IMPLEMENTASI ALGORITME POLY1305-AES PADA PROTOKOL MQTT

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Tista Pamungkas Ragil Alit
NIM: 135150201111180

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

PENGESAHAN

IMPLEMENTASI ALGORITME POLY1305-AES PADA PROTOKOL MQTT

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Tista Pamungkas Ragil Alit
NIM: 135150201111180

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

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.Sc.	Widhi Yahya, S.Kom, M.Sc.
NIP: 19831283 201803 200 2	NIK: 2016078911211001

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, didalam 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, 02 Januari 2019

Tista Pamungkas Ragil Alit

NIM: 135150201111180

KATA PENGANTAR

Puji dan syukur atas kehadirat Tuhan Yang Maha Esa berkat rahmat dan limpahan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi berjudul "Implementasi Algoritme Poly13015-AES pada Protokol MQTT". Penulisan skripsi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer.

Penulis menyadari bahwa penulisan skripsi ini tidak lepas dari dukungan dan bantuan berbagai pihak. Oleh sebab itu, dalam kesempatan ini penulis ingin menyampaikan rasa hormat dan ucapan terima kasih kepada:

1. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, Bapak Agus Wahyu Widodo, S.T, M.Sc dan Bapak M. Tanzil Furqon, S.Kom, M.ComSc selaku Ketua Jurusan Teknik Informatika, Ketua Program Studi Teknik Informatika dan Sekretaris jurusan Teknik Informatika.
2. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing I yang telah dengan sabar dan tulus meluangkan waktu dan pikiran untuk memberi bimbingan, arahan, serta motivasi dan saran-saran yang berharga kepada penulis selama menyusun skripsi.
3. Bapak Widhi Yahya, S.Kom., M.Sc. selaku dosen pembimbing II yang telah dengan sabar dan tulus meluangkan waktu dan pikiran untuk memberi bimbingan, arahan, serta motivasi dan saran-saran yang berharga kepada penulis selama menyusun skripsi.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga mengharapkan adanya saran serta kritik yang bersifat membangun untuk penelitian selanjutnya. Semoga skripsi ini dapat memberikan manfaat bagi perkembangan ilmu pengetahuan dan semua pihak yang membacanya.

Malang, 02 Januari 2019

Penulis

Tista.pam@gmail.com

ABSTRAK

Tista Pamungkas Ragil Alit, Implementasi Algoritme Poly1305-AES pada Protokol MQTT

Dosen Pembimbing: Ari Kusyanti, S.T, M.Sc dan Widhi Yahya, S.Kom, M.Sc

Message Queue Telemetry Transport (MQTT) merupakan salah satu protokol *Internet of Things* (IoT) yang dirancang khusus untuk komunikasi antar mesin yang memiliki karakteristik dapat bekerja pada *low power*, menggunakan *bandwidth* yang kecil, keandalan dalam pengiriman paket dan protokol ini menggunakan arsitektur *publish-subscribe*. Protokol MQTT hanya menyediakan mekanisme autentikasi untuk keamanannya yang secara *default* tidak menjamin keamanan data dalam transmisinya sehingga privasi data dan integritas data menjadi masalah dalam implementasi protokol. Oleh sebab itu, dilakukan penerapan metode *Message Authentication Code* (MAC) menggunakan algoritme Poly1305-AES yang berbasis *block cipher*. Berdasarkan pengujian, algoritme Poly1305-AES memiliki peningkatan penggunaan *memory* 0,013 MB pada *publisher* dan 0,028 MB pada *subscriber* dan algoritme Poly1305-AES mampu menangani serangan perubahan, penyisipan dan pensubtitusian data. Penelitian ini memberikan hasil bahwa algoritme Poly1305-AES memiliki performa yang cukup bagus berdasarkan nilai peningkatan penggunaan *memory* dan ketahanan terhadap serangan.

Kata kunci: *IoT*, *MQTT, integrity*, *Message Authentication Code, Poly1305-AES*

ABSTRACT

Tista Pamungkas Ragil Alit, Implementasi Algoritme Poly1305-AES pada Protokol MQTT

Dosen Pembimbing: Ari Kusyanti, S.T, M.Sc dan Widhi Yahya, S.Kom, M.Sc

Message Queuing Transport Telemetry (MQTT) is one of the Internet of Things (IoT) protocols specifically designed for communication between machines that have characteristics that can be used at low power, uses small bandwidth, connectivity in packet delivery and protocols using this technology to publish- subscribe. The MQTT protocol only provides authentication security for security which defaults do not guarantee the security of data in transmission so that data privacy and data integrity are problems in the implementation of the protocol. Therefore, the application of the Message Authentication Code (MAC) method uses the Poly1305-AES algorithm based on block ciphers. Based on testing, the Poly1305-AES algorithm has an increase in memory usage of 0.013MB to publishers and 0.028MB to customers and the Poly1305-AES algorithm can support changes, insertions and substitution data. This study presents the results of the Poly1305-AES algorithm which has a pretty good performance based on the value of increasing memory usage and resistance to attacks.

Keyword: IoT, MQTT, integrity, Message Authentication Code, Poly1305-AES

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Internet of Things(IoT) pertama kali diperkenalkan oleh Kevin Ashton pada tahun 1999. Sejak 15 tahun yang lalu diperkenalkan, hingga saat ini belum ada sebuah kesepakatan mengenai definisi *Internet of Things* (IoT). *Internet of Things* (IoT) adalah sistem informasi global yang sangat besar terdiri dari banyak objek yang dapat diidentifikasi, dirasakan dan diproses berdasarkan standar dan interoperabilitas protokol komunikasi (Wangbong Lee, 2016). Pada dasarnya ada dua aristektur yang dapat diterapkan pada jaringan IoT, yaitu *client-server* dan *publish-subscribe.*

Pada saat ini tersedia banyak sekali protokol untuk pengembangan jaringan *Internet of Things* (IoT). Oleh karena itu, harus memilih protokol yang memiliki keandalan dalam pengiriman paket dan penggunaan *bandwidth* yang kecil. Protokol *Message Queue Telemetry Transport* (MQTT) merupakan salah satu protokol yang dirancang khusus untuk komunikasi antar mesin yang memiliki karakteristik dapat bekerja pada *low power*, menggunakan *bandwidth* yang kecil, keandalan dalam pengiriman paket dan protokol ini menggunakan arsitektur *publish-subscribe* (Atmoko, et al., 2017). Protokol MQTT memiliki tiga bagian penting yaitu *topic*, *broker*, serta *client* berupa *publisher* dan *subscriber*. Ketiga bagian tersebut membentuk sebuah sistem dengan tujuan utama yaitu pengumpulan data sensor yang berasal dari *publisher*. Proses pengumpulan data pada arsitektur *publish-subscribe* berdasarkan *topic* atau *event* yang didefinisikan oleh *publisher* dan *subscriber* akan menyatakan ketertarikan atas *topic* atau *event.* Sehingga *topic* pada protokol MQTT bersifat penting karena pada *publish-subscribe* pengumpulan data dikelompokkan berdasarkan *topic.*

Secara umum, terdapat dua masalah utama dari jaringan IoT, yaitu heterogenitas dan keamanan. Heteroginitas yaitu banyaknya ragam perangkat yang dapat berkomunikasi. Keamanan sendiri diperlukan untuk menjaga kerahasian data, integritas data, autentikasi, dan penyangkalan (Mahmoud Elkhodr, 2016). Protokol *Message Queue Telemetry Transport* (MQTT) hanya menyediakan autentikasi untuk mekanisme keamanannya yang secara *default* tidak mengenkripsi data dalam transfer datanya sehingga privasi data dan integritas data menjadi masalah dalam implementasi protokol *Message Queue Telemetry Transport* (MQTT) (Andy, et al., 2017).

Berdasarkan permasalahan umum implementasi jaringan IoT dan penggunaan protokol MQTT, penelitian ini hanya berfokus di permasalahan keamanan di jaringan IoT pada sisi integritas data. Integritas data berhubungan dengan penjagaan perubahan data atau manipulasi data oleh pihak-pihak yang tidak berhak seperti penyisipan, penghapusan, dan pensubtitusian data lain kedalam data sebenarnya. Maka metode pengecekan integritas data sangat dibutuhkan untuk implementasi IoT.

Permasalahan keamanan pada sisi integritas data sangatlah penting diperhatikan. Karena data yang diterima mempengaruhi hasil pemrosesan

lanjutan seperti pengambilan keputusan oleh unit kontrol dan penyimpanan pada basis data. Menurut penelitian yang dilakukan oleh Yindong, Liping, & Ziran yang berjudul *An Approach to Verifying Data Integrity for Cloud Storage,* membahas pendekatan verifikasi integritas data memberikan usulan menggunakan metode kriptografi *message authentication code*(MAC) untuk melakukan pengecekan integritas data dan autentikasi. Metode MAC dapat dirancang dengan dua pendekatan. Pendekatan pertama fungsi *hash* satu arah, sedangkan pendekatan kedua menggunakan *symmetric cipher* berbasis *block cipher* (Munir, 2005).

