Perintah di Perancangan Sistem

Fungsi / Perintah	Parameter	Deskripsi
MQTT CONNACK	1. *sessionPresent* 2. *returnCode*	1. *Response* dari *broker* atas *request* koneksi *publisher*. 2. *sessionPresent* mengindikasikan jika *broker* telah memiliki sesi persisten *client* dari interaksi sebelumnya. 3. *returnCode* mengindikasikan berhasil atau tidaknya koneksi yang diminta oleh *client*. *returnCode* memiliki tipe data *integer*, yaitu: 0 jika koneksi diterima dan 1 sampai 5 jika koneksi ditolak.
MQTT PUBLISH	1. *topicName* 2. *qos* 3. *retainFlag* 4. *payload* 5. *dupFlag*	1. *publish* berfungsi untuk mengirimkan pesan ke *broker*. 2. *topicName* mengindikasikan nama *topic* yang menjadi tujuan pengiriman pesan, merupakan string dengan hirarki yang dipisahkan olah karakter "/". 3. *Quality of Service* (QoS) merupakan tingkat kemungkinan sampainya pesan, merupakan *integer* dari 0 hingga 2. 4. *retainFlag* memiliki tipe data *boolean* yang mengindikasikan disimpan atau tidaknya pesan pada *broker*. 5. *payload* merupakan isi pesan sebenarnya. 6. *dupFlag* mengindikasikan bahwa pesan merupakan pesan duplikat yang dikirimkan ulang karena saat pengiriman pesan asli tidak menerima ACK.

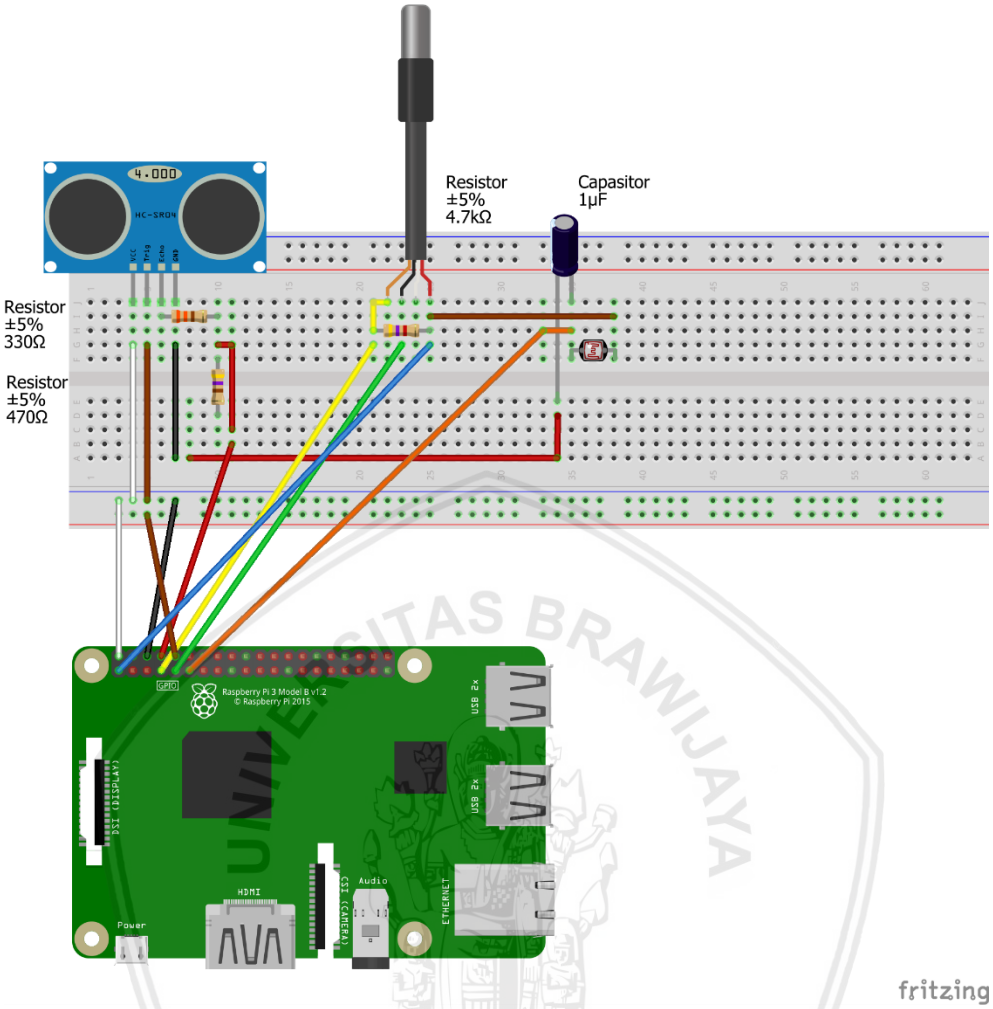
Lanjutan Tabel 5.1 Pengertian Fungsi dan Perintah di Perancangan Sistem

Fungsi / Perintah	Parameter	Deskripsi
MQTT SUBSCRIBE	1. *topicName* 2. *qos*	1. *subscribe* berfungsi untuk menerima pesan yang di-*publish*. 2. *topicName* mengindikasikan nama *topic* dari pesan yang ingin diterima, merupakan string dengan hirarki yang dipisahkan olah karakter "/". 3. *Quality of Service* (QoS) merupakan tingkat kemungkinan diterimanya pesan, merupakan *integer* dari 0 hingga 2.
MQTT SUBACK	*returnCode*	1. *Response* dari *broker* atas pesan *subscribe*. 2. *returnCode* memiliki tipe data *integer* 0 sampai 2 jika pesan *subscribe* berhasil dan 128 jika pesan *subscribe* gagal.
Data (sensor 1 + sensor 2 + sensor 3)		Mengambil data dari 3 sensor yang digunakan.

5.1.1.1 Perancangan *Publisher*

a. Perancangan Perangkat Keras *Publisher*

Perancangan perangkat keras *publisher* yang digunakan sebagai input data dari sensor yang diterapkan pada algoritme SHA-2 untuk integritas data. *Input* dari sensor Ultrasonik HC-SR04, sensor Suhu DS18B2O, dan sensor Cahaya Photoresistor LDR GL5528 digunakan oleh *publisher* sebagai data yang *outputnya* di kirim ke *subscriber,* dimana data yang dikirimkan *publisher* menggunakan integritas data akan di cek oleh *subscriber* ada atau tidaknya data yang diubah saat dikirim.



Gambar 5.1 Skema Perancangan Sensor *Publisher*

Pada Gambar 5.1 menunjukan skema perancangan sensor yang digunakan untuk pengambilan data oleh *publisher*. Serta koneksi *port* yang digunakan sensor pada Raspberry Pi ditunjukkan pada Tabel 5.2 dan bagian yang di arsir merupakan *port* yang tidak digunakan pada perancangan sensor *publisher.*

Tabel 5.2 Koneksi Pin Perangkat Keras *Publisher*

NO	Pin Raspberry Pi 3	Pin Ultrasonik	Pin Sensor DS18B2O	Pin Sensor Cahaya Photoresistor
1	3.3v DC Power		VCC	VCC
2	5v DC Power	VCC		
3	GPIO02 (SDA1, I^{2}C)			
4	5v DC Power			
5	GPIO03 (SCL1, I^{2}C)			
6	Ground	Ground		Ground

Lanjutan Tabel 5.2 Koneksi Pin Perangkat Keras *Publisher*

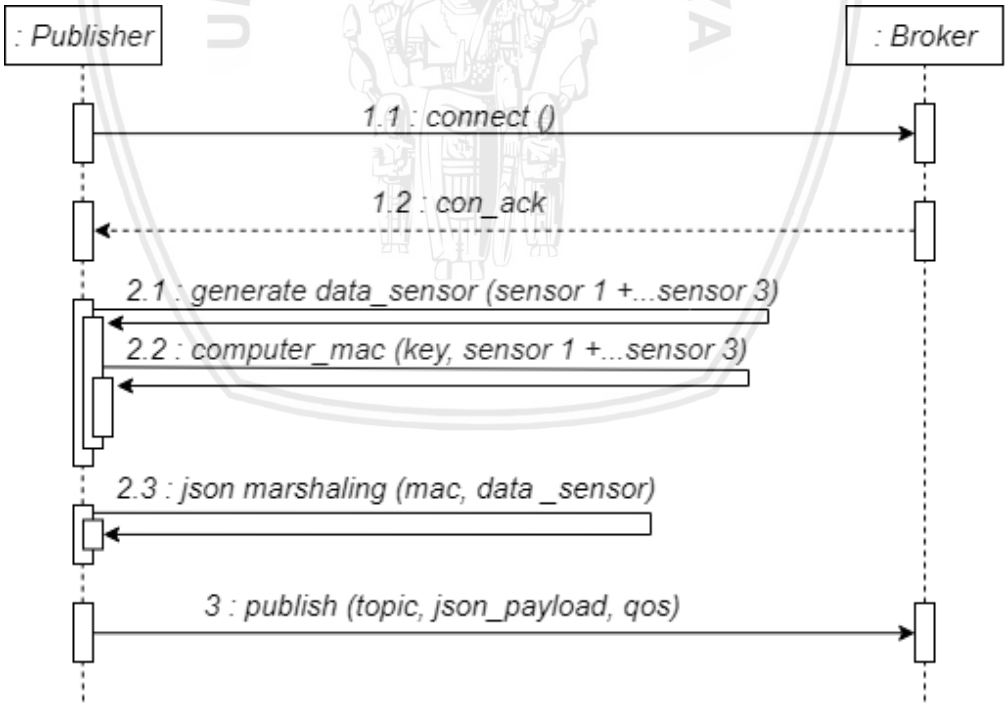
NO	Pin Raspberry Pi 3	Pin Ultrasonik	Pin Sensor DS18B2O	Pin Sensor Cahaya Photoresistor
7	GPIO04 (GPIO, GCLK)		DATA	
8	GPIO14(TXDO)	Echo		
9	Ground		Ground	
10	GPIO15(RXDO)	Trigger		
11	GPIO17(GPIOGEN0)			DATA
12	GPIO18(GPIOGEN01)			
13	GPIO27(GPIOGEN02)			
14	Ground			
15	GPIO22(GPIOGEN03)			
16	GPIO23(GPIOGEN04)			
17	3,3v DC Power			
18	GPIO24(GPIOGEN05)			
19	GPIO10(SPI_MOSI)			
20	Ground			
21	GPIO09(SPI_MISO)			
22	GPIO25(GPIOGEN06)			
23	GPIO11(SPI_CLK)			
24	GPIO08(SPI_CE0_N)			
25	Ground			
26	GPIO07(SPI_CE1_N)			
27	ID_SD(I^2C ID EEPROM)			
28	ID_SC(I^2C ID EEPROM)			
29	GPIO05			
30	Ground			
31	GPIO06			
32	GPIO12			
33	GPIO13			
34	Ground			

Lanjutan Tabel 5.2 Koneksi Pin Perangkat Keras *Publisher*

NO	Pin Raspberry Pi 3	Pin Ultrasonik	Pin Sensor DS18B2O	Pin Sensor Cahaya Photoresistor
35	GPIO19			
36	GPIO16			
37	GPIO26			
38	GPIO20			
39	Ground			
40	GPIO21			

b. **Perancangan Perangkat lunak *Publisher*.**

Perancangan perangkat lunak *publisher* berfungsi sebagai alur kerja proses saat melakukan *publish* data dari 3 sensor yang digunakan. Terdapat tiga proses yang terjadi pada *publisher* saat mengirimkan data ke *broker ,* yaitu melakukan koneksi ke *broker ,* mengambil data dari 3 sensor yang digunakan, dan melakukan *publish* ke *broker.* Proses tersebut dapat di lihat pada Gambar 5.2.



Gambar 5.2 Perancangan *Publisher*

Proses pertama yang sesuai pada Gambar 5.2 yaitu membuat koneksi ke *broker* dengan parameter berupa alamat IP dan nomor *port* yang digunakan *broker* serta parameter opsional seperti *cleanSession* dan *keepAliveInterval.* Setelah itu *broker* memberi balasan berupa pesan *con_ack.* Proses selanjutnya yaitu

pembacaan data dari 3 sensor yang digunakan. Data sensor yang dihasilkan kemudian digunakan sebagai masukan saat pembuatan MAC dengan *key* yang telah ditentukan. *Publish* data dapat dilakukan jika data telah diubah menjadi format JSON karena terdapat dua data yang akan di-*publish* yaitu MAC dan data sensor.

5.1.1.2 Perancangan *Subscriber*

a. Perancangan Mesin Virtual

Perancangan mesin virtual pada *subscriber* dan *broker* menggunakan virtualBox sebagai virtualisasi untuk *guest* ubuntu server yang mempunyai konfigurasi yang ditunjukkan pada Tabel 5.3.

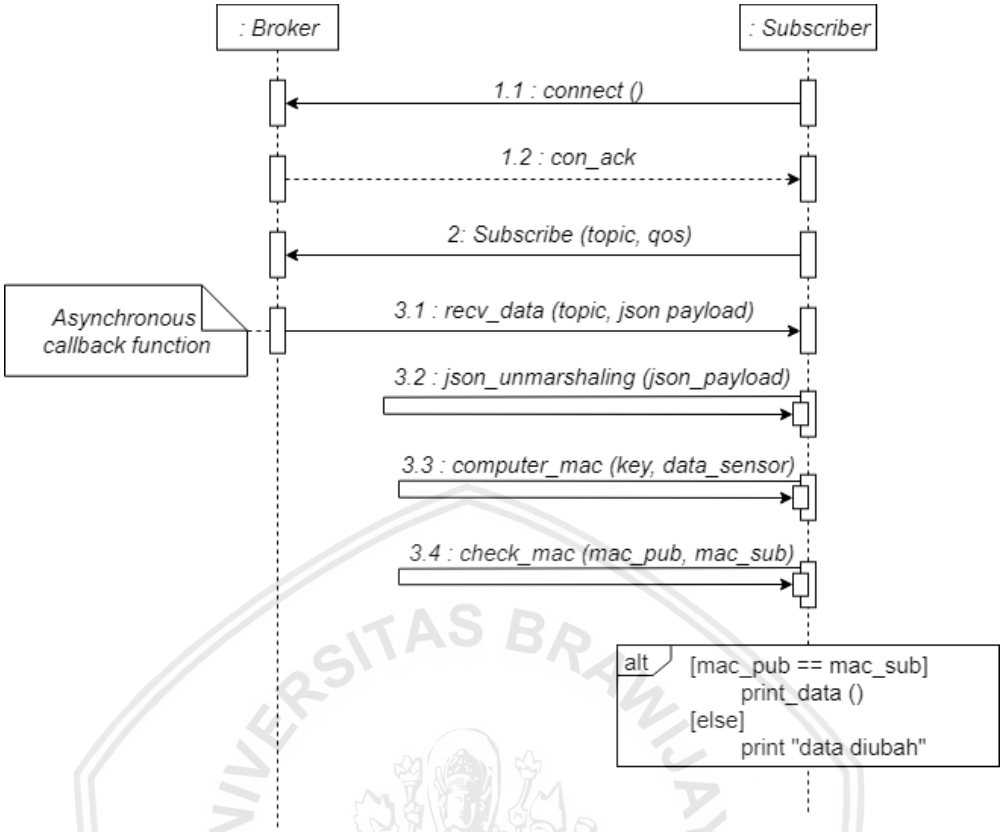
Tabel 5.3 Konfigurasi Mesin Virtual

	Sistem		Penyimpanan	Jaringan
	Memori	Prosesor		
Kapasitas	1 GB	1 CPU	10 GB	2 Adapter

Pada konfigurasi jaringan menggunkan 2 macam mode jaringan yaitu NAT dan Bridged Adapter. NAT berfungsi untuk menghubungkan langsung ke OS host yaitu windows, agar *guest* ubuntu server mendapatkan koneksi internet daroi OS host. Sedangkan Bridged Adapter digunakan *guest* ubuntu server agar dapat mengirim dan menerima data.

b. Perancangan Perangkat Lunak *Subscriber*

Perancangan perangkat lunak *subscriber* digunakan untuk menggambarkan alur maupun proses saat melakukan *subscribe*. Terdapat tiga proses bertahap yang terjadi pada *subscriber*. Ketiga proses tersebut ditunjukkan melalui Gambar 5.3.



Gambar 5.3 Perancangan *Subscriber*

Berdasarkan *sequence* diagram perancangan *subscriber* yang ditunjukkan pada Gambar 5.3. Proses interaksi pertama yang dilakukan yaitu pembuatan koneksi ke *broker* dengan parameter alamat IP dan nomor port yang digunakan broker, dan parameter opsional seperti *cleanSession* dan *keepAliveInterval*. Jika koneksi ke *broker* berhasil, maka broker memberi balasan berupa pesan *con_ack*.

Proses selanjutnya yaitu *subscriber* melakukan *subscribe* dengan mengirim pesan *subscribe* ke broker pada *topic* dan *qos* yang telah ditentukan. Saat melakukan subscribe, terdapat fungsi *callback* yang bersifat *asynchronous* dan akan dijalankan saat sebuah data diterima.