MAC yang dibuat berdasarkan pendekatan fungsi *hash* seperti HMAC-MD5 atau HMAC-SHA1 telah dianggap tidak aman karena rentan terhadap *collision attack* (Aumasson, et al., 2013). Oleh karena itu, penelitian ini merancang MAC menggunakan *symmetric cipher* berbasis *block cipher* dengan algoritme Poly1305-AES*.* Algortime Poly1305-AES memiliki keamanan yang lebih terjamin, proses komputasi yang lebih cepat, serta kecepatan komputasinya yang tidak hanya terbatas pada pesan berukuran pendek saja (Bernstein, 2005)

Untuk implementasi integritas data pada protokol MQTT, terdapat beberapa aktor, yaitu *publisher*, *broker*, dan *subscriber*. *Publisher* merupakan perangkat sensor yang akan mengirimkan data dari lapangan. *Broker* bertugas untuk menjembatani segala alur data yang terjadi antara *publisher* dan *subscriber*. *Subscriber* bertugas untuk menerima data dari *publisher* dan memasukkan ke dalam basis data. Sedangkan untuk mengimplementasikan fitur keamanan integritas data dan autentikasi pada sistem ini menggunakan algoritme Poly1305-AES yang akan diterapkan pada *publisher* dan *subscriber*. Dengan dibuatnya sistem pengecekan integritas data sensor pada protokol MQTT menggunakan algoritme Poly1305-AES, diharapkan dapat memberi solusi keamanan data, khususnya integritas data yang membutuhkan komputasi cepat dan ringan.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, dapat dirumuskan masalah yang ada yaitu:

1. Bagaimana *subscriber* dapat mengetahui integritas data bedasarkan *Message Authentication Code* (MAC) yang diperoleh dari *publisher* ?
2. Bagaimana peforma algoritme Poly1305-AES saat digunakan untuk pengecekan integritas data pada protokol MQTT?

1.3 Tujuan

Berdasarkan rumusan masalah di atas, dapat disimpulkan tujuan yang ada yaitu:

1. Menerapkan metode *message authentication code* (MAC) sehingga *subscriber* dapat mengetahui integritas data yang diterima berdasarkan *message authentication code* (MAC) yang diperoleh dari *publisher.*
2. Mengetahui kinerja metode pengamanan *message authentication codes* pada Protokol *Message Queue Telemetry Transport* (MQTT).

1.4 Manfaat

Manfaat penelitian ini adalah sebagai berikut:

1. Penelitian ini diharapkan dapat menjadi sebagai acuan serta referensi untuk membangun keamanan suatu jaringan IoT, khususnya dalam kasus menjaga integritas data pada sisi kemanan jaringan.
2. Penelitian ini dapat menjadi wawasan mengenai keamanan data pada jaringan IoT dan juga dapat menjadi acuan untuk mengembangkan dan melanjutkan penelitian yang memiliki keterkaitan dengan penelitian pada skripsi ini.

1.5 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih terfokus, maka penelitian ini dibatasi dalam hal:

1. Menggunakan metode *message authentication codes* dengan algoritme Poly1305-AES dalam pengamanan integritas data.
2. Prokotol komunikasi data yang digunakan pada penelitian ini adalah protokol MQTT.
3. Sistem basis data menggunakan MongoDB.
4. Perangkat IoT (*Node* sensor) menggunakan Raspberry Pi.
5. Proses pengamanan data berada pada *publisher* dan *subscriber.*
6. Tidak ada pertukaran *key* antara *publisher* dan *subscriber.*

1.6 Sistematika Penulisan

Dalam penyusunan skripsi ini, struktur penulisan yang digunakan beserta penjelasannya adalah sebagai berikut:

BAB 1 : Pendahuluan

Menguraikan masalah yang diangkat secara umum dan menjelaskan mengenai struktur penulisan yang digunakan dalam penelitian. Bab ini terdiri dari: latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan.

BAB 2 : Landasan Kepustakaan

Membahas 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.

BAB 3 : Metodologi

Menguraikan mengenai metode serta langkah kerja penelitian yang terdiri atas studi literatur, analisis kebutuhan, perancangan sistem, implementasi, pengujian serta pengambilan kesimpulan dari penelitian yang telah dilakukan.

BAB 4 : **Rekayasa Kebutuhan**
Membahas tentang kebutuhan sistem, seperti kebutuhan *user, fungsional, software* dan *hardware* pada sistem.

BAB 5 : **Perancangan dan Implementasi**
Menjelaskan perancangan sistem dan proses implementasi sesuai dengan rekayasa kebutuhan dan perancangan sistem.

BAB 6 : **Pengujian**
Pengujian dilakukan jika perancangan dan implementasi telah selesai dilakukan. Pengujian bertujuan untuk melakukan uji coba sistem, serta mengetahui kinerja sistem.

BAB 7 : **Penutup**
Memuat kesimpulan dari pembuatan dan pengujian sistem, dan saran untuk pengembangan atau penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Temuan serta teori yang merupakan hasil dari berbagai penelitian yang pernah dilakukan sebelumya, merupakan hal yang penting dan perlu sehingga dapat dijadikan data pendukung. Penelitian dahulu yang relevan menurut peneliti dapat dijadikan bagian tersendiri. Fokus pada penelitian terdahulu yang dijadikan acuan terkait adalah IoT *security* yang sudah ada beserta permasalahan - permasalahannya.

Penelitian pertama yaitu Hybrid Security Techniques for Internet of Things Healthcare Applications yang dilakukan oleh Lobna Yehia, Ayman Khedr, Ashraf Darwish (2015). Penelitian ini memberikan integrasi, merangkum dan survei beberapa teknik keamanan terutama teknik hibrida yang dapat diterapkan dengan aplikasi perawatan kesehatan di lingkungan IoT seperti *Access Control*, *Hashing*, *Steganography*, *Cryptography*, dan *Hybrid Cryptography*. Sifat informasi yang terbuka dalam jaringan IoT akan membawa risiko pada keamanan jaringan dan data yang dikumpulkan. Peneliti memberikan kesimpulan bahwa pada jaringan *IoT* sangatlah penting untuk memberikan sistem keamanan yang lebih karena sifat informasi yang sangat trebuka pada jaringan *IoT.*

Penelitian kedua yaitu Attack Scenarios and Security Analysis of MQTT Communication Protokol in IoT Sistem yang dilakukan oleh Syaiful Andy, Budi Rahardjo, Bagus Hanindhito (2017) yang membahas tentang prtokol MQTT yang hanya menyediakan autentikasi untuk mekanisme keamanan secara default, tidak mengenkripsi data sehingga integritas data menjadi masalah dalam penerapan protokol MQTT pada jaringan IoT. Kemudian penelitian ini juga membahas skenario penyerangan terhadap protokol MQTT pada sisi privasi data, autentikasi, integritas data, ketidak jelasan *port*, dan *botnet over* MQTT. Berdasarkan penelitian sebelumnya peneliti berfokus pada masalah integritas data yang terjadi pada protokol komunikasi MQTT dalam jaringan IoT dengan menggunakan metode *Message Authentication Code* (MAC).

Penelitian ketiga yaitu The Poly1305-AES message-authentication code oleh Daniel J. Bernstein (2005) yang membahas tentang penggunaan algoritme Poly1305-AES dalam pengautentikasian pesan, serta kelebihannya dibandingkan dengan algoritme MAC lainnya, terkait performansi dan variabel pembanding lainnya. Karena penulis menjelaskan kelebihan algoritme poly1305 dibanding algoritme MAC lain dalam sisi keamanan yang lebih terjamin, proses komputasi yang lebih cepat, *cipher* yang dapat diganti dari AES menjadi algoritme kriptografi cipher block lain, serta kecepatan komputasinya yang tidak hanya terbatas pada pesan berukuran pendek saja, menjadi rujukan saya untuk menggunakan algoritme Poly1305-AES.

Perbedaan dari penelitian sebelumnya dengan penelitian yang akan dilakukan akan dirangkum dan dijelaskan pada Tabel 2.1.

Tabel 2.1 Kajian Pustaka

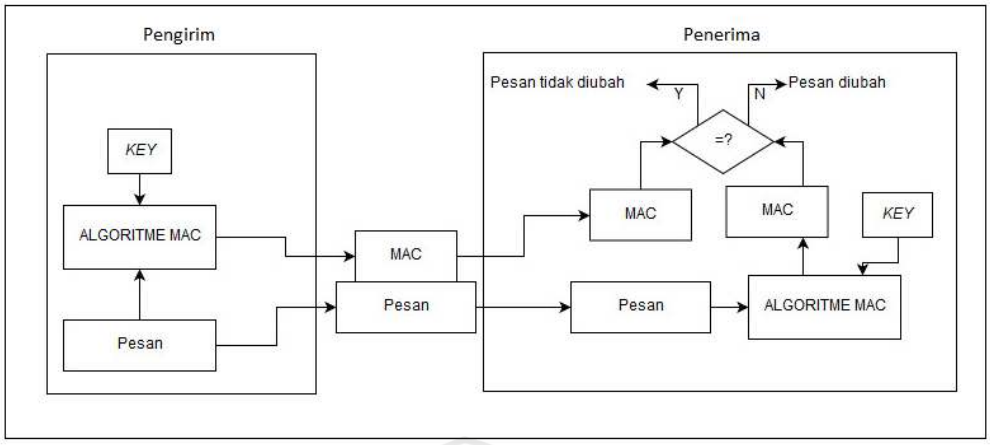
No	Judul Penelitian	Nama Peneliti	Perbedaan	
			Kajian Pustaka	Skripsi Penulis
1	*Hybrid Security Techniques for Internet of Things Healthcare Applications*	Lobna Yehia, Ayman Khedr, Ashraf Darwish (2015)	Penelitian ini merangkum dan survei beberapa teknik keamanan dan menjelaskan sifat informasi yang terbuka dalam jaringan IoT	Penulis menerapkan metode *Message Authentication Code* (MAC)untuk mengatasi masalah keamanan integritas data pada jaringan *IoT*.
2	*Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System*	Syaiful Andy, Bagus Hanindhito, Budi Rahardjo (2017)	Penelitian ini membahas macam-macam jenis skenario serangan yang dapat terjadi pada protokol komunikasi MQTT untuk jaringan IoT.	Penulis menggunakan algoritme Poly1305-AES sebagai solusi permasalahan integritas data dengan metode MAC.
3	*The* Poly1305-AES *message-authentication code*	Daniel J. Bernstein (2005)	Dokumentasi asli dari pembuat algoritme Poly1305-AES yang menjelaskan kelebihan dan cara kerja algoritme Poly1305-AES.	Penulis menerapkankan algoritme Poly1305-AES untuk pengecekan integritas data dengan metode *Message Authentication Code* (MAC) yang dimiliki oleh algoritme Poly1305-AES.