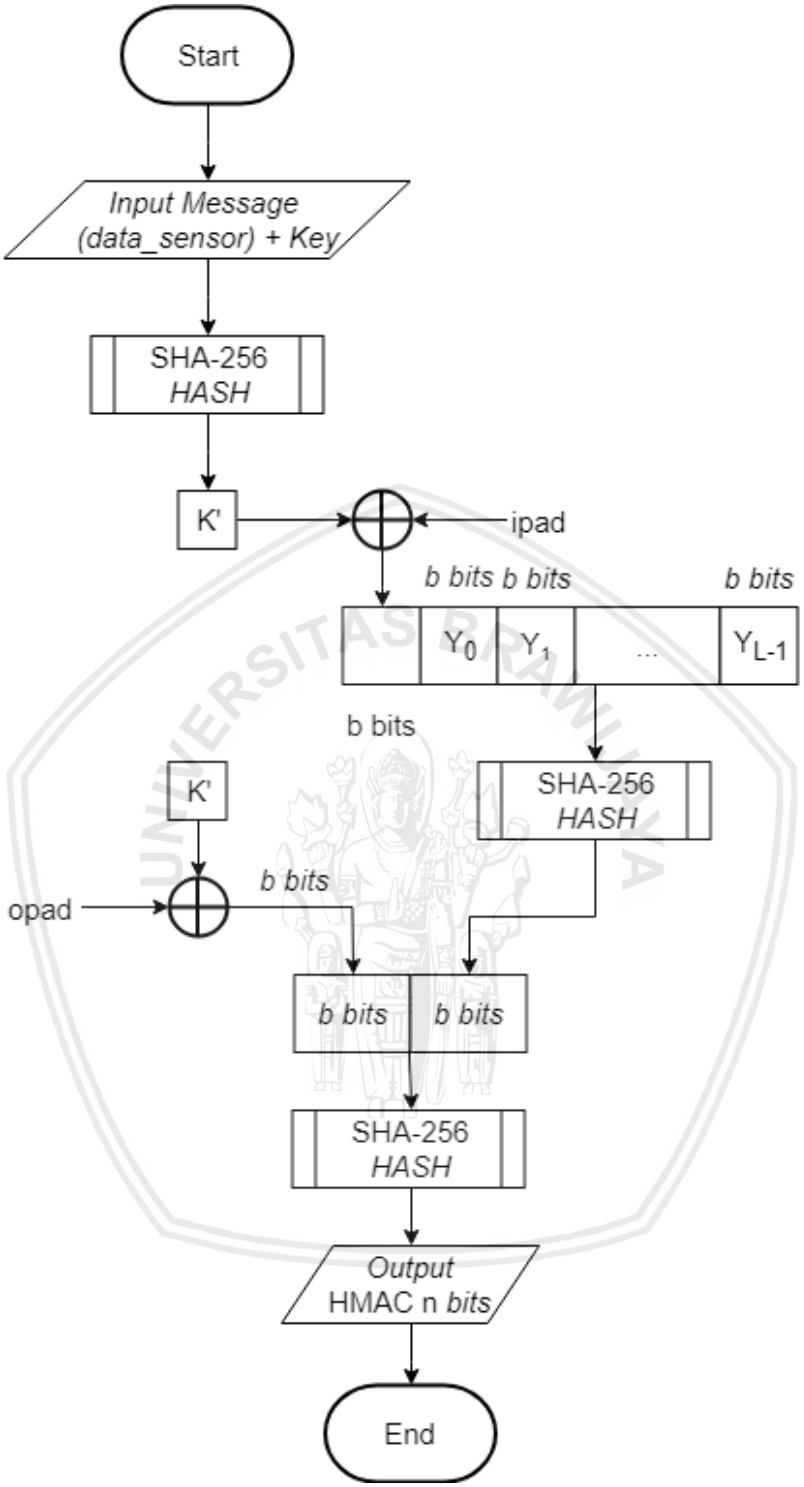
Fungsi *callback subscriber* mengidentifikasi pesan yang diterima berdasarkan *topic*-nya kemudian mengubah *payload* JSON dengan JSON *unmarshaling* agar didapat MAC dan data sensor dari *publisher*. Data sensor tersebut kemudian digunakan sebagai masukan untuk membuat MAC dengan *key* yang sama dengan yang digunakan oleh *publisher*. Setelah MAC dihasilkan oleh *subscriber*, kemudian membandingkan MAC tersebut dengan MAC dari *publisher*. Jika kedua MAC sama, maka data sensor akan ditampilkan, jika MAC *subcriber* tidak sama dengan MAC *publisher* maka akan mencetak kata data diubah.

5.1.2 Perancangan Algoritme SHA-256

Perancangan algoritme menjelaskan tentang alur atau proses pembuatan MAC algoritme SHA-256 yang terjadi pada *publisher* dan *subscriber* . Dalam SHA-2 terdapat beberapa parameter umum yang digunakan. Parameter tersebut ditunjukkan pada Tabel 5.4 dan alur pembuatan MAC algoritme SHA-256 ditunjukkan pada Gambar 5.4.

Tabel 5.4 Parameter SHA-2

No	Parameter	Keterangan
1	$a, b, c, \ldots, h$	variabel kerja w-bit yang digunakan dalam perhitungan nilai *hash*, $H^{(i)}$.
2	$H^{(i)}$	Nilai $i^{(th)}$ *hash*. $H^{(0)}$ adalah nilai *hash* awal; $H^{(N)}$adalah nilai *hash* akhir dan digunakan untuk menentukan pesan yang di *hash*.
3	$H_j^{(i)}$	$j^{(th)}$ adalah nilai dari *hash* $i^{(th)}$, di mana $H_0^{(i)}$ adalah *word hash* paling kiri adalah nilai i.
4	K_t	Nilai konstan yang akan digunakan untuk iterasi t dari perhitungan hash.
5	k	Jumlah nol ditambahkan ke pesan selama langkah padding.
6	ℓ	Panjang pesan, M, dalam bit.
7	m	Jumlah bit dalam blok pesan, $M^{(i)}$.
8	M	Pesan yang harus di *hash.*
9	$M^{(i)}$	Blok pesan i, dengan ukuran m bit.
10	$M_j^{(i)}$	Kata di $j^{(th)}$ dari blok pesan $i^{(th)}$ di mana $M_0^{(i)}$ adalah kata paling kiri blok pesan i.
11	n	Jumlah bit yang akan diputar atau digeser ketika suatu kata dioperasikan.
12	N	Jumlah blok yang mengisi pesan.
13	T	Kata di w-bit sementara yang digunakan dalam komputasi hash.
14	w	Jumlah bit dalam satu kata.
15	W_t	t^{th} kata di w-bit dari jadwal pesan.



Gambar 5.4 Perancangan HMAC SHA-256

Alur pembuatan HMAC SHA-256 dari memasukkan *input* berupa pesan dan *key* sampai menjadi MAC dari hasil pesan dan *key* yang sudah di *input*-kan dijelaskan pada Tabel 5.5.

Tabel 5.5 Proses pembuatan HMAC SHA-256

Langkah	Keterangan
1	Memasukkan *input message* yang berupa data sensor.
2	Menghitung K' menggunakan SHA-256 .
3	Menghitung *hash* untuk K'.
4	Jika panjang K' = B, mengatur K0 = K'. Lanjut ke langkah 7.
5	Jika panjang K'>B, *hash* K' untuk memperoleh sebuah *string* L *byte.* (K'=H(K')).
6	Jika panjang K'<B, tambahkan nol pada akhir K' untuk membentuk *string* B *byte* K'0.
7	Melakukan XOR antara K'0 dengan *ipad* untuk memperoleh *string byte* berukuran B. (K'0 ⊕ *ipad*).
8	Menambahkan *string m* ke dalam hasil *string* dari langkah 7 sebelumnya. ((K'0 ⊕ *ipad*)‖ *m*).
9	Menerapakan H untuk *string* yang diperoleh dari langkah 8. (H(K'0 ⊕ *ipad*)‖ *m*).
10	Melakukan XOR K'0 dengan *opad.* (K'0 ⊕ *opad*).
11	Menyisipkan hasil dari langkah 9 ke langkah 10. ((K'0 ⊕ *opad*) ‖ H(K'0 ⊕ *ipad*)‖ *m*)
12	Menerapkan H dari langkah 11. (H(K'0 ⊕ *opad*) ‖ H(K'0 ⊕ *ipad*)‖ *m*).
13	Pilih *n byte* paling kiri dari hasil langkah 12 sebagai hasil MAC.

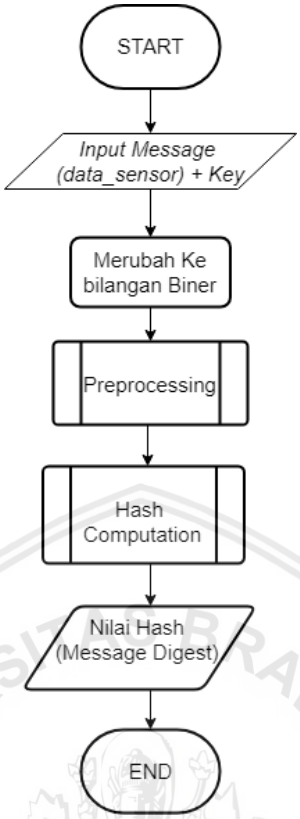
Keterangan Simbol dan rumus dari HMAC :

B = *Blocksize*

$$\text{HMAC}(K,m) = H((K' \oplus opad) \| (H((K' \oplus ipad) \| m)) \quad (5.1)$$

H = Fungsi *hash.*
K = Kunci Rahasia.
m = Pesan yang akan di *hash.*
K' = Turunan dari K dengan ditambahkan angka 0 ke arah kanan pada blok inputan dari fungsi *hash* atau hasil *hash* dari K apabila K lebih panjang dari fungsi *hash*.
‖ = Penggabungan / penyambungan
⊕ = Operasi Bitwise XOR ("eksklusif-OR").
Opad = *Outer Padding* (0x5c5c…5c panjang satu blok konstan heksadesimal)
Ipad = *Inner Padding* (0x3636…36 panjang satu blok konstan heksadesimal)

Pada Gambar 5.5 menjelaskan alur proses *hashing* yang terjadi pada algoritme SHA-2 yang tidak bisa dijelsakan secara lansung. Oleh karena itu dijelaskan pada Gambar 5.6 dan Gambar 5.7.

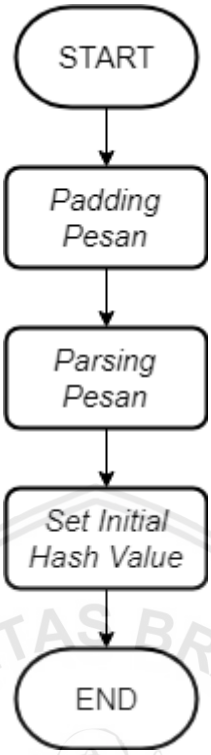


Gambar 5.5 Perancangan Algoritme SHA-256

Proses hashing menggunakan algoritme SHA-256 yang dimulai dari memasukkan *input* berupa pesan dan key yang dijadikan *message digest.*

Tabel 5.6 Proses pembuatan Algoritme SHA-256

Langkah	Keteranagan
1	Memasukkan *input message* yang berupa data sensor.
2	Mengubah *input message* menjadi *biner*
3	Melakukan *preprocessing*
4	Melakukan *Hash Computation*
5	Hasil *output* berupa hasil *message digest*

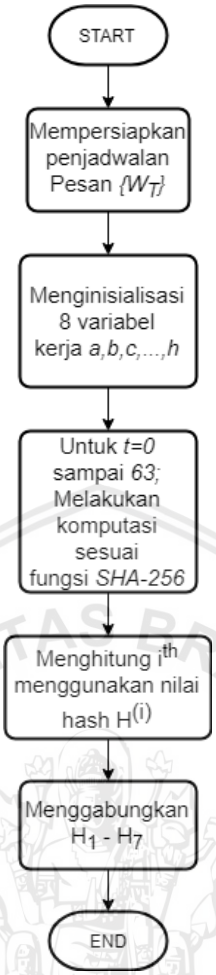


Gambar 5.6 Perancangan *Preprocessing* Algoritme SHA-256

Proses yang terjadi di dalam algoritme SHA-2 pada saat preprocessing yang menghasilkan inisialisai inisialisasi H^0, H^1, H^2,..., H^7.

Tabel 5.7 Perancangan *Preprocessing* Algoritme SHA-256

Langkah	Keteranagan
1	Pesan berupa biner disisipkan angka 1 dan ditambahkan bit 0 hingga panjang pesan kongruen dengan 448 modulo 512. Panjang pesan yang asli kemudian ditambahkan sebagai angka biner 64 bit maka panjang pesan sekarang menjadi kelipatan 512 bit
2	Pesan yang sudah di *padding* tadi kemudian dibagi menjadi N buah blok, dimana setiap blok memiliki panjang 512 bit
3	Melakukan inisialisasi H^0, H^1, H^2,..., H^7



Gambar 5.7 Perancangan *Hash Computation* Algoritme SHA-256

Penjelasan proses penggabungan nilai H^1- H^7 yang terjadi pada proses *hash computation algoritme* SHA-2 ditunjukkan pada Tabel 5.8.

Tabel 5.8 Perancangan *Hash Computation* Algoritme SHA-256

Langkah	Keteranagan
1	Mempersiapkan penjadwalan pesan
2	Inisialisasi delapan variabel kerja, a, b, c, d, e, f, g, dan h
3	Melakukan perhitungan sebanyak 64 kali putaran unutuk setiap perhitungan blok. Delapan variabel yang diberi label a, b, c, d, e, f, g, dan h yang nilainya terus berganti selama perputaran sebanyak 64 kali
4	Melakukan putaran sebanyak 64 kali, nilai *hash* $H^{(i)}$ kemudian menghitung nilai H^0, H^1, H^2,..., H^7
5	Menggabungkan delapan variabel yang sudah dikompuitasi

5.1.3 Perancangan Pengujian

Perancangan pengujian merupakan penGambaran tentang parameter dan scenario yang digunakan untuk menguji algoritme dan sistem yang diterapkan. Pengujian dibagi menjadi 3 yaitu pengujian algoritme, pengujian fungsional , dan pengujian performa sistem.

5.1.3.1 Pengujian Validitas Algoritme SHA-256

Pengujian validitas algoritme berfungsi untuk mengetahui validitas algoritme SHA-2 yang diterapkan. Pengujian algoritme dilakukan dengan cara *test vector* yang telah disediakan oleh pembuat algoritme SHA-2. Pengujian *test vector* adalah pengujian dengan membandingakan hasil *output* dari kode program algoritme SHA-2 yang diterapkan dengan data *test vector.* Berdasarkan *test vector* yang disediakan oleh pembuat algoritme SHA-2, pengujian dilakukan 6 kali dengan panjang *output* 256 *bits* serta *key* dan data yang dijadikan sebagai *input* dapat dilihat pada Tabel 5.9.

Tabel 5.9 *key* dan Data *test vector*

No	Key	Data
1	0b0b0b0b0b0b…0b	4869205468657265
2	4a656665	7768617420646f207961207761e7420666f72206e 6f7468696e673f
3	aaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaa aaaaaaaa	dddddddddddddddddddddddddddddddddd… ddd
4	aaaaaaaaaaaaa...aa a	54657374205573696e67204c6172676572205468… 374
5	aaaaaaaaaaaaa...aa a	5468697320697320612074657374207573696e67 …d2e
6	010203040506…81 9	Cdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcd… cdcd
7	0c0c0c0c0c…0c0c0 c	546573742057697468205472756e636174696f6e

5.1.3.2 Pengujian Waktu Esekusi Algoritme SHA-256

Pengujian Waktu Esekusi algoritme SHA-256 berguna untuk mengetahui lama waktu yang dibutukan setiap pengambilan data dari sensor yang dirubah menjadi MAC. Cara pengujian ini di uji pada *publisher* dan *subscriber* sebanyak 30 kali dan diulang 10 kali , waktu esekusi di hitung saat data dari 3 sensor di terima oleh algorima sebagai *input* dan berupa kode MAC yang di hasilkan dari data 3 sensor sebagai *output*.

5.1.3.3 Pengujian Fungsional Sistem

Pengujian fungisonal sistem berisikan scenario yang digunakan untuk memastikan sistem yang dibuat berjalan sesuai yang di inginkan atau tidak. Keberhasilan pengujian fungsional sistem diperoleh dari keluaran sistem saat di jalankan. Skenario melakukan pengujian fungsional sistem dijelaskan pada Tabel 5.10.

Tabel 5.10 Skenario Perancangan Pengujian Fungsional

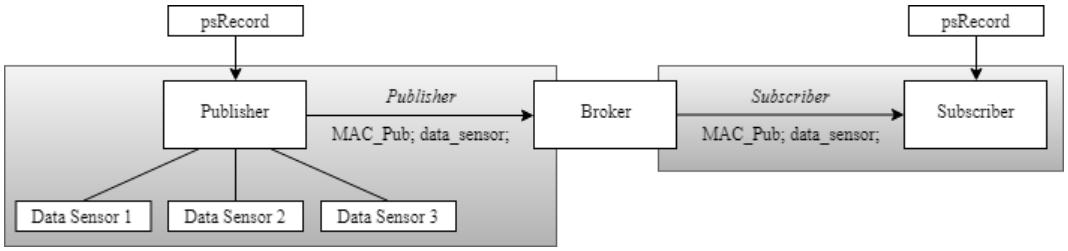
Kode	*Test Case*	Keterangan
P_01	Pengujian pembuatan MAC oleh algoritme SHA-256.	Memastikan *publisher* dan *subscriber* menghasilkan MAC algoritme SHA-256 sesuai *key* dan data sensor yang didapatkan.
P_02	Pengujian pengambilan data dari masing-masing sensor.	Memastikan Raspberry Pi 3 yang digunakan sebagai *publisher* dapat membaca data dari 3 sensor ang digunakan.
P_03	Pengujian pengambilan data sensor *publisher.*	Memastikan *publisher* dapat mengambil dan mengirimkan data dari 3 sensor yang digunakan.
P_04	Pengujian MQTT *Publish.*	Memastikan *publisher* dapat melakukan *publish* data dengan *payload* JSON yang berisi MAC SHA-256 dan data sensor.
P_05	Pengujian MQTT *Subscribe*.	Memastikan *subscriber* dapat menerima data dari *publisher* dan membaca *payload* JSON yang berisi MAC SHA-256 dan data sensor.
P_06	Pengujian pengecekan integritas data sensor oleh *subscriber*.	Memastikan *subsciber* dapat menghasilkan MAC dari data sensor yang dikirim oleh *publisher* dan membandingkan MAC yang dihasilkan oleh *subscriber* dengan MAC yang dihasilkan oleh *publisher.*

5.1.3.4 Pengujian Performa Sistem

Perancangan pengujian performa sistem berfungsi sebagai tanda bahwa sistem yang diterapkan sudah sesuai dan berjalan dengan baik. Pengujian performa sistem meliputi cara kerja pemrosesan algoritme dalam penggunaan *memory* yang digunakan, untuk mengetahui penggunaan *memory* yang digunakan pada pegujian ini menggunakan aplikasi psRecord saat melakukan *publish* dan *subscribe*. Setelah itu untuk pengujian waktu eseksusi algoritme SHA-256 menguji waktu yang di butuhkan algoritme dan sistem dalam satu kali proses menggunakan modul python *time*.

Perancangan pengujian performa sistem untuk menguji *memory* yaitu dengan cara mengitung kedua parameter pengujian performa sistem yaitu saat

sistem menggunkan intergritas data dan tanpa integritas data. Perancangan pengujian performa dapat dilihat pada Gambar 5.8.



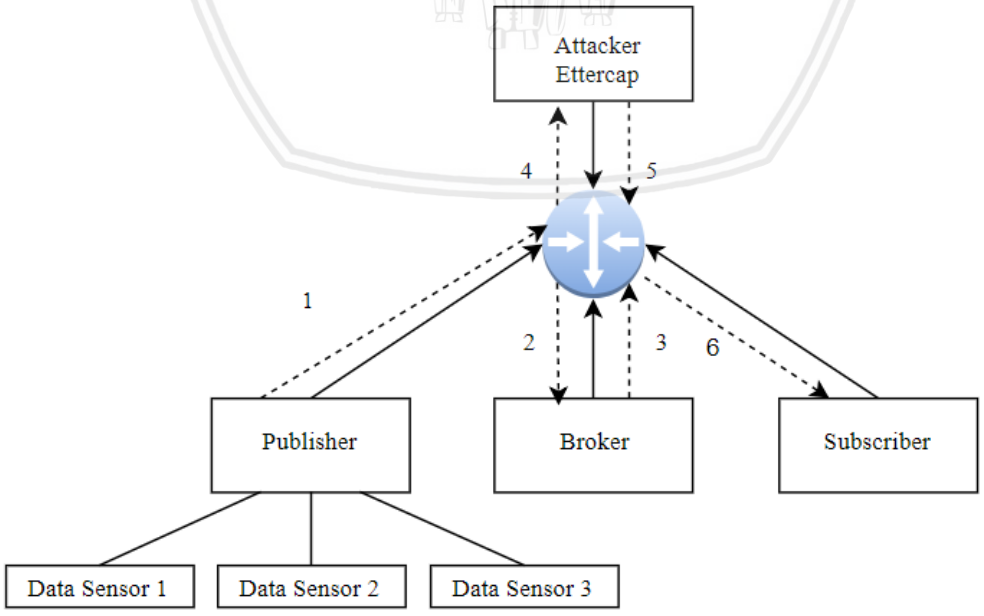
Gambar 5.8 Perancangan Pengujian Performa

Di dalam Gambar 5.8 menggambarkan pengujian untuk mengukur *memory* menggunakan psRecord pada *publisher* dan *subscriber*, di dalam pengujian performa ini di lakukan sebanyak 30 kali dan diulang sebanyak 10 kali untuk mendapatkan hasil yang akurat.

5.1.3.5 Pengujian Keamanan Sistem

Pengujian keamanan sistem menjelaskan tentang pengecekan integritas data pada protokol MQTT sudah sesuai dengan masalah yang dihadapi di dalam penelitian. Di dalam menguji keamanan ini nilai data sensor yang di *publish* akan dirubah, pengujian ini dilakukan untuk memastikan *subscriber* dapat mengetahui data sensor yang benar dan data sensor yang sudah diubah.

Pengujian keamanan sistem ini menggunakan aplikasi Ettercap, di dalam aplikasi Ettercap ini menggunakan filter agar dapat mengubah *payload* MQTT sebelum *publisher* mengirim pesan ke *subscriber.* Cara menggunakan Ettercap adalah dengan menggunakan filter Etterfiltter yang digunkan untuk filter di Ettercap. Perancangan pengujian keamanan dapat dilihat di Gambar 5.9.



Gambar 5.9 Perancangan Pengujian Keamanan

Pada Gambar 5.9 dapat diliaht alur pengujian keamanan dengan menggunakan perangakt lunak Ettercap pada sebuah host yang terpisah di dalam mesin Virtual Ubuntu Server.

5.2 Implementasi

Pada bagian implementasi berisi tentang pembacaan data dari 3 sensor yang digunakan sebagai *inputan* data, implementasi algoritme SHA-2 sebagai integritas data dari sensor yang digunakan, dengan menganalisa dari perancangan sistem implementasi dibagi menjadi 2 bagian yaitu implementasi sistem dan implementasi algoritme SHA-2. Implentasi sistem pada *publisher* menggunakan Raspbian OS, sedangkan *broker* dan *subscriber* menggunakan *guest* Ubuntu server.

5.2.1 Implementasi Sistem

Implementasi pada setiap sistem berisi tentang cara instalasi, konfigurasi, dan penjelasan dari potongan kode program yang dibutuhkan agar sistem dapat berjalan sesuai yang diharapkan, di dalam arsitektur *publish – subscribe* menggunkan protokol MQTT untuk bertukar pesan. Jenis protkol MQTT yang digunakan oleh *broker* yaitu Eclipse MQTT, serta pengunaan Bahasa pemrograman *python* sebagai Bahasa pemrograman yang di implementasikan pada *publisher* dan *subscriber*. Penjelasan tentang implementasi ditunjukkan pada bab implementasi *publisher*, implementasi *broker*, dan implementasi *subscriber*.

5.2.1.1 Implementasi *Publisher*

Publisher adalah *client* dari protocol MQTT yang bertugas mengirimkan pesan dari pembacaan 3 sensor yang digunakan, untuk mengimplementasikan MQTT *client* menggunakan bahasa pemrograman *python* membutuhkan *library* Paho MQTT selain itu juga membutuhkan *install python* untuk menjalankan *library* Paho MQTT. Berikut ini adalah cara yang digunakan untuk proses instalasi *python* dan *library* Paho MQTT yang dijelaskan pada Tabel 5.11.

Tabel 5.11 Proses instalasi *python* dan *library* Paho MQTT pada *Publisher*

No	Perintah
1	`$ sudo apt install python python pip && pip install paho-mqtt`
2	`$ git clone https://github.com/eclipse/paho.mqtt.python`
3	`cd paho.mqtt.python`
4	`python setup.py install`

1. Baris 1 : perintah untuk menginstal *Python Package Index* (PyPi) dan Paho MQTT
2. Baris 2 : perintah untuk mendapatkan kode lengkap dari *Python Package Index* (PyPi). Setelah kode program Paho MQTT berhasil di unduh.
3. Baris 3-4 : perintah memindah *directory* untuk memulai kompilasi.

Hasil Paho MQTT tersebut digunakan sebagai *library* yang terdapat pada file *publisher1.py.* Proses yang berada pada *publisher* dijelaskan pada Tabel 5.12.

Tabel 5.12 Kode Program *Publisher*

No	*Source Code*
1	`import json`
2	`import paho.mqtt.client as mqtt`
3	`import sha2`
4	`import time`
5	`import os`
6	`import glob`
7	`import RPi.GPIO as GPIO`
8	`GPIO.setwarnings(False)`
9	
10	`pin_to_circuit = 11`
11	
12	`# Set Pembacaan Board Raspberri pi sebagai Board Mode`
13	`GPIO.setmode(GPIO.BOARD))`
14	
15	`...`
16	`# Inisialisasi object mqtt`
17	`mqttc = mqtt.Client("pub1", clean_session=True)`
18	
19	`# Buat koneksi ke broker`
20	`mqttc.connect( "192.168.201.100", 1883 )`
21	
22	`while True:`
23	`suhu = read_temp()`
24	`dist = "%.3f" % distance()`
25	`kekeruhan = rc_time(pin_to_circuit)`
26	
27	`sensor_ketinggian = str(dist)`
28	`sensor_suhu = str(suhu)`
29	`sensor_kekeruhan = str(kekeruhan)`
30	
31	`#string sensor 1 + sensor 2 + sensor 3`
32	`group_data_sensor = sensor_kekeruhan + sensor_suhu +`
33	`sensor_ketinggian`
34	`mac_pub = sha2.hmac_sha256("key_publisher",`
35	`group_data_sensor)`
36	
37	`#json encoding`
38	`array_data_sensor = [sensor_ketinggian, sensor_suhu,`
39	`sensor_kekeruhan]`
40	`dictionary_pub = {"mac" : str(mac_pub), "sensor" :`
41	`array_data_sensor}`
42	`json_pub = json.dumps(dictionary_pub)`
43	
44	`#mqtt publish`
45	`mqttc.publish("/node/Sensor", payload=json_pub)`
46	`print("Published")`
47	`time.sleep(1)`
48	

1. Baris 1-7 : Memuat modul – modul yang diperlukan.
2. Baris 8 : Digunakan untuk mengabaikan peringatan.
3. Baris 10 : Deklarasi varibel `pin_to_circuit` sama dengan 11.
4. Baris 13 : Set GPIO sebagai `GPIO.BOARD`.
5. Baris 17 : Variabel `mqttc` berisi `mqtt.client` bernama `pub1,clean_session=True`.
6. Baris 20 : Membuat koneksi ke broker menggunakan IP `192.168.201.100` dan port 1883.

7. Baris 22 : Perulangan while dengan parameter TRUE, yang akan mengeksekusi kode program tanpa kondisi.
8. Baris 23 : Suhu berisi hasil dari pemanggilan method `read_temp`.
9. Baris 24 : Pemotongan tiga angka di belakang koma hasil pemanggilan method `distance`.
10. Baris 25 : Kekeruhan berisi hasil dari pemanggilan method `rc_time` dimana `pin_to_circuit` bernilai 11.
11. Baris 27-29 : Merubah data integer menjadi tipe data string.
12. Baris 33-34 : Variabel `group_data_sensor` berisi penjumlahan. `sensor_kekeruhan + sensor_suhu + sensor_ketinggian`.
13. Baris 35-36 : Variabel `mac_pub` berisi hasil perhitungan algoritme SHA-2 dengan *key publisher* dengan nilai data pada variable `group_data_sensor`.
14. Baris 39-40 : Variabel `array_data_sensor` merupakan tipe data array yang berisikan `sensor_ketinggian, sensor_suhu, sensor_kekeruhan`.
15. Baris 41-42 : Variabel `dictionary_pub` dengan tipe data dictionary berisi pasangan *key value* mac dan hasil perhitungan variable `array_data_sensor`.
16. Baris 43 : Variabel json_pub berisi data dari Variabel dictionary_pub yang di ubah menjadi format data string.
17. Baris 46 : Mengirimkan data yang dibuat dengan topik tertentu.
18. Baris 47 : Mencetak informasi Published menandakan bahwa data berhasil di *publish.*
19. Baris 48 : Delay 1 seconds berfungsi untuk membuat waktu jeda antara *publish* data pertama dan seterusnya.

a. Sensor Ultrasonik HC-SR04

Berikut ini adalah potongan dari file *publish1.py* yang digunakan untuk membaca data dari sensor yang digunakan yaitu sensor Ultrasonik HC-SR04 proses dari pembacaan sensor dijelaskan pada Tabel 5.13.

Tabel 5.13 Kode Pembecaan Sensor Ultrasonik HC-SR04

No	*Source Code*
1	...
2	# Inisialisasi Sensor Ultrasonic
3	GPIO_TRIGGER = 8
4	GPIO_ECHO = 10
5	
6	GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
7	GPIO.setup(GPIO_ECHO, GPIO.IN)
8	...
9	# Fungsi Sensor Ultrasinic
10	def distance():
11	GPIO.output(GPIO_TRIGGER, True)
12	
13	time.sleep(0.00001)
14	GPIO.output(GPIO_TRIGGER, False)
15	
16	StartTime = time.time()
17	StopTime = time.time()
18	
19	while GPIO.input(GPIO_ECHO) == 0:

```
20          StartTime = time.time()
21
22      while GPIO.input(GPIO_ECHO) == 1:
23          StopTime = time.time()
24
25      TimeElapsed = StopTime - StartTime
26      distance = (TimeElapsed * 34300) / 2
27
28      return distance
```

1. Baris 3 : Deklarasi variabel `GPIO_TRIGGER` bernilai 8, berfungsi set `GPIO_TRIGGER` ke pin 8.
2. Baris 4 : Deklarasi variabel `GPIO_ECHO` bernilai 10, berfungsi set `GPIO_ECHO` ke pin 10.
3. Baris 6 : Setup GPIO 8 sebagai `GPIO.OUT`.
4. Baris 7 : Setup GPIO 10 sebagai `GPIO.IN`.
5. Baris 10 : Deklarasi method `distance()`, berfungsi untuk pengukuran sensor ultrasonic.
6. Baris 11 : Set `GPIO_TRIGGER` menjadi *HIGH* (mengeluarkan sinyal).
7. Baris 13 : Delay 0,00001 seconds, berfungsi untuk memberi waktu jeda dari *HIGH* ke *LOW*.
8. Baris 14 : Set `GPIO_TRIGGER` menjadi *LOW*.
9. Baris 16-17 : Varibel `StartTime` dan variabel `StopTime` yang berisi waktu berdasarkan detik.
10. Baris 19 : Cek jika ada sinyal yang diterima.
11. Baris 20 : Simpan waktu sebagai sinyal *LOW*.
12. Baris 22 : Cek jika sinyal yang diterima sudah berhenti.
13. Baris 23 : Simpan waktu menjadi sinyal *HIGH*.
14. Baris 25 : Menghitung `TimeElapsed` dengan cara `StopTime - StartTime`.
15. Baris 26 : (`TimeElapsed` / 2) * 34300, `TimeElapsed` adalah waktu `StopTime - StartTime` dibagi 2 karena harus melakukan perjalanan bolak – balik dan dikalikan 34300 yaitu kecepatan suara.
16. Baris 28 : *Return* nilai dari variabel distance.

b. Sensor Cahaya Photoresistor LDR GL5528

Berikut ini adalah potongan dari file *publish.py* yang digunakan untuk membaca data dari sensor yang digunakan yaitu sensor Cahaya Photoresistor LDR GL5528 proses dari pembacaan sensor dijelaskan pada Tabel 5.14.

Tabel 5.14 Kode Pembecaan Sensor Cahaya Photoresistor LDR GL5528

No	*Source Code*
1	...
2	# Fungsi Sensor LDR
3	def rc_time (pin_to_circuit):
4	count = 0
5	
6	GPIO.setup(pin_to_circuit, GPIO.OUT)
7	GPIO.output(pin_to_circuit, GPIO.LOW)
8	time.sleep(0.1)
9	
10	GPIO.setup(pin_to_circuit, GPIO.IN)

```
11
12      while (GPIO.input(pin_to_circuit) == GPIO.LOW):
13          count += 1
14
15      return count
```

1. Baris 3 : Deklarasi method `rc_time` dengan parameter `pin_to_circuit`, berfungsi untuk pengukuran sensor *Photoresistor LDR* dengan parameter `pin_to_circuit`.
2. Baris 4 : Counter awal sama dengan 0.
3. Baris 6 : `pin_to_circuit` sebagai `GPIO.OUT`.
4. Baris 7 : Set `pin_to_circuit` menjadi *LOW.*
5. Baris 8 : Delay 0,1 seconds, berfungsi memberikan waktu jeda `pin_to_circuit` dari *LOW* ke *HIGH*.
6. Baris 10 : `pin_to_circuit` sebagai `GPIO.IN`.
7. Baris 12-13 : Melakukan looping sampai `pin_to_circuit` menjadi *HIGH.*
8. Baris 15 : *Return* nilai dari variabel distance

c. Sensor Suhu DS18B20

Berikut ini adalah potongan dari file *publish.py* yang digunakan untuk membaca data dari sensor yang digunakan yaitu sensor Suhu DS18B2O proses dari pembacaan sensor dijelaskan pada Tabel 5.15.