2.2 Message Authentication Code (MAC)

Message Authentication Code (MAC) adalah sebuah tanda pengenal untuk membuktikan integritas suatu dokumen yang didapatkan dengan menggunakan pesan tak bermakna yang diperoleh dari pemrosesan sebagian isi dokumen menggunakan sebuah *private key*. Secara teknis, setengah dokumen diproses menggunakan *private key* sehingga menghasilkan pesan MAC, yang lebih sederhana dari isi dokumen. Pesan MAC ini kemudian dilekatkan dengan dokumen dan dikirim ke penerima. Penerima kemudian menggunakan kunci yang sama untuk memperoleh pesan MAC dari dokumen yang diterima dan membandingkannya dengan pesan MAC yang ia terima.

MAC dapat menjamin dua macam keamanan data integritas dan autentikasi. Integritas dapat terdeteksi jika isi dokumen diubah, karena pesan MAC yang dihasilkan dan diterima akan berbeda. Sedangkan autentikasi terdeteksi dengan penggunaan kunci privatnya. Hanya pengirim dan penerima yang berhak yang mengetahui kunci privat tersebut. Bedanya dengan tanda tangan digital adalah pada kunci ini. MAC menggunakan kunci yang sama (algoritme simetri), sedangkan tanda tangan digital menggunakan kunci privat dan kunci public (algoritme kunci publik).

Algoritme MAC dapat dibangun dengan menggunakan dua cara, yaitu fungsi hash dan algoritme simetri. Jika menggunakan hash, fungsi ini diberlakukan bagi isi dokumen yang telah dilekatkan dengan kunci privat. Sedangkan algoritme simetri yang banyak diimplementasikan dalam MAC sejauh ini adalah dengan mengambil satu blok dari isi dokumen dan dilakukan enkripsi menggunakan algoritme *cipher* blok. Algoritme *cipher* blok bisa juga digunakan untuk membangun sebuah algoritme MAC, yaitu dengan mengambil sebagian dari isi dokumen yang sekiranya unik dan penting (sehingga setiap pesan dapat memiliki pesan MAC yang berbeda). Proses pengiriman pesan, hasil *output* MAC akan dihasilkan dengan masukan pesan, kemudian yang dikirim oleh pengirim adalah pesan beserta MAC. Sedangkan penerima pesan juga akan mencocokkan 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 proses pengiriman pesan seperti yang ditunjukkan pada Gambar 2.1



Gambar 2.1 Cara Kerja MAC

Berdasarkan pada Gambar 2.1 cara kerja MAC membutuhkan *key* dan pesan. *Key* yang digunakan MAC menggunakan simetric *key* yang artinya *key* pengirim sama dengan penerima karena. Pesan masukan akan diproses berdasarkan algoritme MAC yang digunakan. Setelah *output* MAC dihasilkan, selanjutnya data dapat dikirimkan beserta *output* MAC tersebut. Penerima kemudian memproses MAC berdasarkan *key* pengirim dan data yang diterima. Jika *output* MAC yang dihasilkan penerima sama dengan *output* MAC yang diterima dari pengirim, maka data tersebut tidak mengalami pengubahan saat dikirim.

2.3 *Advanced Encryption Standard* (AES)

Advanced Encryption Standard (AES) dibuat oleh Dr. Vincent Rijment dan Dr. Joan Daemen, dipublikasikan oleh NIST (National Institute of Standard and Technology) pada tahun 2001 digunakan untuk menggantikan algoritme *Data Encryption Standard* (DES) yang semakin lama semakin mudah untuk membobol kuncinya. Algoritme ini diperoleh melalui kompetisi yang dilakukan pada tahun 1997.

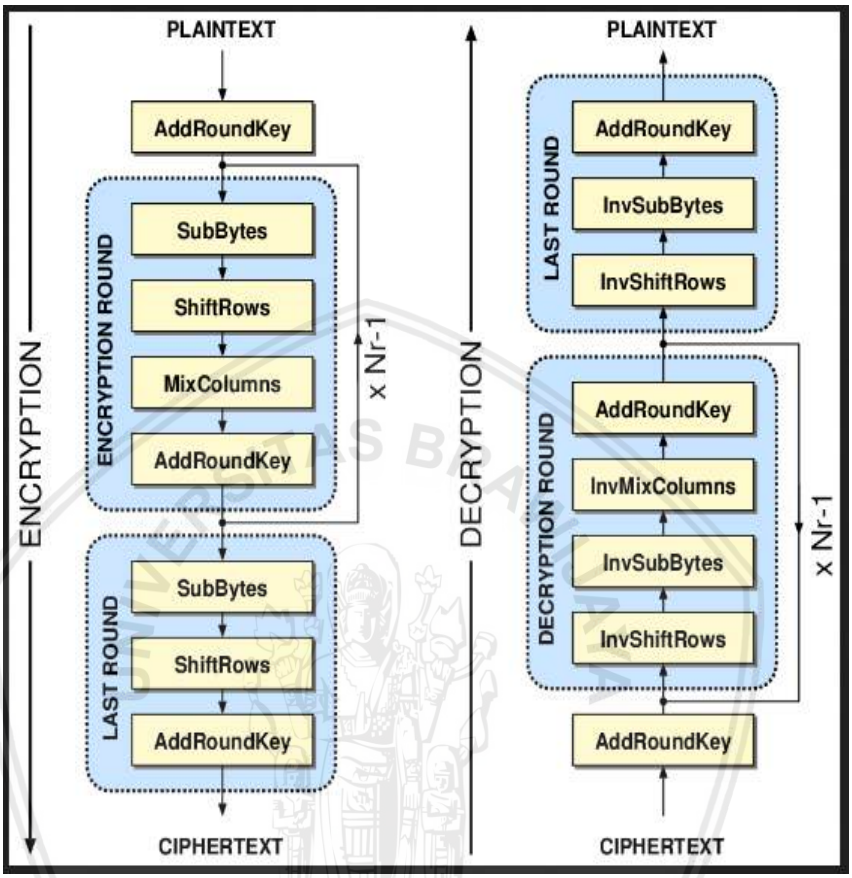
Algoritme AES mengizinkan block data berukuran 128, 168 , 192 , 224 dan 256 bit dan key 128 , 192 dan 256 bit. Standar block data dan key yang digunakan pada metode AES ialah 128 bit. Panjang kunci (*key*) akan mempengaruhi jumlah putaran (*round*) yang diimplementasikan pada algoritme AES.

Tabel 2.2 memperlihatkan jumlah putaran (Nr) berdasarkan varian *key* AES yang harus diimplementasikan pada masing – masing kunci.

Tabel 2.2 Varian AES

Varian AES	Jumlah *Key* (Nk)	Jumlah *Block* (Nb)	Jumlah *Round* (Nr)
AES-128	4	4	10
AES-168	6	4	12
AES-256	8	4	14

Operasi AES dilakukan pada array dua dimensi yang disebut *state* dan *state* yang akan memproses enkripsi dan dekripsi. Proses enkripsi algoritme AES terdiri dari empat jenis transformasi bytes yaitu *AddRoundKey*, *SubBytes*, *ShiftRows* dan *MixColumns*. Penjelasan mengenai proses enkripsi dan dekripsi dapat dilihat pada Gambar 2.2.



Sumber: Sliman A, et al.(2012)

Gambar 2.2 Proses Enkripsi dan Dekripsi AES

Proses pertama enkripsi adalah input yang disalin ke *state* akan mengalami transformasi byte berupa AddRoundKey. Setelah itu, *state* akan ditransformasi byte dengan *SubBytes, ShiftRows, MixColumns* dan *AddRoundKey* secara berulang sebanyak jumlah putarannya. Perlu diketahui pada perulangan terakhir proses enkripsi tidak terjadi transformasi *MixColumns*. Sedangkan proses dekripsi AES merupakan proses *invers* dari proses enkripsi AES.

2.4 Algoritme Poly1305-AES

Poly1305-AES merupakan jenis algoritme MAC yang dipergunakan untuk mengautentikasi integritas pesan dengan menggunakan 32-byte *secret key* yang di-*share* antara *sender* dan *receiver* supaya *receiver* dapat membandingkan nilai *authentication tag* yang telah dihitung oleh pengirim *(sender)* dengan *authentication tag* yang dihitung sendiri oleh penerima *(receiver)*. Poly1305-AES ini menggunakan algoritme kriptografi *cipher block* AES dalam proses enkripsi

pesannya. Sehingga, tingkat keamanannya setara dengan tingkat keamanan AES, dan untuk memecahkan Poly1305-AES ini jalan satu-satunya adalah dengan memecahkan AES (Bernstein, 2005).

Algoritme ini mengkomputasi 128-bit (16-bytes) pesan yang merupakan urutan *byte-byte* dimana tiap *byte* merupakan bilangan *integer* positif antara 0 sampai 255, kunci yang digunakan ialah 32-byte *secret key* yang dishare antara *sender* dan *receiver*, dan terbagi menjadi 2 bagian. 16-byte pertama kunci AES k, dan yang kedua ialah sebuah *string* 16-byte. Bagian kedua tersebut merepresentasikan 128-bit *integer* r dalam format *little endian* dan *nonces* Poly1305-AES membutuhkan 16-byte *nonce* yang bernilai unik, artinya *sender* seharusnya tidak memakai *nonce* yang sama untuk dua buah pesan yang berbeda. Poly1305-AES memasukkan *nonce* n melalui AESk untuk membangkitkan 16-byte string AESk(n). *Pseudocode* algoritme Poly1305-AES akan dijelaskan pada Tabel 2.2.