Tabel 5.15 Kode Pembecaan Sensor Suhu DS18B2O

No	*Source Code*
1	...
2	#Sensor Suhu
3	suhu = 0;
4	
5	os.system('modprobe w1-gpio')
6	os.system('modprobe w1-therm')
7	
8	base_dir = '/sys/bus/w1/devices/'
9	device_folder = glob.glob(base_dir + '28*')[0]
10	device_file = device_folder + '/w1_slave'
11	...
12	# Fungsi Sensor Suhu Data Mentah/ Nilai Voltase
13	def read_temp_raw():
14	f = open(device_file, 'r')
15	lines = f.readlines()
16	f.close()
17	return lines
18	
19	# Fungsi Sensor Suhu/Pengolahan Untuk Jadi Celcius
20	def read_temp():
21	lines = read_temp_raw()
22	while lines[0].strip()[-3:] != 'YES':
23	time.sleep(0.2)
24	lines = read_temp_raw()
25	equals_pos = lines[1].find('t=')
26	if equals_pos != -1:
27	temp_string = lines[1][equals_pos+2:]
28	temp_c = float(temp_string) / 1000.0
29	return temp_c

1. Baris 3 : Suhu awal sama dengan 0.

2. Baris 5-6 : Mengaktifkan one-wire interface, berfungsi untuk komunikasi sensor hana menggunkan 1 kawat.
3. Baris 8 : Deklarasi varibel yang berisi `/sys/bus/w1/devices/`.
4. Baris 9 : Untuk menemukan file ang berada pada direktori `/sys/bus/w1/devices/28`.
5. Baris 10 : `device_file` berisi `device_folder + '/w1_slave'`.
6. Baris 13 : Deklarasi method `read_temp_raw():`, berfungsi untuk mendapatkan data mentah dari sensor.
7. Baris 14-16 : Membuka isi `device_file` dengan hak ases hanya `read`.
8. Baris 20 : Deklarasi method `read_temp()`, berfungsi untuk mengubah data mentah menjadi Celsius.
9. Baris 21 : Variabel `lines` memanggil fungsi `read_temp_raw()`.
10. Baris 22 : Perulangan selama nilai variable pada indeks ke 0 tidak berisi *YES* maka `time.sleep(0.2), lines = read_temp_raw(). time.sleep`.
11. Baris 23 : Delay 0.2 seconds.
12. Baris 24 : Variabel `lines` memanggil fungsi `read_temp_raw()`.
13. Baris 25 : Variabel `equals_pos` hasil pencarian t dengan nilai lines indeks ke 1.
14. Baris 26-29 : Jika variable `equals_pos` tidak sama dengan 1 baca data sensor yang dihasilkan.

5.2.1.2 Implementasi *Broker*

Eclipse MOSQUITTO adalah protokol MQTT yang digunakan untuk *broker* yang cocok digunakan untuk komunikasi antar mesin yang memiliki daya yang rendah. MQTT *broker* berfungsi sebagai perantara komunikasi antara client yaitu *client publisher* dan *client subscriber.* Berikut ini adalah cara yang digunakan untuk instalasi mosquitto pada Ubuntu Sever yang dijelaskan pada Tabel 5.16.

Tabel 5.16 Proses Instalasi mosquitto pada *Broker*

No	Perintah
1	`$ sudo apt install mosquitto -y`
2	`$ sudo service mosquitto restart`

1. Baris 1 : Perintah untuk instalasi mosquitto. Setelah ecplise mosqitto berhasil di *install service* mosquitto harus diulang.
2. Baris 2 : Perintah untuk mengulang *service* mosquitto.

5.2.1.3 Implementasi *Subscriber*

Subscriber adalah *client* dari protocol MQTT yang bertugas menerima pesan dari *publisher*, untuk mengimplementasikan MQTT *client* menggunakan bahasa pemrograman *python* membutuhkan *library* Paho MQTT selain itu juga membutuhkan *install python* untuk menjalankan *library* Paho MQTT. Berikut ini adalah cara yang digunakan untuk proses instalasi *python* dan *library* Paho MQTT yang ditunjukkan pada Tabel 5.17.

Tabel 5.17 Proses instalasi *python* dan *library* Paho MQTT pada *Subscriber*

No	Perintah
1	`$ sudo apt install python python pip && pip install paho-mqtt`
2	`$ git clone https://github.com/eclipse/paho.mqtt.python`
3	`cd paho.mqtt.python`
4	`python setup.py install`

1. Baris 1 : perintah untuk menginstal *Python Package Index* (PyPi) dan Paho MQTT
2. Baris 2 : perintah untuk mendapatkan kode lengkap dari *Python Package Index* (PyPi). Setelah kode program Paho MQTT berhasil di unduh.
3. Baris 3-4 : perintah memindah *directory* untuk memulai kompilasi.

Hasil Paho MQTT tersebut digunakan sebagai *library* yang terdapat pada file *subscriber1.py.* Proses yang berada pada *subscriber* dijelaskan pada Tabel 5.18.

Tabel 5.18 Kode Program *Subscriber*

No	*Source Code*
1	`import json`
2	`import sha2`
3	`# Import library paho-mqtt`
4	`import paho.mqtt.client as mqtt`
5	
6	`# Inisiasi object mqtt`
7	`mqttc = mqtt.Client("sub1", clean_session=True)`
8	
9	`# Buat koneksi ke broker`
10	`mqttc.connect( "192.168.201.100", 1883 )`
11	
12	`def on_connect(mqttc, obj, flags, rc):`
13	`  print("Connected")`
14	
15	`def on_message(mqttc, obj, msg):`
16	`  #json encoding`
17	`  data_str = json.loads(str(msg.payload.decode("utf-8")))`
18	`  mac_pub = data_str["mac"]`
19	`  data_sensor = data_str["sensor"]`
20	
21	`  #baca array dalam json`
22	`  data_sensor_ketinggian = data_sensor[0]`
23	`  data_sensor_suhu = data_sensor[1]`
24	`  data_sensor_kekeruhan = data_sensor[2]`
25	
26	`  #string sensor 1 + sensor 2 + sensor 3`
27	`  group_data_sensor = data_sensor_kekeruhan +`
28	`data_sensor_suhu + data_sensor_ketinggian`
29	`  mac_sub = sha2.hmac_sha256("key_publisher",`
30	`group_data_sensor)`
31	
32	`  if (str(mac_sub) == mac_pub) :`
33	`  print ("mac = " + str(mac_pub) + "\ndata = {" +`
34	`"kekeruhan : " + data_sensor_kekeruhan + " suhu : " +`
35	`data_sensor_suhu + " ketinggian : " + data_sensor_ketinggian +`
36	`"}\n")`
37	`  else :`
38	`  print ("data diubah")`
39	
40	`# Set callback function`
41	`mqttc.on_connect = on_connect`
42	`mqttc.on_message = on_message`
43	

```
# Subscribe ke topik tertentu
mqttc.subscribe("/node/Sensor", qos=1)

# Loop supaya subscribernya tidak berhenti
mqttc.loop_forever()
```

1. Baris1-4 : Memuat modul – modul yang diperlukan.
2. Baris 7 : Inisialisai objek mqttc sebagai mqtt client.
3. Baris 10 : Membuat koneksi ke broker dengan mengatur alamat ip dan port yang digunakan.
4. Baris 12-13: Deklarasi fungsi `on_connect` sebagai penanda jika subscriber sudah terhubung dengan broker.
5. Baris 15 : Deklarasi fungsi `on_message`.
6. Baris 17 : Variabel `data_str` yang berfungsi mengubah format data string ke format data json.
7. Baris 18 : Varibel `mac_pub` berisi hasil value `data_str` pada kunci `mac`.
8. Baris 19 : Variabel `data_sensor` berisi hasil value `data_str` pada kunci sensor.
9. Baris 21-24 : Variabel `data_sensor_ketinggian` berisi nilai `data_sensor` pad a indeks ke 0, variabel `data_sensor_suhu` berisi nilai `data_sensor` pada indeks ke 1, Variabel `data_sensor_kekeruhan` berisi nilai `data_sensor` pad a indeks ke 2.
10. Baris 27-28 : Variabel `group_data_sensor` berisi penjumlahan `sensor_kekeruhan + sensor_suhu + sensor_ketinggian`.
11. Baris 29-30 : Variabel `mac_pub` berisi hasil perhitungan algoritme SHA-2 dengan *key publisher* dengan nilai data pada variabel `group_data_sensor`.
12. Baris 32-38 : Jika `mac_sub = mac_pub` maka akan mencetak `mac pub, data_sensor_kekeruhan,data_sensor_suhu, data_sensor_ketinggian` , jika `mac_sub` tidak sama dengan `mac_pub` maka akan mencetak kata `data diubah`.
13. Baris 41-42 : Inisialisasi unutk memanggil fungsi dengan variable.
14. Baris 45 : Untuk *subscribe* topik yang tersedia.
15. Baris 48 : Perulangan agar *subscriber* selalu menyala.

5.2.2 Implementasi Algoritme SHA-256

Algoritme di dalam implementasi ini berfungsi sebagai MAC yang di dapatkan dari hasil *hashing key* dan data yang di peroleh dari 3 sensor yang di jumlahkan. Implementasi di pengujian sistem menggunakan algoritme SHA-2 menggunakan 256 *bits* dijelaskan pada Tabel 5.19.

Tabel 5.19 Program SHA-256

No	*Source Code*
1	`from functools import reduce`
2	`from math import log,ceil`
3	
4	`def intToList2(number,length):`
5	`    return [(number >> i) & 0xff`
6	`            for i in reversed(range(0,length*8,8))]`
7	

```
8   def intToList(number):
9       L1 = log(number,256)
10      L2 = ceil(L1)
11      if L1 == L2:
12          L2 += 1
13      return [(number&(0xff<<8*i))>>8*i for i in
14  reversed(range(L2))]
15
16  def listToInt(lst):
17      return reduce(lambda x,y:(x<<8)+y,lst)
18
19  def bitList32ToList4(lst):
20      def bitListToInt(lst):
21          return reduce(lambda x,y:(x<<1)+y,lst)
22
23      lst2 = []
24      for i in range(0,len(lst),8):
25          lst2.append(bitListToInt(lst[i:i+8]))
26      return list([0]*(4-len(lst2)))+lst2
27
28  def list4ToBitList32(lst):
29      def intToBitList2(number,length):
30          return [(number>>n) & 1
31                  for n in reversed(range(length))]
32
33      lst2 = []
34      for e in lst:
35          lst2 += intToBitList2(e,8)
36      return list([0]*(32-len(lst2)))+lst2
37
38  def add32(p,q,r=None,s=None,t=None):
39      mask32 = (1<<32)-1
40      p2,q2 = listToInt(p), listToInt(q)
41      if t is None:
42          if s is None:
43              if r is None:
44                  return intToList2((p2+q2)&mask32,4)
45              else:
46                  r2 = listToInt(r)
47                  return intToList2((p2+q2+r2)&mask32,4)
48          else:
49              r2,s2 = listToInt(r),listToInt(s)
50              return intToList2((p2+q2+r2+s2)&mask32,4)
51      else:
52          r2,s2,t2 = listToInt(r),listToInt(s),listToInt(t)
53          return intToList2((p2+q2+r2+s2+t2)&mask32,4)
54
55  def xor(x,y,z=None):
56      if z is None:
57          return list(i^j for i,j in zip(x,y))
58      else:
59          return list(i^j^k for i,j,k in zip(x,y,z))
60
61  def sha256(m):
62      def padding(m):
63          def bitListToList(lst):
64              lst2 = [0]*((8-len(lst)%8)%8)+lst
65              return [reduce(lambda x,y:(x<<1)+y,lst2[i*8:i*8+8])
66                      for i in range(len(lst2)//8)]
67
68          def intToBitList(number):
69              return list(map(int,list(bin(number)[2:])))
70
71          if type(m) is int:
72              m1 = intToBitList(m)
```

```
73              L = len(m1)
74              k = (447-L)%512
75              return
76  bitListToList(m1+[1]+list([0]*k))+intToList2(L,8)
77          else:
78              m1 = m
79              if type(m) is str:
80                  m1 = list(map(ord,m))
81              if not(type(m) is list):
82                  raise TypeError
83              L = len(m1)*8
84              k = (447-L)%512
85              return
86  m1+bitListToList([1]+list([0]*k))+intToList2(L,8)
87  . . .
88          nonlocal H
89          [a,b,c,d,e,f,g,h] = H
90          K = [0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
91               0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
92               0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
93               0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
94               0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
95               0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
96               0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
97               0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
98               0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
99               0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
100              0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
101              0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
102              0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
103              0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
104              0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
105              0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2]
106 . . .
107     H = list(map(lambda x:intToList2(x,4),H0))
108     mp = padding(m)
109     for i in range(0,len(mp),64):
110         compress(mp[i:i+64])
111     return listToInt([s2 for s1 in H for s2 in s1])
112 . . .
```

1. Baris 1 -2 : Memuat modul – modul yang diperlukan.
2. Baris 5-6 : Mengubah angka menjadi daftar byte dengan panjang yang ditentukan. Baris 9-14 : Mengubah dari setiap panjang bilangan bulat menjadi daftar bilangan bulat.
3. Baris 17 : Mengubah daftar byte menjadi angka.
4. Baris 20-21 : Mengubah daftar 32-bit menjadi 4-byte.
5. Baris 29-36 : Mengubah daftar 4-byte ke dalam 32-bit.
6. Baris 39-53 : Menambahkan hingga lima nomor ke 32-bit.
7. Baris 56-59 : Mengevaluasi XOR dari dua atau tiga *operand*.
8. Baris 62-86 : Merupakan proses *message padding* Pesan berupa biner disisipkan angka 1 dan ditambahkan bit 0 hingga panjang pesan kongruen dengan 448 modulo 512. Panjang pesan yang asli kemudian ditambahkan sebagai angka biner 64 bit maka panjang pesan sekarang menjadi kelipatan 512 bit.
9. Baris 90-105 : 32 bit pertama dari bagian pecahan *cube root* dari 64 bilangan prima pertama 2..311.

Tabel 5.20 *Initial Hash Value*

No	*Source Code*
1	`H0 = [0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,`
2	`0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19]`

Inisialisasi awal nilai inisialisasi H^0, H^1, H^2,..., H^7 dengan masukan berupa 8 bit hex.

Tabel 5.21 *Prepare message schedule* SHA-256

No	*Source Code*
1	`W = [None]*64`
2	`for t in range(16):`
3	`W[t] = m[t*4:t*4+4]`
4	`for t in range(16,64):`
5	`W[t] = add32(sigma1(W[t-2]),W[t-7],sigma0(W[t-`
6	`15]),W[t-16])`

Untuk mencari nilai W_t dengan menggabungkan nilai W_t dari *range t* sampai 16 dan panjang nilai W_t dari range 16 sampai 64 yang memamnggil fungsi `add32` yang dikalikan fungsi `sigma` dari operasi *compress* dengan rumus `sigma1(W[t2]),W[t-7],sigma0(W[t-15]),W[t-16]`.

Tabel 5.22 *Intermediate Hash Computation*

No	*Source Code*
1	`for t in range(64):`
2	`T1 =`
3	`add32(h,Sigma1(e),Ch(e,f,g),intToList2(K[t],4),W[t])`
4	`T2 = add32(Sigma0(a),Maj(a,b,c))`
5	`h = g; g = f; f = e; e = add32(d,T1)`
6	`d = c; c = b; b = a; a = add32(T1,T2)`
7	`H = [add32(x,y) for x,y in`

Melakukan perhitungan sebanyak 64 kali putaran unutuk setiap perhitungan blok. Delapan variabel yang diberi label *a, b, c, d, e, f, g, h* yang nilainya terus berganti selama perputaran sebanyak 64 kali.

Tabel 5.23 Fungsi *Compress* SHA-256

No	*Source Code*
1	`def compress(m):`
2	`def Ch(x,y,z):`
3	`return list([(i&j)^((i^0xff)&k) for i,j,k in`
4	`zip(x,y,z)])`
5	
6	`def Maj(x,y,z):`
7	`return list([(i&j)^(i&k)^(j&k) for i,j,k in`
8	`zip(x,y,z)])`
9	
10	`def rotRight(p,n):`
11	`p2 = list4ToBitList32(p)`
12	`return bitList32ToList4(p2[-n:]+p2[:-n])`
13	
14	`def shiftRight(p,n):`
15	`p2 = list4ToBitList32(p)`
16	`return bitList32ToList4(list(bytes(n))+p2[:-n])`
17	
18	`def Sigma0(p):`

```
19              return
20  xor(rotRight(p,2),rotRight(p,13),rotRight(p,22))
21
22          def Sigma1(p):
23              return
24  xor(rotRight(p,6),rotRight(p,11),rotRight(p,25))
25
26          def sigma0(p):
27              return
28  xor(rotRight(p,7),rotRight(p,18),shiftRight(p,3))
29
30          def sigma1(p):
31              return
32  xor(rotRight(p,17),rotRight(p,19),shiftRight(p,10))
```

1. Baris 2-3 : Untuk mencari nilai method Ch dengan parameter (x,y,z) menggunakan rumus $(x \wedge y) \oplus (\neg x \wedge z)$.
2. Baris 6-8 : Untuk mencari nilai method Maj dengan parameter (x,y,z) menggunakan rumus $(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$.
3. Baris 10 -16 : Mendefinisikan isi method `rotRight` dan `shiftRight` yang digunakan untuk menghitung method `sigma`.
4. Baris 18 – 32 : Mendefinisikan method `sigma` yang memanggil method `rotRight` dan `shiftRight`.

Tabel 5.24 Compute the i^{th} intermediate hash value $H^{(i)}$

No	*Source Code*
1	`H = [add32(x,y) for x,y in zip([a,b,c,d,e,f,g,h],H)]`

Menggabungkan hasil H^0, H^1, H^2,..., H^7 dengan 8 variabel kerja w-bit a, b, c, ..., h. yang memanggil method `add32` dengan parameter `(x,y)`.