Tabel 2.3 *Pseudocode* Algoritme Poly1305-AES

No.	*Pseudocode* Algoritme Poly1305-AES
1	**Input** : k (key), r(little endian), n(nonce), msg(pesan)
2	mod1305 = mod2^{130} – 5
3	endianR = little_endian(r)
4	Panjang pesan = (len(msg) + 15) / 16
5	Padding = 0
6	Perulangan i dengan Panjang pesan
7	Partisi pesan msg[i*16 : i*16+16] + b"\x01"
8	Partisi pesan +=(17 - len(Konversi)) * b"\x00"
9	num = endianR(Partisi pesan)
10	Padding (Padding + num) * endianR
11	Padding pesan % mod1305
12	Input AES(k, n)
13	**Output** : hasil (Padding pesan % mod1305 + Input AES(k, n)) % (1 << 128)

2.5 Message Queue Telemetry Transport (MQTT)

MQTT dibuat pertama kali oleh Dr. Andy Stanford-Clark dari IBM, dan Arlan Nipper dari Arcom. MQTT merupakan singkatan dari *Message Queue Telemetry Transport*. Ini merupakan protokol komunikasi *publish/subscribe topic-based* yang sangat sederhana dan ringan, yang didesain untuk alat yang memiliki kemampuan terbatas, *bandwidth* yang rendah, *latency* yang tinggi atau jaringan yang kurang dapat diandalkan. Prinsip dari desain ini adalah untuk meminimalkan penggunaan *bandwidth* jaringan dan kebutuhan sumber daya pada perangkat serta pada waktu yang sama juga berusaha untuk memastikan keandalan dan kepastian dari pengiriman data. Prinsip yang ada ini juga memunculkan beberapa ide protokol mengenai *machine-to-machine* (M2M) atau IoT yang menginginkan perangkat di dunia untuk saling terhubung, dan untuk aplikasi *mobile* dimana *bandwidth* dan daya baterai pada keadaan yang cukup. *Port* TCP/IP 1883 oleh IANA telah

didaftarkan untuk digunakan bersama MQTT. TCP/IP *port* 8883 juga telah didaftarkan untuk penggunaan MQTT melalui SSL. Protokol MQTT memungkinkan untuk mengirimkan *username* dan *password* di dalam paket pada versi V3.1. Enkripsinya dapat diatur dengan SSL, yang dilakukan secara independen oleh protokol MQTT itu sendiri. Pengamanan tambahan dimungkinkan untuk ditambahkan dengan melakukan enkripsi pada data aplikasi yang salaing dikirim dan diterima, namun itu bukan sesuatu yang sudah terbangun dengan sendiri pada protokol, yang bertujuan untuk tetap membuat protokol MQTT menjadi sederhana dan ringan (MQTT, 2016).

Pada penelitian ini, digunakan berbagai *library* pada metode *publish subscribe*. Masing-masing metode beberapa menggunakan *library* yang sama. Selain itu, untuk pemrograman pada *node* sensor dan *subscriber* yang berbasis *python*, maka diperlukan *library* yang bisa digunakan pada *platform* tersebut agar komunikasi data dapat dilakukan. *Paho MQTT Client* adalah *library* yang bertindak sebagai *library* agar file Python yang dibuat dapat menggunakan fitur *MQTT*. *Library* ini berfungsi untuk mendeklarasikan koneksi dari *publisher* ke *broker* ataupun dari *subscriber* ke *broker*, selain itu *library* ini juga berfungsi untuk *publish* pesan ke *broker* (Python Software Foundation, 2017).

2.6 Internet of Things

Internet of Things merujuk pada suatu jaringan yang menghubungkan berbagai benda "*things*" yang dapat diidentifikasi dalam dunia fisik di berbagai jaringan dengan berbagai protokol berbeda (Guoqiang, et al., 2013). IoT memungkinkan suatu objek fisik untuk "mendengar", "melihat", "berpikir" dengan mengharuskan mereka untuk melakukan tugas, yaitu saling "berbicara" bersama objek yang lain, untuk saling berbagi informasi dan menentukan keputusan. IoT merubah objek tersebut dari tradisional menjadi "pintar" dengan memanfaatkan teknologi pokok yang telah dimiliki objek tersebut. IoT memiliki beberapa komponen utama untuk mewujudkan fungsionalitasnya. Komponen tersebut terdiri dari *identification, sensing, communication, computation, service*s, dan *semantics* (Al-Fuqaha, et al., 2015).

Identification penting bagi IoT untuk menamakan dan menyesuaikan layanan dengan permintaan yang ada. Metode pada *Identification* digunakan untuk menyediakan identitas yang jelas bagi setiap objek yang ada pada jaringan. *Sensing* memiliki arti mengumpulkan data dari berbagai objek yang saling terhubung dalam jaringan, kemudian mengirimkannya ke *data warehouse, database*, atau *cloud*. *Communication* menghubungkan berbagai macam objek untuk secara bersama dapat memberikan layanan yang spesifik. Secara umum, objek IoT harus beroperasi dalam keadaan *low* power. *Computation* pada IoT merupakan *processing unit* atau perangkat lunak yang merepresentasikan "otak" dan kemampuan komputasi dari IoT. Pokok utama pada IoT *Service* yaitu melakukan virtualisasi pada data fisik, mengumpulkan data yang diperoleh, mengambil keputusan dari data yang diperoleh, dan menyediakan layanan untuk mengakses data - data tersebut. IoT *Semantics* merupakan kemampuan untuk

mengambil pengetahuan secara tepat oleh berbagai mesin untuk menyediakan layanan yang dibutuhkan (Al-Fuqaha, et al., 2015).

2.7 Python

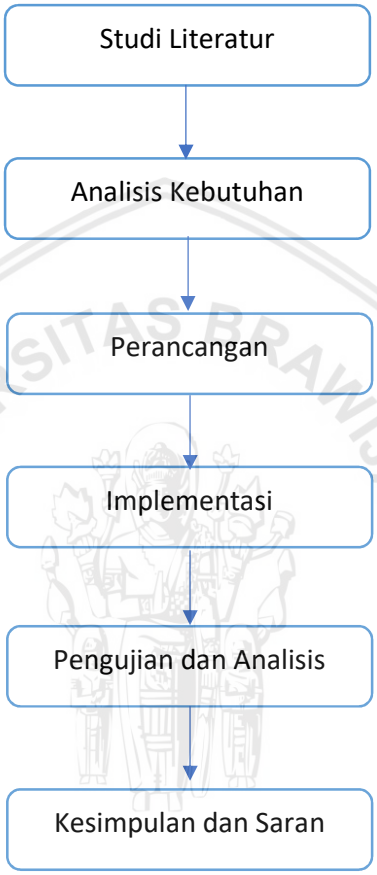
Python merupakan bahasa pemrograman yang bersifat *high level*, interaktif, *object oriented* dan dapat diinterpretasikan. Python adalah bahasa yang jelas serta *powerfull* dibandingkan dengan Perl, Ruby, Scheme, atau Java (Tutorialspoint, 2016). Python didesain untuk sangat mudah dibaca oleh seseorang. Menggunakan bahasa Inggris sebagai kata kunci yang sering digunakan, ditambah dengan bahasa lain yang digunakan sebagai tanda baca, dan python juga merupakan bahasa yang memiliki konstruksi sintaks yang lebih sedikit dari bahasa yang lain. Sama halnya dengan Perl dan PHP, Python dapat diproses secara *runtime* oleh interpreter yang berbentuk seperti *prompt* yang berinteraksi secara langsung untuk menuliskan sebuah program. Python merupakan bahasa yang cocok dipelajari oleh *programmer* awam yang tidak hanya ingin membuat sebuah proses teks yang simpel, namun juga dapat membuat sebuah *web browser* hingga game (Python Software Foundation, 2017).

2.8 MongoDB

MongoDB merupakan *database cross-platform,* berorientasi pada dokumen yang memiliki kinerja yang tinggi, ketersediaan yang tinggi, dan mudah untuk melakukan skalabilitas. MongoDB bekerja dengan konsep koleksi dan dokumen. *Database* merupakan wadah fisik untuk melakukan koleksi oleh MongoDB. Setiap *database* dapat mengatur kumpulan dari *file* yang telah dimiliki pada *file* sistem. Satu *server* mongoDB biasanya memiliki beberapa *database*. Koleksi merupakan kumpulan dari dokumen mongoDB, ini sama halnya dengan sebuah tabel RDBMS. Koleksi ada pada *database* tunggal. Koleksi tidak memaksakan skema. Dokumen yang terdapat dalam koleksi, dapat memiliki bidang yang berbeda, namun umumnya semua dokumen dalam koleksi memiliki tujuan yang sama atau saling terkait. Dokumen adalah satu set pasangan *key* dengan *value*. Dokumen memiliki skema yang dinamis. Skema yang dinamis berati bahwa dokumen dalam koleksi yang sama tidak harus memiliki struktur yang sama, dan pada struktur yang ada pada dokumen memungkinkan untuk melakukan penyimpanan data dalam berbagai tipe (Tutorialspoint, 2016). MongoDB memiliki *library* Koneksi PyMongo yang merupakan *API* yang bekerja pada bahasa pemrograman Python. Distribusi ini memiliki berbagai *tools* yang dapat digunakan untuk bekerja dengan *database* MongoDB, dan direkemondasikan jika bekerja pada *database* MongoDB dengan menggunakan Python (MongoDB, 2015). Dan Mongoengine adlah *API* python berguna untuk object data mapper yang bekerja pada MongoDB.