Tabel 5.25 HMAC-SHA256

No	*Source Code*
1	`def hmac_sha256(k,m):`
2	`    opad = list([0x5c]*64);ipad = list([0x36]*64)`
3	`    if type(k) is int:`
4	`        k1 = intToList(k)`
5	`        L = len(k1)`
6	`        if L > 64:`
7	`            K = intToList2(sha256(k),32)+list([0]*32)`
8	`        else:`
9	`            K = k1+list([0]*(64-L))`
10	`    else:`
11	`        k1 = list(map(ord,k))`
12	`        L = len(k1)`
13	`        if L > 64:`
14	`            K = intToList(sha256(k1))`
15	`        else:`
16	`            K = k1+list([0]*(64-L))`
17	`    if type(m) is int:`
18	`        M = intToList(m)`
19	`    else:`
20	`        M = list(map(ord,m))`
21	`    arg1 = xor(K,opad)`
22	`    arg2 = xor(K,ipad)`
23	`    return sha256(arg1+intToList(sha256(arg2+M)))`

1. Baris 2 : Insialisasi variabel opad dan ipad dengan opad sepanjang `0x5c` yang dikalikan 64 dan ipad `0x36`.
2. Baris 3-20 : Untuk menentukkan panjang dari nilai *key* yang dimasukkan menjadi bilangan biner dengan pemanggilan method `intToList` dan `intToList2`.
3. Baris 21-23 : Menggabungakan hasil *key* dengan opad yang digabungakan lagi dengan hasil *key* dengan ipad lalu ditambah *message* dari hasil *hash* SHA-256 yang menghasilkan MAC SHA-256 dengan menggunakan metode HMAC.

BAB 6 PENGUJIAN DAN ANALISIS

Pengujian dan analisis membahas tentang hasil pengujian terhadap algoritme dan sistem yang diimplementasikan. Bagian pengujian dilakukan untuk memastikan terpenuhi atau tidaknya kebutuhan yang telah didefisinikan pada analisis kebutuhan dan perancangan sistem, serta memberikan solusi terhadap permasalahan yang menjadi latar belakang penelitian. Pengujian dilakukan sesuai dengan perancangan pengujian pada bab pengujian dan pengujian validitas dan waktu eksekusi algoritme, pengujian fungsional sistem, pengujian performa sistem, serta pengujian keamanan sistem.

6.1 Pengujian Validitas Algoritme SHA-256

Pengujian test vector digunakan untuk menguji algoritme SHA-256 yang diimplementasikan sudah berjalan dengan benar atau belum. Pengujian validitas algoritme SHA-256 menggunakan metode *Known Answer Test* (KAT), yaitu dengan mencocokan hasil dari algoritme yang diterapkan menggunakan data *test vector.* Data *test vector* berisi *key, input vector* yang berupa data, hasil dari *key* dan data yang dimasukkan akan menghasilkan MAC. Apabila *output* MAC yang dihasilakan berbeda dari pembuat algoritme SHA-2 maka algoritme SHA-2 yang diterapkan belum benar.

Pada pengujian ini terdapat 7 kali percobaan yang disediakan oleh pembuat algoritme SHA-2 dengan panjanng *output* 256 bit yang menghasilkan nilai MAC yang berbeda sesuai dengan *key* dan data yang dimasukkan.

Tabel 6.1 Pengujian *Test Vector*

No	Key	Data	Output	Hasil
1	0b0b0b0b 0b0b…0b	4869205468657265	b0344c61d8db3853 5ca8afceaf0bf12b88 1dc200c9833da726e 9376c2e32cff7	Valid
2	4a656665	7768617420646f207961207 7616e7420666f72206e6f74 68696e673f	5bdcc146bf60754e6 a042426089575c75a 003f089d2739839de c58b964ec3843	Valid
3	aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaaaaaaa aaaa	dddddddddddddddddddd ddddddddddddddddd…ddd	773ea91e36800e46 854db8ebd09181a7 2959098b3ef8c122d 9635514ced565fe	Valid

Lanjutan Tabel 6.1 Pengujian *Test Vector*

No	Key	Data	Output	Hasil
4	aaaaaaaaa aaaa...aaa	5465737420557369 6e67204c6172676572205468…374	60e431591ee0b67f0 d8a26aacbf5b77f8e 0bc6213728c514054 6040f0ee37f54	Valid
5	aaaaaaaaa aaaa...aaa	5468697320697320612074 657374207573696e67…d2e	9b09ffa71b942fcb27 635fbcd5b0e944bfd c63644f0713938a7f 51535c3a35e2	Valid
6	01020304 0506…819	Cdcdcdcdcdcdcdcdcdcdcdcd cdcdcdcdcdcdcdcdcd…cdcd	82558a389a443c0ea 4cc819899f2083a85f 0faa3e578f8077a2e 3ff46729665b	Valid
7	0c0c0c0c0 c…0c0c0c	5465737420576974682054 72756e636174696f6e	a3b6167473100ee0 6e0c796c2955552bf a6f7c0a6a8aef8b93f 860aab0cd20c5	Valid

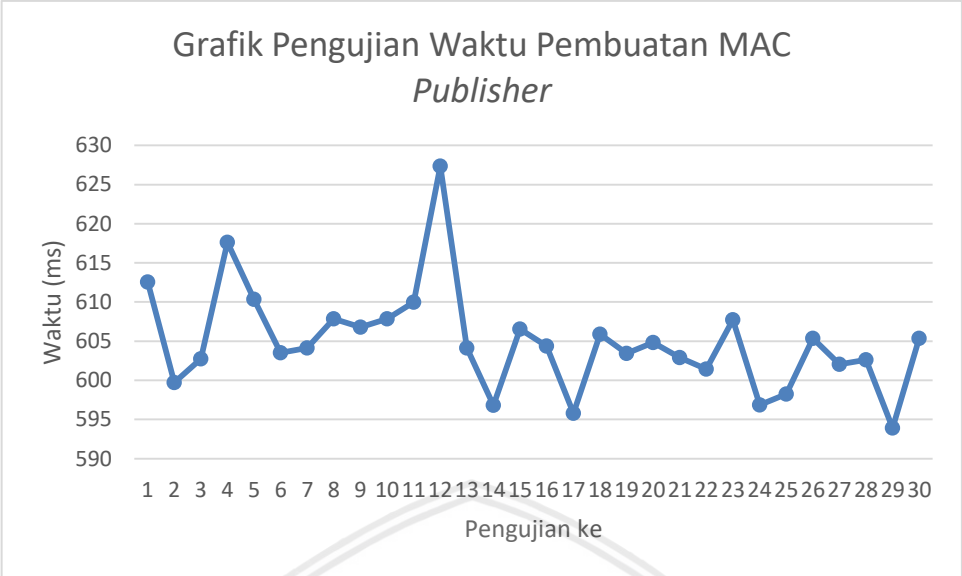
```
================================ test vector sha-2 ================================
mac : b0344c61d8db38535ca8afceaf0bf12b881dc200c9833da726e9376c2e32cff7 | test : valid
mac : 5bdcc146bf60754e6a042426089575c75a003f089d2739839dec58b964ec3843 | test : valid
mac : 773ea91e36800e46854db8ebd09181a72959098b3ef8c122d9635514ced565fe | test : valid
mac : 60e431591ee0b67f0d8a26aacbf5b77f8e0bc6213728c5140546040f0ee37f54 | test : valid
mac : 9b09ffa71b942fcb27635fbcd5b0e944bfdc63644f0713938a7f51535c3a35e2 | test : valid
mac : 82558a389a443c0ea4cc819899f2083a85f0faa3e578f8077a2e3ff46729665b | test : valid
mac : a3b6167473100ee06e0c796c2955552bfa6f7c0a6a8aef8b93f860aab0cd20c5 | test : valid
PS E:\>
```

Gambar 6.1 Hasil Pengujian *test vector*

Tabel 6.1 merupakan hasil pengujian yang dilakukan dengan skenario memasukkan *key* dan data yang dapat dilihat pada Tabel 6.1. Dalam pengujian ini seluruh skenario yang dilakukan memberikan hasil sama dengan hasil dari *output vector,* maka dapat disimpulkan bahwa algoritme SHA-2 yang diimplementasikan sudah sesuai dan benar.

6.2 Waktu Esekusi Algoritme SHA-256

Pengujian waktu esekusi algoritme SHA-256 digunakan unutk mengetahui performa dari algoritme SHA-256 dengan cara menghitung waktu yang dibutuhkan algoritme SHA-256 untuk menghasilkan kode MAC sepanjang 256 *bits.* Hasil dari pengujian ini dijadikan efektifitas penggunaan algoritme terhadap sistem MQTT yang diterapkan. Skenario pengujian waktu eseksusi yaitu dengan menjalankan kode program algoritme SHA-256 pada *publisher* dan *subscriber* sebanyak 30 kali dan diulang sebanyak 10 kali dengan data *input* dari data sensor yang digunakan. Hasil pengujian waktu pembuatan MAC SHA-256 ditunjukkan melalui grafik pada Gambar 6.2 dan Gambar 6.3.



Gambar 6.2 Grafik Perbandingan Waktu Pembuatan MAC Algoritme SHA-256 pada *Publisher*

Hasil pengujian waktu pembuatan MAC pada *publisher* berdasarkan Gambar 6.2 menunjukkan bahwa waktu pembuatan MAC algoritme SHA-256 mengalami kenaikan setelah eksekusi kedua. Kenaikan waktu proses pembuatan MAC ini mencapai waktu tertinggi yaitu 627,228 ms yang disebabkan *publisher* melakukan koneksi ulang ke *broker*. Sedangkan nilai paling rendah pada *publisher* terdapat pada pengujian ke-29 yaitu 593,924 ms

Gambar 6.3 Grafik Perbandingan Waktu Pembuatan MAC Algoritme SHA-256 pada *Subscriber*

Hasil pengujian waktu pembuatan MAC pada *subscriber* berdasarkan Gambar 6.3 menunjukkan bahwa waktu pembuatan MAC algoritme SHA-256. Jika dibandingkan dengan waktu pembuatan MAC pada *publisher*, waktu tertinggi berada di tengah proses sedangkan pada *subscriber* mencapai nilai tertinggi berada pada akhir proses pengujian dengan nilai 737,532 ms. Sedangkan nilai paling rendah pada *subscriber* dan *publisher* berada pada akhir pengujian dengan

nilai 705,074 ms. Perbandingan waktu pembuatan MAC pada *publisher* dan *subscriber* dijelaskan melalui Tabel 6.2.

Tabel 6.2 Hasil Pengujian Waktu pembuatan MAC Algoritme SHA-256

Keterangan	*Publisher* (ms)	*Subscriber* (ms)
Nilai Maksimum	627,338	737,532
Rata-Rata	604,958	717,344
Nilai Minimum	593,924	705,074

Perbandingan hasil pengujian waktu pembuatan MAC berdasarkan Tabel 6.3 didapatkan rata-rata yaitu 604,958 ms pada *publisher* dan 717,344 ms pada *subscriber*. Dengan nilai tersebut maka dapat diambil selisih waktu rata-rata pembuatan MAC pada *subscriber* lebih besar sekitar 112,386 ms. Hal ini disebabkan karena *subscriber* dijalankan pada mesin *virtual*.

6.3 Pengujian Fungsional Sistem

Bagian ini berisi tentang hasil dan analisis fungsional dari sistem yang di ujikan. Pengujian ini memastikan bahwa semua fitur telah berjalan dan sesuai dan dapat berjalan dengan benar. Keberhasilan pengujian di dapatkan dari hasil yang sesuai dengan yang di harapkan. Hasil dari pengujian fungsional di jelaskan pada Pengujian pembuatan MAC oleh algoritme SHA-256 , pengujian pengambilan data dari masing- masing sensor, pengujian pengambilan data publisher, pengujian MQTT *publish*, pengujian MQTT *subscribe*, dan pengujian pengecekan integritas data sensor oleh *subscriber*.

6.3.1 Pengujian pembuatan MAC oleh Algoritme SHA-256

Pengujian pembuatan MAC oleh algoritme SHA-256 digunakan untuk memastikan algoritme SHA-256 yang diimplementasikan pada *publihser* dan *subscriber* dapat memproses *key* dan data sensor yang diberikan sebagai masukkan dapat diubah menjadi MAC.

1. Kode : P_01
2. Prosedur Pengujian :
   a. Memasukkan *key* dan data sensor yang sama pada *publisher* dan *subsciber*.
   b. Mencocokkan MAC yang diperoleh dari *subscriber* dengan MAC yang diperoleh dari *publisher.*
   c. Memasukkan data sensor yang sama tetapi *key* yang berbeda pada *publisher* dan subsciber.
   d. Mencocokkan MAC yang diperoleh dari *subscriber* dengan MAC yang diperoleh dari *publisher.*
3. Hasil yang diharapkan :
   MAC yang diperoleh dari *subscriber* sama dengan MAC yang diperoleh dari *publisher* apabila *key* dan data sensor yang dimasukkan sama.
4. Hasil pengujian :

Publisher dan *subsciber* menghasilkan MAC yang sama saat *key* dan data sensor yang dimasukkan sama namun saat data sensor yang digunakan sama tetapi *key* berbeda maka menghasilkanMAC yang berbeda pula antara *publisher* dan *subsciber.*

Gambar 6.4 Hasil Pengujian pembuatan MAC oleh Algoritme SHA-256

Gambar 6.4 adalah hasil dari pengujian pembuatan MAC oleh SHA-256 yang di implementasikkan pada *publisher* dan *subsciber.* Saat diberi masukkan yang *key* dan data sensor yang sama antara *publisher* dan *subsciber* menghasilkan MAC yang sama. Namun pada saat *subscriber* diberi data yang sama tetapi *key* yang berbeda dengan *publisher, subsciber* menghasilkan MAC yang berbeda. Hal ini bertujuan untuk membutikan bahwa algoritme SHA-256 dapat diterapkan pada *publisher* dan *subsciber* berhasil memproses *key.*

6.3.2 Pengujian pengambilan data dari masing- masing sensor

Pengujian pengambilan data dari masing- masing sensor dilakukan unutk memastikan Raspberry Pi dapat membaca data dari ketiga sensor yang digunakan sebagai *input* data.

1. Kode : P_02
2. Prosedur Pengujian
    a. Membaca data dan menampilkan dari sensor Suhu DS18B2O
    b. Membaca data dan menampilkan dari sensor Ultra Sonik HC-SR04
    c. Membaca data dan menampilkan dari sensor Cahaya Photoresistor LDR GL5528
3. Hasil yang diharapkan :
   Raspberry Pi dapat membaca dan menapilkan nilai data dari masing-masing sensor yang digunakan.
4. Hasil pengujian
   Raspberry Pi berhasil membaca nilai dan menampilakn data dari masing-masing sensor yang digunakan.

```
yogi@ubuntu2:~$ python3 sensor_suhu.py
data = suhu : 20.769
data = suhu : 11.389
data = suhu : 32.072
data = suhu : 30.559
data = suhu : 28.997
data = suhu : 25.323
data = suhu : 24.516
data = suhu : 31.98
data = suhu : 20.753
data = suhu : 25.11
data = suhu : 13.844
data = suhu : 26.115
data = suhu : 27.627
data = suhu : 25.967
data = suhu : 34.172
data = suhu : 32.279
data = suhu : 21.129
data = suhu : 24.461
data = suhu : 30.214
data = suhu : 27.375
data = suhu : 34.672
data = suhu : 17.123
data = suhu : 18.969
data = suhu : 29.408
data = suhu : 11.302
data = suhu : 17.544
data = suhu : 25.722
data = suhu : 14.34
data = suhu : 26.4
data = suhu : 21.452
```

Gambar 6.5 Hasil Pengujian pengambilan data dari sensor Suhu DS18B2O

Gambar 6.5 merupakan hasil dari pengambilan data dari sensor Suhu DS18B2O oleh Raspberry Pi 3 yang diambil sebanyak 30 kali. Data dari yang terbaca oleh sensor Suhu DS18B2O dapat ditampilakan oleh Raspberry Pi 3 sebagai nilai dari suhu air di kolam air ikan hias.

```
yogi@ubuntu2:~$ python3 sensor_ultrasonik.py
data = ketinggian : 150.951
data = ketinggian : 166.266
data = ketinggian : 175.756
data = ketinggian : 151.221
data = ketinggian : 152.998
data = ketinggian : 199.184
data = ketinggian : 161.72
data = ketinggian : 154.094
data = ketinggian : 176.021
data = ketinggian : 158.094
data = ketinggian : 153.573
data = ketinggian : 197.432
data = ketinggian : 156.078
data = ketinggian : 195.179
data = ketinggian : 173.328
data = ketinggian : 160.462
data = ketinggian : 194.595
data = ketinggian : 164.473
data = ketinggian : 173.15
data = ketinggian : 171.437
data = ketinggian : 199.243
data = ketinggian : 161.53
data = ketinggian : 158.452
data = ketinggian : 152.45
data = ketinggian : 182.59
data = ketinggian : 165.037
data = ketinggian : 173.8
data = ketinggian : 180.27
data = ketinggian : 190.922
data = ketinggian : 181.515
```