BAB 3 METODOLOGI

Bab ini menguraikan setiap langkah yang akan ditempuh dalam penyusunan skripsi, diantaranya studi literatur, analisis kebutuhan simulasi, perancangan simulasi, implementasi simulasi, pengujian dan analisis, kesimpulan dan saran. Alur metode penelitian dijelaskan pada Gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori – teori pendukung tersebut meliputi:

1. *Message authentication code*
   Pada bagian ini, dilakukan studi tentang apa itu *message authentication codes.*

2. AES
   Pada bagian ini, dilakukan studi tentang apa itu algoritme AES dan bagaimana cara kerjanya.
3. Poly1305-AES
   Pada bagian ini, dilakukan studi tentang apa itu algoritme Poly1305-AES.
4. Protokol MQTT
   Pada bagian ini, dilakukan studi tentang konsep komunikasi pada protokol MQTT, bagaimana protokol ini bekerja dan bertindak sebagai protokol komunikasi data *publish subscribe*, serta penjelasan *QoS* yang ada pada MQTT.
5. Bahasa Pemrograman Python
   Pada bagian ini, dilakukan studi tentang apa itu bahasa pemrograman python, bagaimana cara kerja bahasa python dan bagaiamana kedudukan bahasa ini terhadap bahasa pemrograman yang lain.
6. MongoDB
   Pada bagian ini, dilakukan studi tentang pengertian MongoDB, apa perbedaan dibandingkan dengan model *database* yang lain, dan bagaimana cara kerjanya.

3.2 Analisis Kebutuhan

Analisis kebutuhan untuk menganalisis beberapa kebutuhan yang diperlukan pada penelitian ini. Analisis kebutuhan pada penelitian ini dijelaskan sebagai berikut:

3.2.1 Kebutuhan Fungsional

Analisis kebutuhan fungsional dilakukan untuk menganalisa kebutuhan yang harus dimiliki oleh sistem. Kebutuhan fungsional sistem merupakan kebutuhan yang akan dicapai sesuai penggunaan algoritme Poly1305-AES sebagai metode pengecekan integritas data sebelum dapat disimpan pada basis data MongoDB. Kebutuhan fungsional terbagi berdasarkan permasalahan yang akan diselesaikan sesuai pada Tabel 3.1.

Tabel 3.1 Kebutuhan Fungsional

Nomor	Kebutuhan
Bagian algoritme	
1	*Publisher* dapat membuat kode MAC Poly1305-AES.
2	*Subscriber* dapat membuat kode MAC Poly1305-AES.
Bagian sistem	
2	*Publisher* dapat dapat terhubung ke *broker* menggunakan *username* dan *password*.
3	*Publisher* dapat mengirim kode MAC Poly1305-AES dan data sensor dalam format JSON.

4	*Subscriber* dapat menerima data JSON dan membacanya.
5	*Subscriber* dapat melakukan pengecekan integritas data berdasarkan kode MAC *publisher.*
6	*Subscriber* dapat menerapkan otorisasi atau *login* ke basis data MongoDB menggunakan *username* dan kata sandi.
7	*Subscriber* dapat menyimpan data sensor yang diterima setelah dipastikan integritasnya ke basis data MongoDB.

3.2.2 Kebutuhan Non-Fungsional

Analisis kebutuhan non fungsional dilakukan untuk menganalisa kebutuhan pendukung dalam melakukan implementasi sistem. Terdapat dua jenis kebutuhan non fungsional yaitu kebutuhan perangkat keras (*hardware*) dan kebutuhan perangkat lunak (*software*).

3.2.2.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras terdiri dari spesifikasi dan informasi perangkat keras yang dibutuhkan untuk melakukan implementasi sistem. Dalam kebutuhan perangkat keras ini dibagi menjadi beberapa komponen dalam sistem. Setiap komponen tersebut memiliki fungsi dan kebutuhan perangkat keras masing-masing .

Pada komponen *Publisher*, perangkat keras yang digunakan berupa Raspberry Pi 3. Perangkat keras ini berfungsi sebagai pembuat dan pengirim data yang berupa nilai suhu. Setelah data nilai sensor di buat maka data nilai sensor tersebut akan dikirimkan ke *broker*. Komponen *broker* menggunakan perangkat keras laptop yang terpasang VirtualBox yang berisi Ubuntu Server OS sebagai *broker* mengirimkan data sensor ke *subscriber*. Sedangkan komponen *subscriber* sama seperti komponen *broker* yang menggunakan laptop yang terpasang VirtualBox yang berisi Ubuntu Server OS sebagai *subscriber* yang akan menerima data dari *broker* dan memproses data sensor kedalam basis data. Berikut gambaran kebutuhan perangkat keras yang dibutuhkan oleh sistem.

Tabel 3.2 Kebutuhan Perangkat Keras

Nama Komponen	Perangkat Keras
Publisher	Raspberry Pi 3
Broker	Laptop
Subscriber	Laptop

3.2.2.2 Kebutuhan Perangkat Lunak

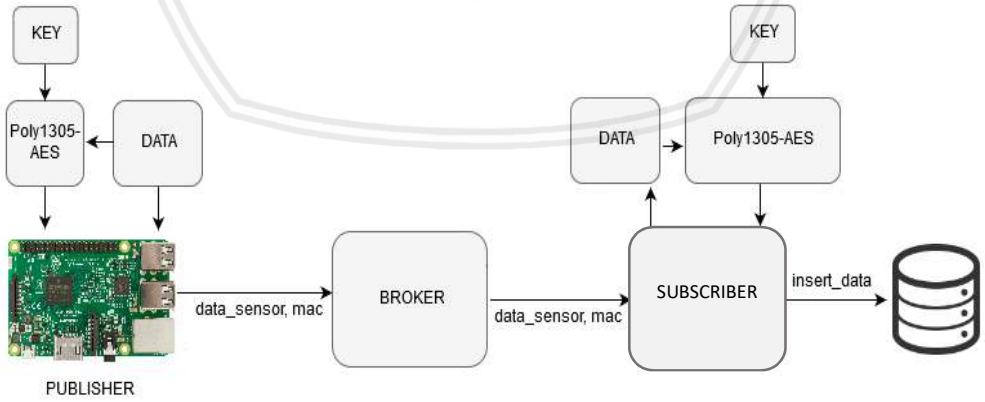
Kebutuhan perangkat lunak terdiri dari spesifikasi dan informasi perangkat lunak yang dibutuhkan untuk melakukan implementasi sistem. Berbagai kebutuhan perangkat lunak tersebut dijelaskan oleh Tabel 3.3.

Tabel 3.3 Kebutuhan Perangkat Lunak

Nama	Versi	Keterangan
Windows 10	1709	Sistem operasi yang digunakan sebagai *host* OS virtualisasi serta digunakan untuk merancang dan mengembangkan sistem
Ubuntu Server	16.04	*Guest OS* yang digunakan pada *broker* maupun *subscriber*
Raspbian	Stretch	Sistem operasi resmi dari Raspberry Pi yang digunakan pada *publisher*
VirtualBox	5.2.8	Perangkat lunak virtualisasi untuk menjalankan *guest OS* Ubuntu Server
MobaXterm	10.5	Perangkat lunak untuk melakukan *remote* SSH terhadap *guest OS* Ubuntu Server
Mosquitto	1.4.12	Perangkat lunak untuk menjalankan *broker* MQTT
Paho MQTT	1.2.0	*Library client* MQTT (*publisher* dan *subscriber*) untuk bahasa pemrograman
MongoDB	3.6.4	Basis data non relasional

3.3 Perancangan

Perancangan dilakukan setelah seluruh kebutuhan dipenuhi melalui tahap rekayasa kebutuhan yang meliputi kebutuhan perangkat lunak dan kebutuhan perangkat keras. Pada penelitian ini kebutuhan utamanya yaitu perangkat lunak untuk membangun jaringan IoT yang terdiri dari *subscribe*r dan *broker*, serta *node* sensor yang berfungsi sebagai *publisher*. Penelitian ini memakai protokol MQTT sebagai protokol komunikasi antar mesin yang berjalan pada sistem.



Gambar 3.1 Gambaran Umum Sistem

3.4 Implementasi

Implementasi dilakukan dengan mengacu pada rekayasa kebutuhan dan perancangan. Implementasi meliputi:

1. Implementasian MQTT *publisher* menggunakan Raspberry Pi 3.
2. Implementasian MQTT *broker* pada *Guest OS* Ubuntu Server.
3. Implementasian MQTT *subscriber* pada *Guest OS* Ubuntu Server.
4. Implementasian algoritme MAC Poly1305-AES baik pada *publisher* maupun *subscriber*.
5. Implementasian basis data MongoDB pada *subscriber.*

3.5 Pengujian dan Analisis

Pengujian sistem dilakukan untuk mengetahui apakah kinerja dan peforma keseluruhan perangkat lunak yang telah dikembangkan sesuai dengan permasalahan yang telah dijelaskan pada bab sebelumnya. Parameter yang digunakan untuk pengujian performa sistem yaitu pengujian waktu eksekusi yang dibutuhkan oleh algoritme Poly1305-AES pada *publisher* maupun *subscriber,* validasi algoritme Poly1305-AES menggunakan *test vector* yang sudah ditetapkan oleh pembuat algoritme Poly1305-AES dan analisis *resource usage* yang dibutuhkan oleh algoritme Poly1305-AES.

3.6 Pengambilan Kesimpulan

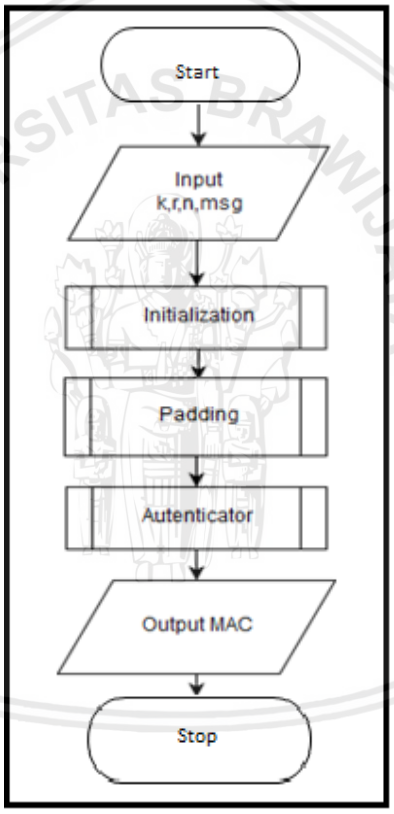
Pengambilan kesimpulan dapat dilakukan jika semua tahapan sebelumnya telah selesai dibangun dan dilakukan. Kesimpulan diambil melalui hasil data pengujian terhadap sistem perangkat lunak yang telah dibangun. Tahap akhir penulisan yaitu berisi saran yang dimaksudkan untuk memberikan pertimbangan mengenai pengembangan dan penelitian lebih lanjut yang mungkin dapat dilakukan.

BAB 4 PERANCANGAN

Pada bab ini akan dibahas mengenai perancangan berdasarkan kebutuhan fungsional dan non-fungsional yang menjadi landasan dalam mengimplementasikan sistem. Tahap perancangan terdiri dari perancangan algoritme, perancangan sistem dan perancangan pengujian.

4.1 Perancangan Algoritme

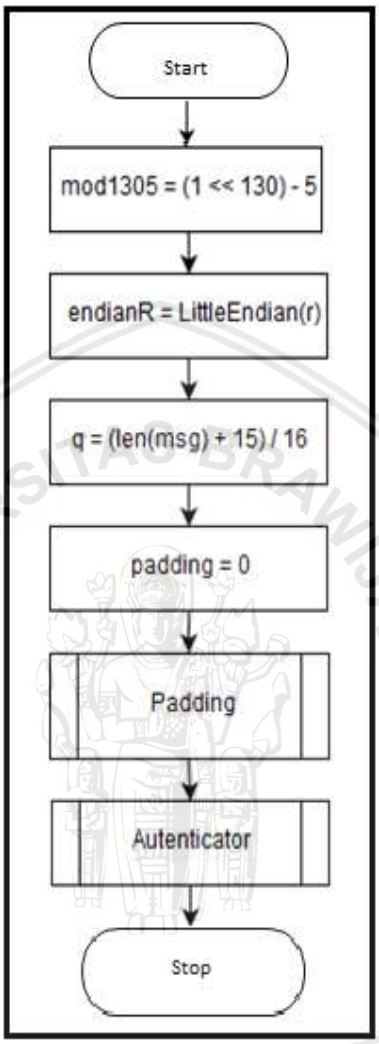
Perancangan komponen algoritme digunakan untuk menjelaskan alur dan cara kerja algoritme Poly1305-AES. Perancangan algoritme Poly1305-AES yang diterapkan pada *publisher* dan *subscriber* akan dijelaskan dengan diagram alir. Algoritme Poly1305-AES diperlukan untuk pengecekan integritas data pada *publisher* dan *subscriber* dengan menghasilkan kode MAC secara garis besar akan ditunjukkan melalui diagram alir Gambar 4.1.



Gambar 4.1 Diagram Alir Poly1305-AES

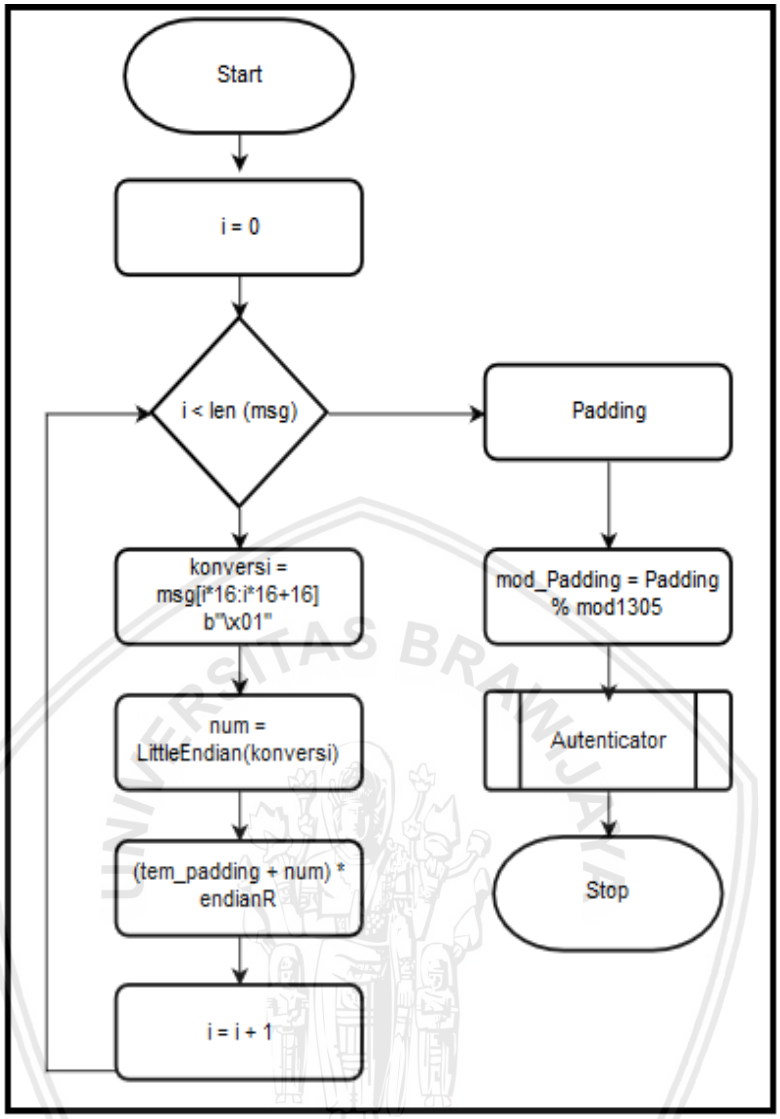
Berdasarkan diagram alir Gambar 4.1, fungsi Poly1305-AES membutuhkan input berupa *key*(k), bilangan *little endian*(r), *nonce*(n) dan pesan(msg). Terdapat empat proses umum yang dijalankan sebelum MAC Poly1305-AES dihasilkan yaitu *Initialization*, *Padding* dan *Autenticator*. Proses *Initialization* akan menginisiasi parameter. Setelah proses *Initialization* terdapat proses *Partition* yang akan membuat pesan terbagi berdasarkan panjang kelipatan pesan untuk mempermudah proses *Padding.* Setelah proses *Partition* terdapat proses *Padding*

untuk menambahan bit-bit dummies sehingga pesan dapat menggenapi panjang blok yang ditentukan. Proses terakhir yaitu *Autenticator* yang berfungsi untuk mengitung hasil *padding* pesan di tambah dengan *key* AES dan *nonce* yang akan dijadikan sebagai hasil *output* MAC. Proses yang dijalankan saat tahap *Initialization* dijelaskan melalui diagram alir proses *Initialization* Gambar 4.2.



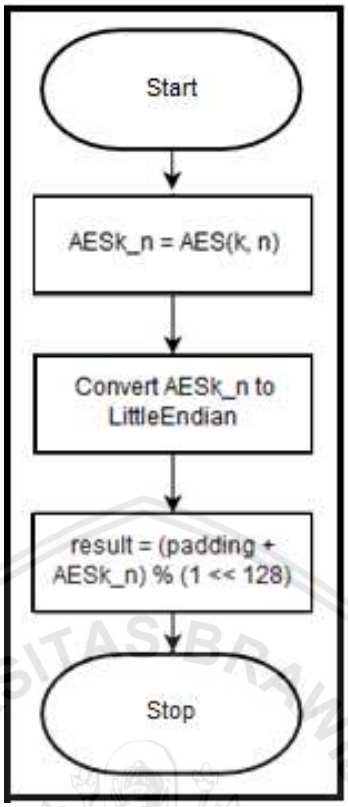
Gambar 4.2 Diagram Alir Proses *Initialization*

Proses *Initialization* pada Gambar 4.2 akan menginisiasi variabel mod1305, endianR, q dan padding yang nantinya akan digunakan untuk proses *Partition, Padding* dan *Autenticator.* Proses *Padding* pesan dijelaskan melalui diagram alir pada Gambar 4.3.



Gambar 4.3 Diagram Alir Proses *Padding*

Proses *Padding* pada Gambar 4.3 akan membuat pesan terbagi berdasarkan panjang kelipatan pesan. Proses *Padding* melakukan perulangan, untuk memecah pesan ke menjadi enambelas bagian dalam bentuk *hexadecimal* dengan penambahan *byte* jika pembagia pesan kurang dari enambelas bagian. Selanjutnya konversi pesan ke bentuk *little endian* dan operasi perkalian antara pesan dengan variabel konstan *little endian* r berdasarkan panjang pesan yang dimasukkan. Setelah melakukan perulangan proses selajutnya adalah proses mod_*Padding* akan melakukan proses operasi perhitungan hasil *padding* pesan *modulo* variabel mod1305. Hasil yang diperoleh dari proses *Padding* akan menjadi acuan pada proses *Auteticator.* Proses *Auteticator* akan dijelaskan pada Gambar 4.5.



Gambar 4.4 Diagram Alir Proses *Autenticator*

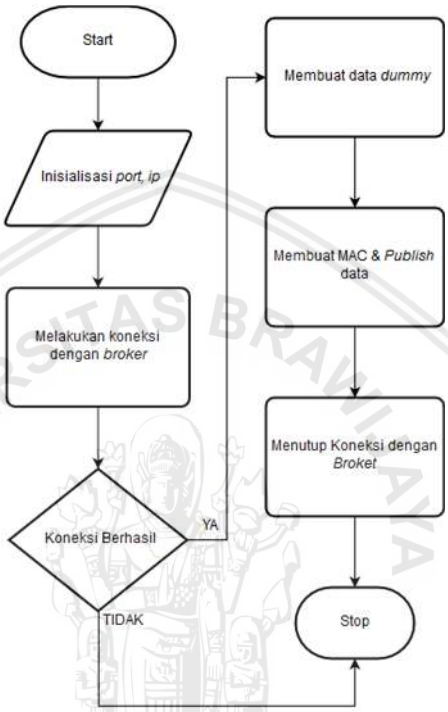
Diagram alir proses *Autenticator* pada Gambar 4.4 merupakan proses akhir yang dijalankan sebelum *output* algoritme Poly1305-AES dihasilkan. Proses ini akan melakukan operasi membuat *key* dan *nonce* pada *library* AES, mengkonversi *key* dan *nonce* ke dalam bentuk *little endian* dan melakukan perhitungan hasil MAC dimana hasil proses *padding* pesan ditambah *key* dan *nonce* setelah itu di *modulo* $mod\ 2^{128}$. Nilai akhir berupa *little-endian* selanjutnya diubah menjadi tipe data *hexadecimal* sebagai hasil *output* algoritme MAC Poly1305-AES.