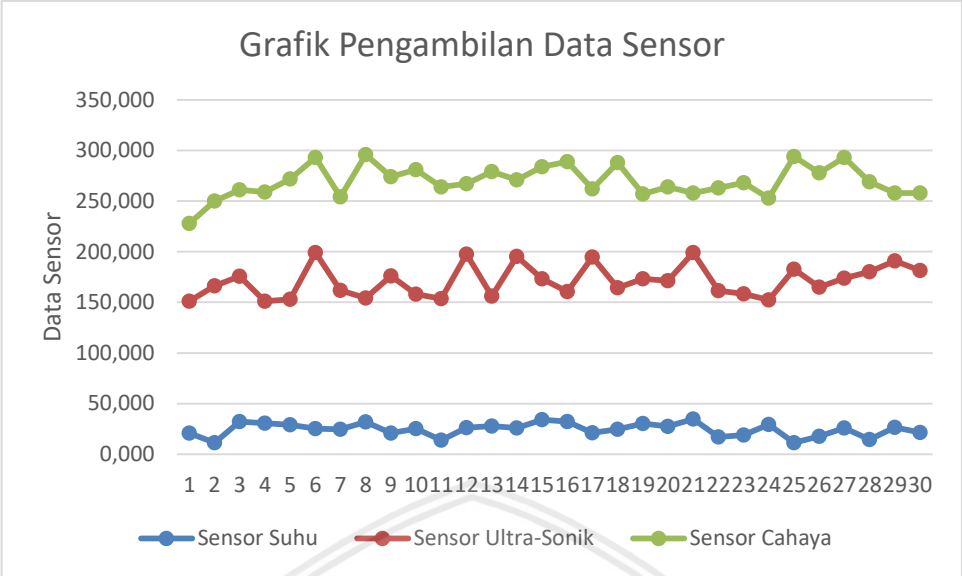
Gambar 6.6 Hasil Pengujian pengambilan data dari sensor Ultra Sonik HC-SR04

Gambar 6.6 merupakan hasil dari pengambilan data dari sensor Ultra Sonik HC-SR04 oleh Raspberry Pi 3 yang diambil sebanyak 30 kali. Data dari yang terbaca oleh sensor Ultra Sonik HC-SR04 dapat ditampilakan oleh Raspberry Pi 3 sebagai nilai dari ketinggian air didalam kolam ikan hias.

```
yogi@ubuntu2:~$ python3 sensor_cahaya.py
data = kekeruhan : 288
data = kekeruhan : 250
data = kekeruhan : 261
data = kekeruhan : 259
data = kekeruhan : 272
data = kekeruhan : 293
data = kekeruhan : 254
data = kekeruhan : 296
data = kekeruhan : 274
data = kekeruhan : 281
data = kekeruhan : 264
data = kekeruhan : 267
data = kekeruhan : 279
data = kekeruhan : 271
data = kekeruhan : 284
data = kekeruhan : 289
data = kekeruhan : 262
data = kekeruhan : 288
data = kekeruhan : 257
data = kekeruhan : 264
data = kekeruhan : 258
data = kekeruhan : 263
data = kekeruhan : 268
data = kekeruhan : 253
data = kekeruhan : 294
data = kekeruhan : 278
data = kekeruhan : 293
data = kekeruhan : 269
data = kekeruhan : 258
data = kekeruhan : 258
```

Gambar 6.7 Hasil Pengujian pengambilan data dari sensor Cahaya Photoresistor LDR GL5528

Gambar 6.7 merupakan hasil dari pengambilan data dari Cahaya Photoresistor LDR GL5528 oleh Raspberry Pi 3 yang diambil sebanyak 30 kali. Data dari yang terbaca Cahaya Photoresistor LDR GL5528 dapat ditampilakan oleh Raspberry Pi 3 sebagai nilai kekeruhan air didalam kolam ikan hias. Hasil analisa pada Gambar 6.5, Gambar 6.6, dan Gambar 6.7 dijelaskan didalam grafik yang ditunjukkan pada Gambar 6.8.



Gambar 6.8 Grafik pengambilan data Sensor

Berdasarkan pada grafik pada Gambar 6.8 diperoleh data dari 3 sensor yang digunakan. Pada pengujian pengambilan data sensor yang pertama adalah sensor suhu yang mempunyai perbedaan pembacaan suhu maksimum dan minimum sekitar 23,370 *Celsius.* Pada sensor Ultra-sonik perbedaan pembacaan tinggi air kolam maksimum dan minimum sekitar 48,292 cm dan pada sensor cahaya mempunyai perbedaan pembacaan intensitas cahaya maksimum dan minimum sekitar 68. Sedangkan Tabel 6.3 merupakan hasil pengujian pengambilan data sensor dengan nilai maksimum, rata-rata, dan nilai minimum.

Tabel 6.3 Pengambilan data senSor

Keterangan	Sensor Suhu	Sensor Ultra-sonik	Sensor Cahaya
Nilai Maksimum	34,672	199,243	296
Rata - rata	24,386	171,0607	269.5
Nilai Minimum	11,302	150,951	228

Berdasarkan nilai rata-rata pada Tabel 6.4 dan grafik pada Gambar 6.8 dapat diambil kesimpulan bahwa suhu rata- rata kolam sekitar 24,386 *Celsius* yang dijadikan sebagai suhu idela kolam, pembacaan Sensor Ultra-sonik sekitar 171,0607 cm yang dijadikan ketinggian ideal air kolam serta pembacaan sensor cahaya sekitar 269,5 yang dijadikan tingkat transparansi air.

6.3.3 Pengujian pengambilan data sensor *Publisher*

Pengujian pengambilan data dari sensor *publisher* dilakukan untuk memastikan *publisher* dapat menampilkan dan mem-*publish* data sensor yang digunakan.

1. Kode : P_03
2. Prosedur Pengujian

a. Menampilkan data dari pembacaan 3 sensor yang dilakukan oleh Raspberry Pi.
b. Melakukan *pubslih* data dari 3 sensor yang digunakan.

3. Hasil yang diharapkan :
   Publisher dapat menampilkan dan bisa melakukan *publish* data sensor yang digunakan.
4. Hasil pengujian
   Publisher berhasil menampilkan dan bisa melakukan *publish* data sensor yang digunakan.

```
yogi@ubuntu2:~$ python3 publish1.py
data = {kekeruhan : 282 suhu : 24.347 ketinggian : 160.263}
data = {kekeruhan : 297 suhu : 12.06 ketinggian : 182.782}
data = {kekeruhan : 278 suhu : 29.385 ketinggian : 161.742}
data = {kekeruhan : 281 suhu : 25.985 ketinggian : 155.176}
data = {kekeruhan : 297 suhu : 21.861 ketinggian : 173.422}
data = {kekeruhan : 267 suhu : 20.934 ketinggian : 161.387}
data = {kekeruhan : 297 suhu : 30.406 ketinggian : 196.588}
data = {kekeruhan : 292 suhu : 17.799 ketinggian : 160.012}
data = {kekeruhan : 258 suhu : 16.999 ketinggian : 159.499}
data = {kekeruhan : 260 suhu : 28.76 ketinggian : 162.153}
data = {kekeruhan : 280 suhu : 21.869 ketinggian : 179.531}
data = {kekeruhan : 275 suhu : 24.077 ketinggian : 182.912}
data = {kekeruhan : 278 suhu : 27.509 ketinggian : 156.258}
data = {kekeruhan : 281 suhu : 21.329 ketinggian : 160.7}
data = {kekeruhan : 268 suhu : 20.364 ketinggian : 172.374}
data = {kekeruhan : 262 suhu : 32.786 ketinggian : 196.386}
data = {kekeruhan : 265 suhu : 24.162 ketinggian : 190.428}
data = {kekeruhan : 279 suhu : 20.219 ketinggian : 185.539}
data = {kekeruhan : 288 suhu : 13.378 ketinggian : 163.475}
data = {kekeruhan : 253 suhu : 28.179 ketinggian : 187.266}
data = {kekeruhan : 269 suhu : 12.855 ketinggian : 152.408}
data = {kekeruhan : 273 suhu : 31.792 ketinggian : 184.947}
data = {kekeruhan : 284 suhu : 27.08 ketinggian : 190.994}
data = {kekeruhan : 288 suhu : 34.209 ketinggian : 182.583}
data = {kekeruhan : 285 suhu : 20.774 ketinggian : 166.382}
data = {kekeruhan : 289 suhu : 29.841 ketinggian : 157.668}
data = {kekeruhan : 288 suhu : 16.499 ketinggian : 192.476}
data = {kekeruhan : 250 suhu : 16.328 ketinggian : 153.62}
data = {kekeruhan : 288 suhu : 33.222 ketinggian : 152.044}
data = {kekeruhan : 296 suhu : 19.675 ketinggian : 197.744}
```

Gambar 6.9 Hasil Pengujian pengambilan data sensor *publisher*

Gambar 6.9 merupakan hasil dari pengujian pengambilan data sensor *publisher.* Data yang dibaca oleh sensor dapat ditampilkan oleh *publisher* sekaligus yang dilakukan sebanyak 30 kali, data berhasil di kirim tanpa ada data yang hilang.

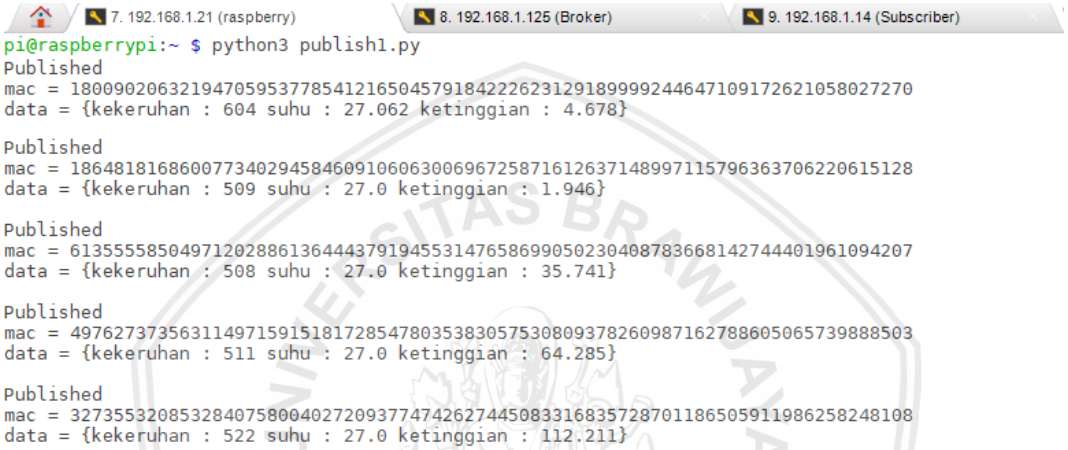
6.3.4 Pengujian MQTT *Publish*

Pengujian MQTT *publish* dilakukan untuk memastikan *publisher* dapat melakukan *publish* dengan *payload* JSON yang berisi MAC SHA-256 dan data sensor.

1. Kode : P_04
2. Prosedur Pengujian :

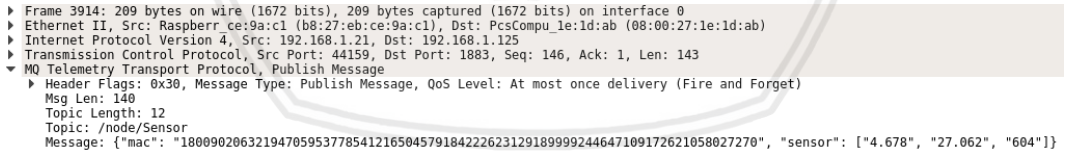
a. Menjalankan hasil kompilasi dengan nama kode program `publish1.py`.
b. Menampilkan dan mencocokkan *payload publish* dengan Wireshark.

3. Hasil yang diharapkan :
   Publisher dapat melakukan *publish* data dengan *payload* JSON yang berisi MAC SHA-256 dan data sensor.
4. Hasil Pengujian :
   Publisher berhasil melakukan *publish* data dengan *payload* JSON yang berisi MAC SHA-256 dan data sensor.



Gambar 6.10 Hasil Pengujian MQTT *Publish*

Gambar 6.10 merupakan hasil dari pengujian MQTT *publish* dengan *payoad* JSON yang berisi MAC SHA-256 serta data sensor yang bernilai 604, 27.062, dan 4.678 . Dari 5 kali pengujian di *screenshot* ini mewakili 30 kali pengujian MQTT *publish.*



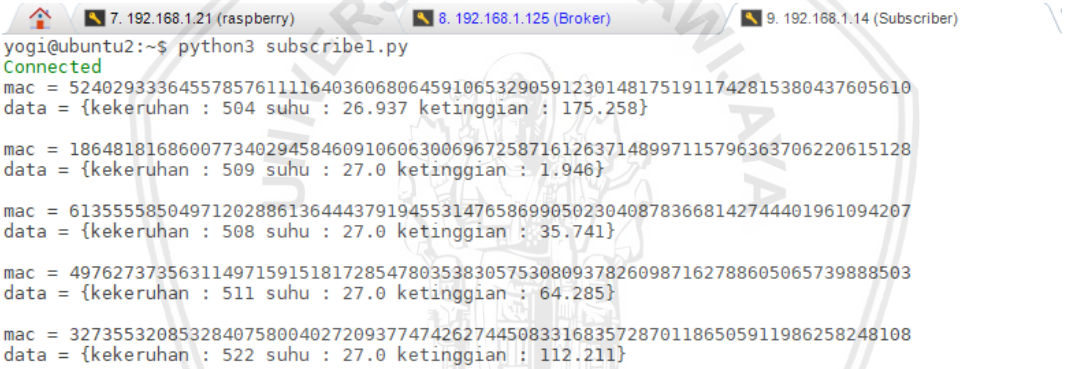
Gambar 6.11 Perbandingan Hasil Pengujian MQTT *Publish* menggunakan Wireshark

Gambar 6.11 adalah hasil dari Perbandingan Hasil dari Pengujian MQTT Publish menggunkan Wireshark. Pada Gambar 6.11 dapat diambil kesimpulan bahwa pengujian MQTT *publish* telah berhasil dilakukan karena *payolad* dari hasil *publish* pada Wireshark dengan *payload publish* pada *publisher* bernilai sama yang berisi MAC SHA-256 dan data sensor.

6.3.5 Pengujian MQTT *Subscribe*

Pengujian MQTT *subscribe* dilakukan untuk memastikan *susbcriber* dapat menerima data dari *publisher* serta membaca *payload* JSON yang berisi MAC SHA-256 dan data sensor.

1. Kode : P_05
2. Prosedur Pengujian :
    a. Menjalankan hasil kompilasi dengan nama kode program `subscribe1.py`.
    b. Menampilkan *payload* dan hasil JSON *unmarshaling* terhadap *payload* yang diterima.
3. Hasil yang diharapkan :
   Subscriber dapat menerima dan membaca *payload* JSON dari *publisher* yang berisikan MAC SHA-256 dan data sensor.
4. Hasil Pengujian :
   Subscriber berhasil menerima *payload* dan dapat ditampilakan serta dibaca melalui proses JSON *unmarshaling*.



Gambar 6.12 Hasil Pengujian MQTT *Subscribe*

Gambar 6.12 adalah hasil pengujian MQTT *subscribe* yang menampilkan hasil dari *payload publisher* dan membaca *payload* dengan proses JSON *unmarshaling.* Kemudian *subscriber* menampilkan hasil dari pembacaan *payload* setelah melakukan proses JSON *unmarshaling* setelah itu *subscriber* menampilkan *payload* asli yang memiliki format JSON. Dapat disimpulkan bahwa *subscribe* telah berhasil karena *payload* yang diterima dapat ditampilkan dan dibaca oleh *subscriber.* Pengujian MQTT *subscribe* dilakukan sebanyak 30 kali pengujian yang diwakili oleh 5 kali pengujian di dalam *screenshot*.

6.3.6 Pengujian pengecekan integritas data sensor oleh *Subscriber*

Pengujian pengecekan integritas data sensor dilakukan untuk memastikkan *subsciber* dapat menghasilkan MAC dari data sensor yang diterima kemudian membandingkan MAC yang dihasilkan oleh *subsciber* dengan MAC yang dihasilkan oleh *publisher.*

1. Kode : P_06
2. Prosedur Pengujian :
    a. Menampilkan hasil dari payload yang diterima oleh publisher

b. Membaca payload json dari publisher yang berupa data sensor untuk dijadikan MAC SHA-256 .
c. Membandingkan hasil MAC dari *subscriber* dengan MAC yang diterima dari *publisher.*
d. Menampilkan hasil perbandingan MAC melalui nilai *boolean* benar atau salah.

3. Skenario pengujian 1 :
   Pengujian tanpa mengubah data dari sensor.
4. Skenario pengujian 2 :
   Pengujian dengan mengubah data dari sensor.
5. Hasil yang diharapkan
   Subscriber dapat mengetahui data sensor yang tidak mengalami pengubahan dan data sensor yang diubah melalui MAC yang dihasilkan oleh *subsciber* dan MAC yang diterima dari *publisher.*
6. Hasil pengujian
   Subscriber berhasil menentukan data sensor yang tidak diubah dan data sensor yang mengalami pengubahan.

```
mac_pub= 411053829783728622637079770422884537829422056200573453329367642546803678957 27
mac_sub= 411053829783728622637079770422884537829422056200573453329367642546803678957 27
data = {kekeruhan : 293 suhu : 26.21 ketinggian : 171.162}
MAC_SUB = MAC_PUB ? TRUE

mac_pub = 86203703422163822340697601466010801092401023832342147919169252011104880944819
mac_sub= 34707259899685444973081472087479363796375314745435128545536136863936412155600
data = {kekeruhan : 666 suhu : 23.456 ketinggian : 135.78}
MAC_SUB = MAC_PUB ? FALSE
```

Gambar 6.13 Hasil Pengujian Pengecekan Integritas data sensor oleh *Subscriber*

Gambar 6.13 merupakan hasil dari pengujian pengecekan integritas data sensor oleh *subscriber.* Didalam pengujian ini terdapat dua skenario yaitu saat pengujian pertama data sensor tidak di ubah dan pada pengujian kedua data sensor diubah secara langsung. Berdasarkan hasil yang didapatkan pada Gambar 6.12 pengujian integritas data sensor oleh *subscriber* berhasil dilakukan karena pada saat data sensor tidak diubah *subscriber* menampilkan nilai *boolean TRUE* dan pada saat data sensor diubah *subscriber* menampilkan nilai *boolean FALSE.*

6.4 Pengujian Performa Sistem

Pengujian performa sistem menjelaskan tentang analisis setelah dilakukannya pengujian penggunanan *memory* di dalam performa sistem yang dilakukan untuk mengetahui saat algoritme SHA-2 di implementasikan di sistem yang digunakan. Cara menguji sistem ini yaitu dengan cara saat sistem menggunakan intergritas data dan tanpa integritas data, yang dilakukan sebanyak 30 kali yang diulang sebanyak 10 kali untuk mendapatkan hasil yang akurat. Menurut Hair dkk. (2010) ketika jumlah sampel sejumlah 30 atau lebih terdapat nilai normalitas pada sampel tersebut maka nilai normalitas tersebut akan mempengaruhi hasil dari pengujian.