4.2 Perancangan Sistem

Perancangan sistem menggambarkan alur kerja sistem yang dibuat pada penelitian. Penelitian ini membuat sitem jaringan IoT pada protokol MQTT yang menyediakan fitur untuk mengirim data sensor, menerima data sensor dan pengecekan integritas data sensor menggunakan algoritme Poly1305-AES. Protokol MQTT merupakan protokol yang berbasis arsitektur komunikasi *publish* dan *subscribe* yang terdiri dari tiga komponen yaitu *publisher*, *broker*, dan *subscriber*. Perancangan sistem dibagi berdasarkan komponen yang saling berinteraksi dengan komponen *broker* yaitu komponen *publisher* dan komponen *subscriber*.

4.2.1 Perancangan Komponen *Publisher*

Perancangan komponen *publisher* menggambarkan alur kerja *publisher* saat proses *publish* data sensor. Pada proses *publish* data sensor menuju *broker* memiliki tiga proses yang berurutan. Proses pertama *publisher* membuat koneksi ke *broker* dengan *ip* dan *port* yang sudah ditentukan. Proses selanjutnya adalah *publisher* membuat data sensor yang berupa data *dummy.* Proses terakhir yaitu membuat MAC berdasarkan data sensor yang dibuat dan melakukan *publish* data ke *broker.* Proses-proses yang dijelaskan akan di tunjukan melalui Gambar 4.5.



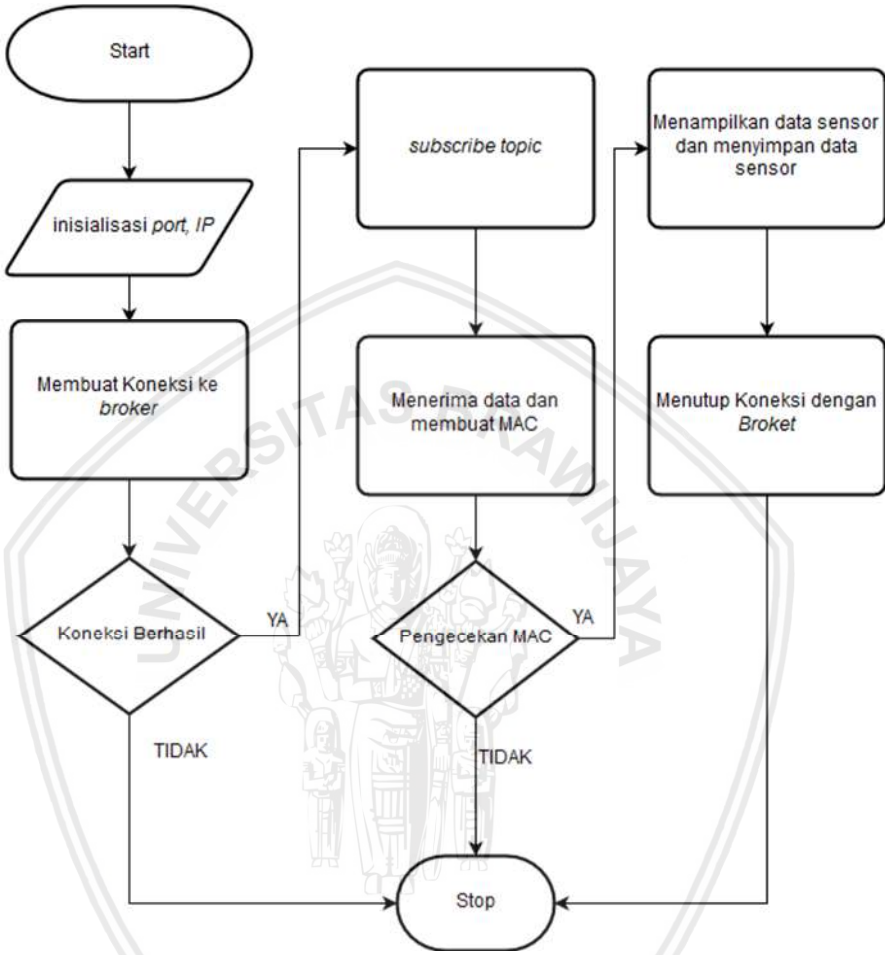
Gambar 4.5 Perancangan Komponen *Publisher*

Proses *Publiser* dimulai dari melakukan inisialisasi *port* dan *IP* dari *broker* protokol MQTT. Jika koneksi diterima oleh *broker* maka proses akan berlanjut pada pembuatan data *dummy* yang digunakan untuk pembuatan MAC. Proses pembuatan MAC dihasilkan dari data *dummy* yang diolah dengan algoritme Poly1305-AES dan setelah pembuatan MAC *publisher* akan mem-*publish* data beserta MAC ke *broker.* Setelah melakukan *publish* data selanjutnya *publisher* akan menutup koneksi ke *broker.* Jika proses koneksi pada *Broker* tidak berhasil, maka proses *publisher* akan berakhir dan aplikasi ditutup.

4.2.2 Perancagan Komponen *Subscriber*

Perancangan komponen *subscriber* menggambarkan alur kerja *subscriber* saat proses *subscribe topic* pada *broker*. Pada proses *subscribe topic* ke *broker* memiliki beberapa proses yang berurutan. Proses pertama *publisher* membuat koneksi ke *broker* dengan *ip* dan *port* yang sudah ditentukan. Proses selanjutnya adalah proses *subscribe topic* sesuai dengan *topic* yang tersedia pada *broker.* Setelah

subscribe topic maka akan menerima data sensor beserta MAC yang selanjutnya akan mengolah data sensor menjadi MAC dengan algoritme Poly1305-AES. Proses selanjutya *subscriber* akan melakukan pengecekan MAC dari *broker* dengan MAC yang dibuat *subscriber* dan akan memasukan data sensor ke basis data MongoDB ketika MAC yang dicocokan sesuai. *Proses-proses yang dijelaskan akan di tunjukan melalui Gambar 4.6.*



Gambar 4.6 Perancangan Komponen *Subscriber*

Berdasarkan Gambar 4.6 perancangan subscriber proses *Subscriber* dimulai dari melakukan inisialisasi *port* dan *IP* dari *broker* protokol MQTT. Jika koneksi diterima oleh *broker* maka proses akan berlanjut pada proses *subscribe topic* yang disediakan *broker* dan ketika koneksi tidak berhasil *subscriber* akan berhenti. Proses selanjutnya diteruskan dengan menerima data sensor berdasarkan *topic* yang telah di-*subscribe* dan *subscriber* akan membuat MAC berdasarkan data yang diterima. Setelah proses pembuatan MAC, *subscriber* akan melakukan pengecekan MAC yang diterima dengan MAC yang dibuat. Jika pengecekan sesuai, *subscriber* akan menampilkan dan memasukan data sensor kedalam basis data MongoDB. Setelah pengecekan MAC *subscriber* akan memutus koneksi ke *broker* dan berhenti. Jika pengecekan MAC tidak sesuai *subscriber* akan menampilkan pesan tidak sesuai dan akan berhenti.

4.3 Perancangan Pengujian

Perancangan pengujian membahas mengenai batasan dan skenario yang digunakan dalam pengujian. Pada penelitian ini, perancangan pengujian meliputi: perancangan pengujian fungsional dan perancangan pengujian performa sistem.

4.3.1 Perancangan Pengujian Fungsional

Perancangan pengujian fungsional berisi skenario untuk memastikan semua fitur telah sesuai dan dapat berjalan dengan benar. Keberhasilan pengujian ini ditentukan oleh kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Skenario pengujian fungsionalitas akan dijelaskan pada Tabel 4.1.

Tabel 4.1 Skenario Pengujian Fungsional

Kode	Fungsi	Skenario
PF_001	Pengujian membuat kode MAC Poly1305-AES pada *Publisher.*	1. *Publisher* mengeksekusi program poly1305.py pada direktori *publisher.*
PF_002	Pengujian membuat kode MAC Poly1305-AES pada *Subscriber*.	1. *Subscriber* mengeksekusi program poly1305.py pada direktori *publisher.*
PF_003	Pengujian *publisher* membangun koneksi dan melakukan *publish* data ke *broker*.	1. *Publisher* mengirim data ke *broker* dengan *payload* yang sudah dibuat.
PF_004	Pengujian *subscriber* membangun koneksi dan menerima data dari *broker.*	1. Membuat koneksi *subscriber* ke *broker* dengan melakukan *subscribe* ke topik yang ada.
PF_005	Pengujian pengecekan integritas data pada *subscriber*.	1. *Subscriber* menerima data *payload* dari broker. 2. *Subscriber* melakukan pencocokan MAC Poly1305-AES dari *publisher* dengan MAC Poly1305-AES yang ada pada *subscriber*.
PF_006	Pengujian penyimpanan data ke dalam basis data MongoDb pada *subscriber*	1. *Publisher* mengirim data ke *broker.* 2. *Subscriber* menerima data *payload* dari *broker*. 3. *Subscriber* melakukan pencocokan MAC Poly1305-AES. 4. *Subscriber* memasukan ke dalam basis data MongoDb.