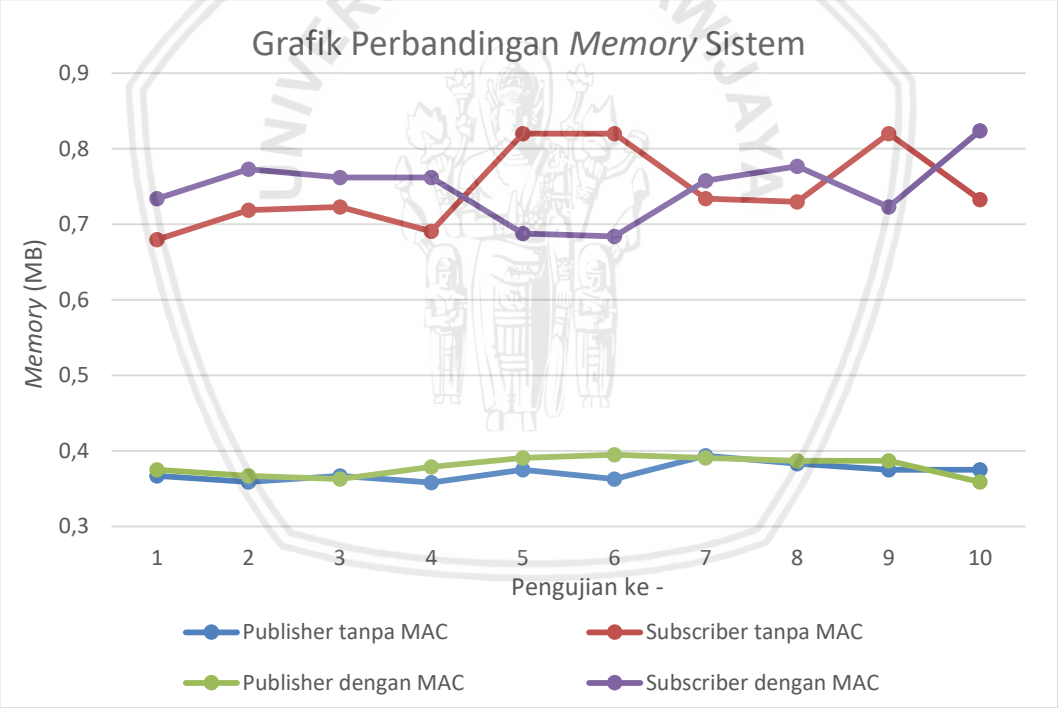
Analisis penggunaan *memory* pada *publisher* dan *subscriber* menggunakan perangkat lunak psRecord. Berikut ini adalah cara yang digunakan untuk instalasi psRecord pada *publisher* dan *subscriber* ditunjukkan pada Tabel 6.4.

Tabel 6.4 Proses Instalasi psRecord

No	Perintah
1	`$ sudo pip install psutil matplotlib psrecord`

1. Baris 1 : Perintah yang digunakan untuk instalasi psRecord. Pada perintah instalasi terdapat *library* tambahan yang diperlukan sebelum psRecord di install yaitu `psutil` dan `matplotlib`.

Setelah psRecord berhasil di *install* pengujian dapat dilakukan. Berikut ini adalah hasil dari pengujian *memory* pada *publisher* dan *subscriber* yang ditunjukkan oleh Gambar 6.14. Menurut Hair dkk. (2010) ketika jumlah sampel sejumlah 30 atau lebih terdapat nilai normalitas pada sampel tersebut maka nilai normalitas tersebut akan mempengaruhi hasil dari pengujian. Oleh sebab itu, setiap skenario pengujian dilakukan sebanyak 30 kali dan diulang sebanyak 10 kali agar mendapat hasil yang akurat.



Gambar 6.14 Grafik Perbandingan penggunaan *Memory*

Pengujian penggunaan *memory* dilakukan melalui perangkat lunak psRecord baik pada *publisher* maupun *subscriber*. Hasil pengujian terhadap penggunaan *memory* pada *publisher* dan *subscriber* ditunjukkan melalui Gambar 6.14 dan Tabel 6.4. Berdasarkan grafik yang ditunjukkan pada Gambar 6.14, perbedaan umum penggunaan *memory* pada *publisher* dan *subscriber* yaitu sekitar 0,3 MB. Penggunaan *memory* pada *publisher* mencapai nilai tertinggi 0,358 MB sebelum penerapan MAC SHA-256 dan 0,359 MB setelah penerapan algoritme SHA-256 . Sedangkan penggunaan *memory* pada *subscriber* mencapai nilai tertinggi 0,68 MB

sebelum penerapan MAC SHA-256 dan 0,684 MB setelah penerapan algoritme SHA-256 . Hasil pengujian penggunaan *memory* pada *publisher* dan *subscriber* sebelum dan setelah penerapan MAC SHA-256 ditunjukkan melalui Tabel 6.5.

Tabel 6.5 Hasil Penghujian Penggunaan *Memory*

Keterangan	*Publisher* (MB)		*Subscriber* (MB)	
	Tanpa MAC	Dengan MAC	Tanpa MAC	Dengan MAC
Nilai Maksimum	0,358	0,359	0,68	0,684
Rata - rata	0,3716	0,3794	0,747	0,7485
Nilai Minimum	0,394	0,395	0,82	0,824

Hasil pengujian penggunaan *memory* berdasarkan Tabel 6.6 didapati bahwa rata-rata penggunaan *memory* pada *publisher* sebelum penggunaan MAC yaitu 0,3716 MB dan 0,3794 MB setelah penggunaan MAC SHA-256 . Sedangkan rata-rata penggunaan *memory* pada *subscriber* sebelum penggunaan MAC yaitu 0,747 MB dan 0,7485 MB setelah penggunaan MAC SHA-256 . Dari hal ini dapat disimpulkan bahwa algoritme SHA-256 menggunakan *memory* sebesar 0,0082 MB pada *publisher* dan 0,0015 MB pada *subscriber*.

6.5 Pengujian Keamanan Sistem

Pengujian keamanan sistem menjelaskan tentang analisis keamanan data saat sistem diimplementasikan berdasarkan dari integritas data, pengujian ini dilakukan unutk membuktikan sistem dapat membedakan data yang benar dan data yang sudah di ubah saat data di transmisikan oleh *publisher.* Di dalam pengujian ini menggunakan serangan *Man-in-the-Midlle.*

Man-in-the-Midlle merupakan serangan yang dilakukan oleh *attacker* yang bisa mengubah data yang ditransmisikan secara rahasia antara dua pihak yang saling berkomunikasi. Contoh metode yang digunakan MITM untuk melakukan serangan yaitu DHCP *spoofing*, ARP *poisoning*, *transparent* proxy, dan DNS *spoofing.* Beberapa tools yang digunakan MITM yaitu Burp Suite, Ettercap, dan ZAP.

Pengujian keamanan sistem ini menggunakan *attacker* yang berupa mesin virtual Ubuntu Server, dan menggunakan *Ettercap* sebagai tools yang digunakan untuk melakukan serangan MITM.

Pengujian keamanan sistem dilakukan menggunakan sebuah mesin virtual Ubuntu Server yang berperan sebagai *attacker*. Sedangkan *tools* yang dipilih untuk melakukan pengujian yaitu Ettercap karena fleksibilitasnya, terutama pada filter yang dapat dibuat sesuai kebutuhan. Adapun cara yang digunakan untuk menginstal Ettercap pada *attacker* yang ditunjukkan pada Tabel 6.6.

Tabel 6.6 Proses Instalasi Ettercap pada *Attacker*

No	Perintah
1	`$ sudo apt install zlib1g zlib1g-dev build-essential \` `> ettercap-text-only -y`

1. Baris 1 : Perintah yang digunakan untuk menginstal Ettercap pada *attacker.* Berdasarkan perintah instalasi Ettercap yang dilakukan, terdapat *library* lain yang dibutuhkan yaitu `zlib` dan `build-essential`.

Terdapat 3 pengujian yang dilakukan menggunakan filter Ettercap yaitu pengubahan, penyubstitusian, dan penyisipan.

6.5.1 Pengujian Pengubahan Data

Pengujian pengubahan data adalah pengujian sistem dengan mengubah nilai semua data sensor yang ditransmisikan. Di dalam Pengujian ini Ettercap membutuhkan filter yang dibuat untuk mengubah data sensor dari *publisher* ditunjukkan pada Tabel 6.7.

Tabel 6.7 Kode Program Filter Ettercap Pengubahan Data

No	Perintah
1	`if (ip.proto == TCP && tcp.dst == 1883 && search(DATA.data,`
2	`"Sensor")) {`
3	`replace("sensor", sensor" : ["145.88", "23.666",`
4	`"635"]});`
5	`msg("MQTT payload replaced\n");`
6	`}`

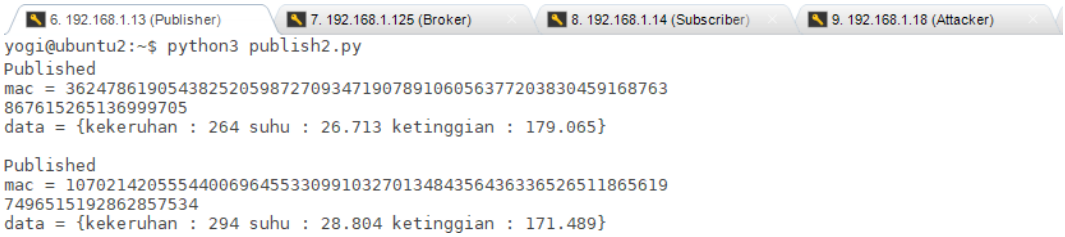
Kode program yang telah dibuat berfungsi untuk melakukan seleksi terhadap paket yang diterima *attacker* saat serangan dilakukan. Paket yang diterima diseleksi berdasarkan urutan kondisi pada kode program. Jika terdapat paket TCP dan nomor port tujuan 1883, serta berisi *string* "`Sensor`" yang berarti bahwa paket tersebut merupakan pesan MQTT dengan *topic* Sensor. Maka akan dilakukan pengubahan bagian `data` dengan nilai baru yaitu `145.88`, `23.666`, dan `635`. Setelah itu untuk menyimpan kode program ditunjukkan pada Tabel 6.8.

Tabel 6.8 Proses *compile* Ettercap Pengubahan Data

No	Perintah
1	`$ etterfilter mqttattacker.filter -o mqttattacker.ef`

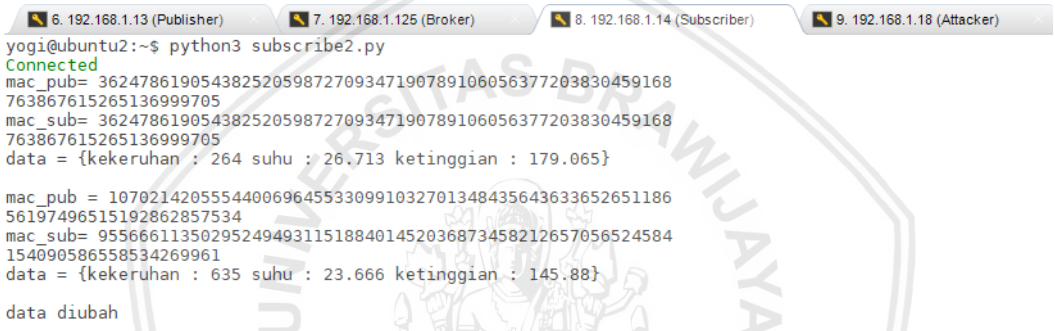
1. Baris 1 : Perintah yang digunakan untuk melakukan *compile* `mqttattacker.filter`

Etterfilter merupakan perangkat lunak khusus yang disediakan oleh Ettercap untuk melakukan kompilasi filter yang telah dibuat. Berdasarkan perintah kompilasi yang dilakukan, perintah tersebut menghasilkan *file* bernama `mqttattacker.ef` yang siap digunakan sebagai filter Ettercap. Skenario yang digunakan untuk menguji keamanan sistem yang diimplementasikan yaitu melakukan *publish* sebelum dan saat serangan menggunakan Ettercap. Gambar 6.15 menunjukkan data yang dikirimkan oleh *publisher*. Sesuai skenario yang telah ditentukan, data pertama adalah data sebelum serangan dilakukan sedangkan data kedua adalah data saat serangan dilakukan.



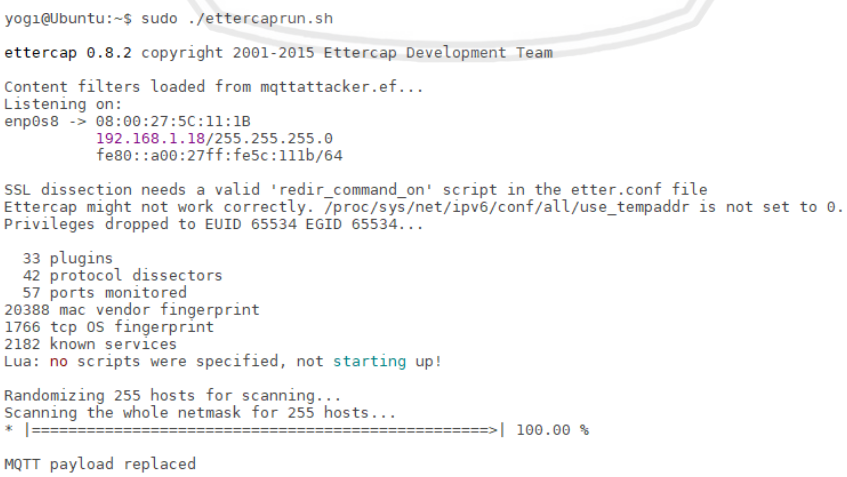
Gambar 6.15 Data *Publish* Pengujian Pengubahan Data

Tanggapan yang dihasilkan *subscriber* terhadap data yang diterima haruslah berbeda sebelum dan saat serangan dilakukan. Pada Gambar 6.15 dapat dilihat bahwa nilai data yang dikirimkan sebelum serangan yaitu kekeruhan : 294 suhu : 28,804, dan Ketinggian : 171,489, sedangkan data saat serangan adalah kekeruhan : 635, suhu : 23,666, dan Ketinggian : 145,88. Perbedaan tanggapan yang dihasilkan *subscriber* dapat dilihat pada Gambar 6.19.



Gambar 6.16 Data *Subscribe* Pengujian Pengubahan Data

Gambar 6.16 menunjukkan bahwa data yang diterima sebelum serangan merupakan data yang tidak diubah, karena nilai MAC yang dihitung *subscriber* sama dengan nilai MAC yang didapat dari *publisher*. Sedangkan data kedua yang diterima *subscriber* saat terdapat serangan berupa data yang telah dirubah, sebab nilai MAC yang dihasilkan *subscriber* tidak sesuai. Pengubahan data yang dilakukan oleh *attacker* ditunjukkan pada Gambar 6.17.



Gambar 6.17 Pengubahan Data Pada *Attacker*

Gambar 6.17 menunjukkan proses pengubahan data saat transmisi terjadi menggunakan metode ARP *poisoning* pada Ettercap. Serangan yang dilakukan merupakan skenario pengujian keamanan sistem kedua, yang mengakibatkan *subscriber* menerima data yang sudah dimodifikasi. Berdasarkan Gambar 6.15, *publisher* mengirimkan data yaitu kekeruhan : 294, suhu : 28,804, dan Ketinggian : 171,489 sedangkan *subscriber* menerima data dengan nilai kekeruhan : 635, suhu : 23,666, dan Ketinggian : 145,88 yang ditunjukkan pada Gambar 6.16. Namun *subscriber* dapat mengetahui pengubahan ini melalui perbedaan MAC yang dihasilkan.

6.5.2 Pengujian Penyubstitusian Data

Pengujian Penyubstitusian adalah pengujian sistem dengan mengganti sebagian nilai data sensor. Di dalam Pengujian ini Ettercap membutuhkan filter yang ditunjukkan pada Tabel 6.9.

Tabel 6.9 Kode Program Filter Ettercap Penyubstitusian Data

No	Perintah
1	`if (ip.proto == TCP && tcp.dst == 1883 && search(DATA.data,`
2	`"Sensor")){`
3	`DATA.data + 141 = "4";`
4	`msg("=> data:XXX replaced with data:XXY | X = original value`
5	`| Y = new value = 4\n");`
6	`}`

Kode program yang telah dibuat berfungsi untuk melakukan seleksi terhadap paket yang diterima *attacker* saat serangan dilakukan. Paket yang diterima diseleksi berdasarkan urutan kondisi pada kode program. Jika terdapat paket TCP dan nomor port tujuan 1883, serta berisi *string* "`Sensor`" yang berarti bahwa paket tersebut merupakan pesan MQTT dengan *topic* Sensor. Maka akan dilakukan pengubahan dengan mengganti nilai *payload* indeks ke 141 yaitu digit terakhir data diganti dengan nilai baru yaitu 4. Setelah itu untuk menyimpan kode program ditunjukkan pada Tabel 6.10.

Tabel 6.10 Proses *compile* Ettercap Penyubstitusian Data

No	Perintah
1	`$ etterfilter mqttattacker2.filter -o mqttattacker2.ef`

2. Baris 1 : Perintah yang digunakan untuk melakukan *compile* `mqttattacker2.filter`

Berdasarkan perintah kompilasi yang dilakukan, perintah tersebut menghasilkan *file* bernama `mqttattacker2.ef` yang siap digunakan sebagai filter Ettercap. Skenario yang digunakan untuk menguji keamanan sistem yang diimplementasikan yaitu melakukan *publish* sebelum dan saat serangan menggunakan Ettercap. Gambar 6.18 menunjukkan data yang dikirimkan oleh *publisher*. Sesuai skenario yang telah ditentukan, data pertama adalah data sebelum serangan dilakukan sedangkan data kedua adalah data saat serangan dilakukan.

```
Published
mac = 244489378987196238500044994120029863279881988736586107715688144257136330829 98
data = {kekeruhan : 277 suhu : 14.656 ketinggian : 172.133}

Published
mac = 417861143505487521152271447853009072853119038684690959984696325770431470880 97
data = {kekeruhan : 252 suhu : 22.106 ketinggian : 188.867}
```

Gambar 6.18 Data *Publish* Pengujian Penyubstitusian Data

Tanggapan yang dihasilkan *subscriber* terhadap data yang diterima haruslah berbeda sebelum dan saat serangan dilakukan. Pada Gambar 6.18 dapat dilihat bahwa nilai data yang dikirimkan sebelum serangan yaitu kekeruhan : 252, suhu : 22,106 dan Ketinggian : 188,867 , sedangkan data saat serangan adalah kekeruhan : 254, suhu : 22,106, dan Ketinggian : 188,867. Perbedaan tanggapan yang dihasilkan *subscriber* dapat dilihat pada Gambar 6.19.

```
mac_pub= 244489378987196238500044994120029863279881988736586107715688144257136330829 98
mac_sub= 244489378987196238500044994120029863279881988736586107715688144257136330829 98
data = {kekeruhan : 277 suhu : 14.656 ketinggian : 172.133}

mac_pub = 417861143505487521152271447853009072853119038684690959984696325770431470880 97
mac_sub= 567426441213477074183751633159173292217792312274698064053591797051337906870 07
data = {kekeruhan : 254 suhu : 22.106 ketinggian : 188.867}

data diubah
```

Gambar 6.19 Data *Subscribe* Pengujian Penyubstitusian Data

Gambar 6.18 menunjukkan bahwa data yang diterima sebelum serangan merupakan data yang tidak diubah, karena nilai MAC yang dihitung *subscriber* sama dengan nilai MAC yang didapat dari *publisher*. Sedangkan data kedua yang diterima *subscriber* saat terdapat serangan berupa data yang telah dirubah, sebab nilai MAC yang dihasilkan *subscriber* tidak sesuai. Pengubahan data yang dilakukan oleh *attacker* ditunjukkan pada Gambar 6.20.

```
yogi@Ubuntu:~$ sudo ./ettercaprun2.sh

ettercap 0.8.2 copyright 2001-2015 Ettercap Development Team

Content filters loaded from mqttattacker2.ef...
Listening on:
enp0s8 -> 08:00:27:5C:11:1B
          192.168.1.18/255.255.255.0
          fe80::a00:27ff:fe5c:111b/64

SSL dissection needs a valid 'redir_command_on' script in the ett
er.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/all/us
e_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %

=> data:XXX replaced with data:XXY | X = original value | Y = new
 value = 4
```

Gambar 6.20 Penyubstitusian Data Pada *Attacker*

Gambar 6.20 menunjukkan proses pengubahan data saat transmisi terjadi menggunakan metode ARP *poisoning* pada Ettercap. Serangan yang dilakukan merupakan skenario pengujian keamanan sistem kedua, yang mengakibatkan *subscriber* menerima data yang sudah dimodifikasi. Berdasarkan Gambar 6.18, *publisher* mengirimkan data yaitu kekeruhan : 252, suhu : 22,106, dan Ketinggian : 188,867 sedangkan *subscriber* menerima data dengan nilai kekeruhan : 254, suhu : 22,106, dan Ketinggian : 188,867 yang ditunjukkan pada Gambar 6.19. Namun *subscriber* dapat mengetahui pengubahan ini melalui perbedaan MAC yang dihasilkan.

6.5.3 Pengujian Penyisipan Data

Pengujian pengubahan data adalah pengujian sistem dengan penambahan data sensor yang ditransmisikan. Di dalam Pengujian ini Ettercap membutuhkan filter yang dibuat untuk mengubah data sensor dari *publisher* ditunjukkan pada Tabel 6.11.

Tabel 6.11 Kode Program Filter Ettercap Penyisipan Data

No	Perintah
1	`if (ip.proto == TCP && tcp.dst == 1883 && search(DATA.data,`
2	`"Sensor")){`
3	`replace("]", "0]");`
4	`msg("=> Inserted char 0 to data sensor\n");`
5	`}`

Kode program yang telah dibuat berfungsi untuk melakukan seleksi terhadap paket yang diterima *attacker* saat serangan dilakukan. Paket yang diterima diseleksi berdasarkan urutan kondisi pada kode program. Jika terdapat paket TCP dan nomor port tujuan 1883, serta berisi *string* "`Sensor`" yang berarti bahwa paket tersebut merupakan pesan MQTT dengan *topic* Sensor. Maka akan dilakukan penyisipan diakhir *payload* dengan nilai yaitu `0`. Setelah itu untuk menyimpan kode program ditunjukkan pada Tabel 6.12.

Tabel 6.12 Proses *compile* Ettercap Penyisipan Data

No	Perintah
1	`$ etterfilter mqttattacker3.filter -o mqttattacker3.ef`

3. Baris 1 : Perintah yang digunakan untuk melakukan *compile* `mqttattacker.filter`

Berdasarkan perintah kompilasi yang dilakukan, perintah tersebut menghasilkan *file* bernama `mqttattacker3.ef` yang siap digunakan sebagai filter Ettercap. Skenario yang digunakan untuk menguji keamanan sistem yang diimplementasikan yaitu melakukan *publish* sebelum dan saat serangan menggunakan Ettercap. Gambar 6.21 menunjukkan data yang dikirimkan oleh *publisher*. Sesuai skenario yang telah ditentukan, data pertama adalah data sebelum serangan dilakukan sedangkan data kedua adalah data saat serangan dilakukan.

```
Published
mac = 544156099606279493162861023260112477192327569866540314866890827010433986667706
data = {kekeruhan : 253 suhu : 31.395 ketinggian : 155.139}

Published
mac = 387515353268037051858449727050126655728224429747049901279025637458834618147 55
data = {kekeruhan : 255 suhu : 23.611 ketinggian : 177.615}
```

Gambar 6.21 Data *Publish* Pengujian Penyisipan Data

Tanggapan yang dihasilkan *subscriber* terhadap data yang diterima haruslah berbeda sebelum dan saat serangan dilakukan. Pada Gambar 6.21 dapat dilihat bahwa nilai data yang dikirimkan sebelum serangan yaitu kekeruhan : 255, suhu : 23,611, dan Ketinggian : 177,615, sedangkan data saat serangan adalah kekeruhan : 2550, suhu : 23,611, dan Ketinggian : 177,615. Perbedaan tanggapan yang dihasilkan *subscriber* dapat dilihat pada Gambar 6.22.

```
mac_pub= 544156099606279493162861023260112477192327569866540314866890827010433986667706
mac_sub= 544156099606279493162861023260112477192327569866540314866890827010433986667706
data = {kekeruhan : 253 suhu : 31.395 ketinggian : 155.139}

mac_pub = 387515353268037051858449727050126655728224429747049901279025637458834618147 55
mac_sub= 109214616411624162248768830656687938968714640636889126052466984610255927119151
data = {kekeruhan : 2550 suhu : 23.611 ketinggian : 177.615}

data diubah
```

Gambar 6.22 Data *Subscribe* Pengujian Penyisipan Data

Gambar 6.22 menunjukkan bahwa data yang diterima sebelum serangan merupakan data yang tidak diubah, karena nilai MAC yang dihitung *subscriber* sama dengan nilai MAC yang didapat dari *publisher*. Sedangkan data kedua yang diterima *subscriber* saat terdapat serangan berupa data yang telah dirubah, sebab nilai MAC yang dihasilkan *subscriber* tidak sesuai. Pengubahan data yang dilakukan oleh *attacker* ditunjukkan pada Gambar 6.23.

```
yogi@Ubuntu:~$ sudo ./ettercaprun3.sh

ettercap 0.8.2 copyright 2001-2015 Ettercap Development Team

Content filters loaded from mqttattacker3.ef...
Listening on:
enp0s8 -> 08:00:27:5C:11:1B
          192.168.1.18/255.255.255.0
          fe80::a00:27ff:fe5c:111b/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/all/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %

=> Inserted char 0 to data sensor
```

Gambar 6.23 Penyisipan Data Pada *Attacker*

Gambar 6.23 menunjukkan proses pengubahan data saat transmisi terjadi menggunakan metode ARP *poisoning* pada Ettercap. Serangan yang dilakukan merupakan skenario pengujian keamanan sistem kedua, yang mengakibatkan *subscriber* menerima data yang sudah dimodifikasi. Berdasarkan Gambar 6.21,

publisher mengirimkan data kekeruhan : 255, suhu : 23,611, dan Ketinggian : 177,615 sedangkan *subscriber* menerima data dengan nilai kekeruhan : 2550, suhu : 23,611, dan Ketinggian : 177,615 yang ditunjukkan pada Gambar 6.22. Namun *subscriber* dapat mengetahui pengubahan ini melalui perbedaan MAC yang dihasilkan. Hal ini memberi kesimpulan lain bahwa pengecekan integritas data telah berhasil dilakukan.

BAB 7 KESIMPULAN DAN SARAN

7.1 Kesimpulan

Berdasarkan masalah yang telah disampaikan serta hasil yang sudah didapat dari hasil penelitian, perancangan, dan pengujian yaitu "Implementasi algoritme SHA-256 menggunakan protokol MQTT pada budidaya ikan hias", maka dibuatlah kesimpulan sebagai berikut :

1. Algoritme SHA-256 yang dirancang pada bubidaya ikan hias menggunakan bahasa pemrograman python3 dan protokol *Message Queue Telemetry Transport* (MQTT) berjenis arsitektur *publish-subscribe* yang mempunyai tiga bagian utama yaitu *topic*, *broker* serta *client* berupa *publisher* dan *consumer* berupa *subscriber*. *Publisher* menggunakan sensor suhu DS18B2O, sensor Ultra Sonik HC-SR04 dan sensor cahaya Photoresistor LDR GL5528 yang digunakan untuk *monitoring* suhu, kedalaman dan transparansi air.
2. *Publisher* melakukan pembacaan dari sensor suhu DS18B2O, sensor Ultra Sonik HC-SR04 dan sensor cahaya Photoresistor LDR GL5528 yang ditambah masukkan *key.* Data yang dikirim oleh *publisher* ke *subscriber* berupa data hasil pembacaan sensor , *key* yang sudah dimasukkan serta MAC hasil dari algoritme SHA-2 yang diperoleh dari data sensor dan *key. Subscriber* melakukan pengecekan integritas data dengan cara mencocokan hasil MAC yang dihasilkan oleh *subscriber* dari data sensor serta *key* yang diterima dan MAC yang dikirim oleh *publisher.* Jika hasil MAC *subscriber* sama dengan hasil MAC *publisher* dapat dipastikan data yang dikirim oleh *publisher* tidak mengalami modifikasi saat data dikrim.
3. Hasil yang didapat dari pengujian pengambilan data dari sensor sebanyak 30 kali tidak ada data yang hilang . Rata – rata data hasil *monitoring* kolam air ikan hias yang ditambah metode integritas data berupa MAC algoritme SHA-256 membutuhkan waktu 604,958 ms pada *publisher* dan 717,344 pada *subscriber*. Peningkatan penggunaan *memory* yang digunakan tanpa dan saat menggunakan MAC algoritme SHA-256 sebesar 0,0078 MB sedangkan pada *subscriber* sebesar 0,0015 MB.

7.2 Saran

Berdasarkan hasil penelitian, perancangan, dan pengujian integritas data menggunakan algoritme SHA-256 , saran yang bisa diberikan oleh peneliti untuk pengembangan sistem selanjutnya adalah :

1. Menggunakan sensor Ph untuk *monitoring* tingkat keasaman air kolam agar memilki parameter yang lebih lengkap mengenai kondisi kolam air ikan hias.
2. Menambahkan metode untuk mengontrol suhu , ketinggian dan tranparansi air kolam ikan hias agar kondisi kolam tetap ideal.

3. Menambahkan data penyimpanan agar data dari sensor dapat dianalisa hasil pembacaan setiap bulan.
4. Menggunakan algoritme enkripsi sebagai keamanan dan kerahasiaan data pada *device* IoT.

DAFTAR PUSTAKA

Aman, M.N., Sikdar, B., K., Chua, C., & Ali, A., 2018. *Low Power Data Integrity in IoT Systems*. IEEE Internet of Things Journal.

Bajaj, S. 2017. *Effect of Environmental Factors on Fish Growth.* Department of Zoology. India

Chiriaco, V., Franzen, A., Thayil, R., Zhang, X. 2017. *Finding Partial Hash Collisions by Brute Force Parallel Programming.* IEEE Long Island Systems, Applications and Technology Conference (LISAT). Farmingdale.NY.

Chiwariro, R., & Rajendran, S., 2018. *Security in Publish / Subscribe Protocol for Internet of Things*. International Journal of Pure and Applied Mathematics.

Datta, S.K., Bonnet, C., & Nikaein , N. 2014. *An IoT Gateway Centric Architecture to Provide Novel M2M Services.* IEEE World Forum on Internet of Things (WF IoT).

Kusumawardhana, P., Ichsan, M., & Primananda, R. 2018. *Implementasi Penyimpanan Data Sensor Nirkabel dengan MongoDB pada Lingkungan IOT Menggunakan Protokol MQTT*. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer.

Hair, J.F., Black, W.C., Babin. B. J. & Andreson R. E., 2010. *Multivariate Data Analysis*. Edisi 7. Upper Saddle River: Prentice Hall

Jameco. 2018. *Raspberry Pi Pinout Diagram and Circuit Notes.* Diambil kembali dari Jameco: www.jameco.com/Jameco/workshop/circuitnotes/raspberry-picircuit-note.html.

Kulkarni, B.P., Joshi, A.V., Jadhav V.V., & Dhamange, A.T., 2017. *IoT Based Home Automation Using Raspberry PI*. International Journal of Innovative Studies in Sciences and Engineering Technology.

Mathews, M.M., & V., Panchami., 2016. *Date Time Keyed - HMAC*. Online International Conference on Green Engineering and Technologies (IC-GET).

Menezes, A. J., Katz, J., van Oorschot, P. C. & Vanstone, S. A., 1996. *Handbook of Applied Cryptography*. s.l.:CRC Press

National Institute of Standards and Technology (NIST).2002. *Secure Hash Standard (SHS).* FIPS PUB 180-2.

Nystrom, M. 2005. *Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA256, HMAC-SHA-384, and HMAC-SHA-512*. RSA Security.

Patchava, V., Kandala, B.H ., & Babu, P.R. 2015. *A Smart Home Automation Technique with Raspberry Piusing IoT.* International Conference on Smart Sensors and Systems (IC-SSS).

Peers, E. 2018. *HC SR04 Ultrasonic Sensor.* Diambil kembali dari Elab Peers: http://www.elabpeers.com/hc-ultrasonic-sensor.html.

Pereira, G.C.C.F., Alves, R.C.A., da Silva, F.L., Azevedo, R.M., Albertini, B.C., & Margi, C.B., 2017. *Performance Evaluation of Cryptographic Algorithms over IoT Platforms and Operating Systems*. Security and Communication Networks.

Potentiallabs. 2018. *DS18B20 Water-Proof Temperature Sensor Probe.* Diambil kembali dari Potentiallabs : https://potentiallabs.com/cart/buy-ds18b20waterproof-online-hyderabad-india.

Ramos, S. H., Benito, M., & Lacuesta, R. 2018. *MQTT Security: A Novel Fuzzing Approach*. Wireless Communications and Mobile Computing.

Ravillaa, Dilli., Putta, Chandra. 2015. *Enhancing the Security of MANETs Using Hash Algorithms*. Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015).

Sebastian, A. 2007. *Implementasi dan perbandingan performa algoritme hash SHA-1, SHA-256 , dan SHA-512.* Program Studi Teknik Informatika. Institut Teknologi Bandung.

Sparkfun. 2018. *Mini Photocell.* Diambil kembali dari Sparkfun: https://www.sparkfun.com/products/9088.