4.3.2 Perancangan Pengujian *Test Vector*

Perancangan Pengujian *Test Vector* merupakan skenario pengujian *Test vector* untuk mengetahui validitas kode program Poly1305-AES yang diterapkan pada sistem. Pengujian algoritme Poly1305-AES menggunakan *test vector* yang telah disediakan oleh pembuat algoritme Poly1305-AES. Skenario yang diterapkan adalah membandingkan *output* yang dihasilkan oleh kode program algoritme Poly1305-AES yang diterapkan dengan *output* data *test vector.*

4.3.3 Perancangan Pengujian Waktu Eksekusi Algoritme

Perancangan pengujian waktu eksekusi algoritme berisi skenario pengujian untuk mengetahui waktu yang dibutuhkan oleh algoritme poly1305-AES setiap pembuatan MAC. Berikut ini adalah skenario pengujian peforma sistem:

1. Menjalankan kode program poly1305-AES yang telah diimplementasikan pada *publisher* dan *subscriber* sebanyak 30 kali dan 10 kali iterasi. Penghitungan waktu eksekusi akan dihitung ketika data dan *key* diterima sebagai *input* algoritme sampai pemrosesan *output* berupa kode MAC.

4.3.4 Perancangan Pengujian Peforma Sistem

Perancangan pengujian performa sistem berisi skenario pengujian untuk menguji performa dari sistem. Pengujian ini berkaitan dengan kinerja sistem dalam pemrosesan algoritme berdasarkan penggunaan *memory* dan kinerja jaringan pada protokol MQTT. Skenario pengujian peforma sistem akan dijelaskan pada Tabel 4.2.

Tabel 4.2 Skenario Pengujian Peforma Sistem

Kode	Pengujian	Skenario
PS_01	Pengujian penggunaan *memory* pada sistem.	Pengujian akan dilakukan dengan membandingkan penggunaan *memory* sistem, saat menggunakan algoritme dan tidak menggunakan algoritme. Dengan mengambil sampel data 30 kali *publish-subscribe* dengan 10 kali iterasi.
PS _02	Pengujian waktu pengecekan integritas data.	Pengujian akan dilakukan dengan menggunakan GNU Parallel. Sampel data diambil dari pengujian dengan 30, 60, 90, 120 dan 150 *client* dengan masing-masing melakukan 30 kali *publish-subscribe.*
PS _03	Pengujian *memory* pengecekan integritas data.	Pengujian akan dilakukan dengan menggunakan GNU Parallel. Sampel data diambil dari pengujian dengan 30, 60, 90, 120 dan 150 *client* dengan masing-masing melakukan 30 kali *publish-subscribe.*

4.3.5 Perancangan Pengujian Keamanan

Perancangan pengujian keamanan berisi skenario untuk memastikan keamanan sistem yang dibuat. Pada pengujian ini peneliti menggunakan serangan *Man in The Middle Attack* (MITM) ke dalam sistem dengan metode ARP Spoofing yang memungkinkan untuk melakukan perubahan data, penyisipan data dan pensubtitusian data. Untuk melakukan serangan ARP Spoofing peneliti memakai aplikasi Ettercap. Ada beberapa sekenario penyerangan untuk melakukan pengujian yaitu:

Tabel 4.3 Skenario Pengujian Keamanan

Kode	Pengujian	Skenario
PK_01	Penyerangan perubahan data.	Pengujian dilakukan dengan melakukan penyerangan terhadap *broker* untuk merubah data sensor yang di-*publish* oleh *publisher* menggunakan aplikasi Ettercap.
PK_02	Penyerangan penyisipan data. .	Pengujian dilakukan dengan melakukan penyerangan terhadap *broker* untuk menyisipkan data kedalam data sensor yang di-*publish* oleh *publisher* menggunakan aplikasi Ettercap.
PK _03	Penyerangan pensubtitusian data.	Pengujian dilakukan dengan melakukan penyerangan terhadap *broker* untuk mensubtitusi data sensor yang di-*publish* oleh *publisher* menggunakan aplikasi Ettercap.

BAB 5 IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi pengecekan integritas data pada potokol MQTT dengan mengacu pada bab analisis kebutuhan dan perancangan. Pada penelitian ini, implementasi meliputi: implementasi komponen komunikasi dan implementasi komponen keamanan. Proses implementasi dilakukan pada mesin virtual dengan sistem operasi Ubuntu *Server* versi 16.04.

5.1 Implementasi Algoritme

implementasi pengecekan integritas data berisi kode algoritme Poly1305-AES yang digunakan untuk pengecekan integritas data, berikut ini adalah tabel kode program algoritme Poly1305-AES dan tabel kode program AES :

Tabel 5.1 Kode Program Poly1305-AES

No.	Kode Program Poly1305-AES
01	`import binascii`
02	`import sys`
03	`import os`
04	`import aesku`
05	
06	`def aes_encrypts(key, input_bytes):`
07	`    return aesku.encrypt(key, input_bytes)`
08	`#Klas utama Poly1305-AES`
09	`class Poly1305:`
10	`    global bytes_to_string`
11	`    def __init__(self, key_aes, r, nonce , string=b"",`
12	`method=aes_encrypts):`
13	`        self.key_aes = key_aes`
14	`        self.r = r`
15	`        self.nonce = nonce`
16	`        self.string = string`
17	`        self.aes = method`
18	
19	`    def update(self, msg):`
20	`        self.string += msg`
21	`    #Fungsi konversi byte ke string`
22	`    def bytes_to_string(binary):`
23	`        return "".join(chr(b) for b in binary)`
24	`    #Fungsi konversi byte,string ke integer`
25	`    def LittleEndian(self, val):`
26	`        return int(binascii.hexlify(val[::-1]), 16)`
27	
28	`    def algo(self):`
29	`        k, r, n, msg = (self.key_aes, self.r, self.nonce,`
30	`self.string)`
31	`        mod1305 = (1 << 130) - 5`
32	`        endianR = self.LittleEndian(r)`
33	`        #pesan + 15 agar tidak 0`
34	`        panjang_msg = (len(msg) + 15) / 16`
35	`        padding = 0`
36	`        for i in range(int(panjang_msg)):`
37	`            konversi = msg[i*16 : i*16+16] + b"\x01"`
38	`            print ("len :",len(konversi))`
39	`            konversi += (17 - len(konversi)) * b"\x00"`
40	`            num = self.LittleEndian(konversi)`
41	`            padding = (padding + num) * endianR`

```
42          mod_padding = padding % mod1305
43          AESk_n = bytes_to_string(self.aes(k, n))
44          AESk_n = self.LittleEndian(AESk_n)
45          hasil_endian = (mod_padding + AESk_n) % (1 << 128)
46          # Konversi hasil(endian) hexa
47          hasil_mac = ''.join(map(lambda i: chr(0xff & (hasil_endian
48  >> 8*i)), range(16)))
49          return hasil_mac
50
51  #Klas PolyRun
52  class PolyRun:
53      global string_to_bytes
54      def string_to_bytes(text):
55          return list(ord(c) for c in text)
56      #fungsi untuk instansiasi memanggil klas utama
57      def poly1305aes(self, k, r, n, m):
58          return Poly1305(k, r, n, m, aes_encrypts).algo()
59      #fungsi untuk menghasilkan MAC
60      def runMAC(self, pesan):
61          mac = self.poly1305aes(b"TistaPamungkasRA", b" "*16,
62  string_to_bytes(" ")*16, pesan)
63          res_mac = binascii.b2a_hex(mac).decode('utf-8')
64          return res_mac
65
```

Penjelasan tabel kode Poly1305-AES :

1. Baris 01-04, memuat modul-modul yang diperlukan.
2. Baris 06-07, deklarasi fungsi `aes_encrypt` untuk mendapatkan *key* AES.
3. Baris 09, deklarasi kelas `Poly1305` merupakan kelas utama untuk membuat algoritme Poly1305-AES.
4. Baris 11-17, deklarasi fungsi __init__ sebagai inisialisasi variable inputan.
5. Baris 19-20, dekalarasi fungsi update untuk inisialisasi `self.__string += msg`.
6. Baris 21-23, deklarasi fungsi bytes_to_string untuk konversi *bytes* menjadi *string*.
7. Baris 24-25, deklarasi fungsi `LittleEndian` untuk membuat konversi inputan bilangan *litleendian*.
8. Baris 28-49, deklarasi fungsi `algo` sebagai fungsi utama untuk implementasi algoritme poly1304-aes terdapat inputan dan logika perulangan sesuai dengan perancangan algoritme.
9. Baris 51-64, deklarasi kelas `PolyRun` untuk menjalankan fungsi pada kelas `Poly1305` denagn *key* dan masukan pesan yang ditentukan.

Tabel 5.2 Kode Program AES

No.	Kode Program AES
01	`sbox = [`
02	`    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,`
03	`0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,`
04	`    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,`
05	`0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,`
06	`    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,`
07	`0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,`
08	`    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,`
09	`0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,`
10	

```
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
]

rcon = [[0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36],
        [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00],
        [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00],
        [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00]
]

nb = 4  # jumlah nb dalam tiap state
nr = 10  # jumlah round
nk = 4  # panjang key (4x32bit)

#fungsi untuk melakukan subtitusi bytes
def sub_bytes(state):
    box = sbox
    for i in range(len(state)):
        for j in range(len(state[i])):
            row = state[i][j] // 0x10
            col = state[i][j] % 0x10
            box_elem = box[16 * row + col]
            state[i][j] = box_elem
    return state

#fungsi untuk melakukan shift row/mengeser bytes
def shift_rows(state):
    count = 1
    for i in range(1, nb):
        state[i] = left_shift(state[i], count)
        count += 1
    return state

#fungsi untuk melakukan mix kolom
def mix_columns(state):
    for i in range(nb):
        s0 = mul_by_02(state[0][i]) ^ mul_by_03(state[1][i]) ^
state[2][i] ^ state[3][i]

```

```
100         s1 = state[0][i] ^ mul_by_02(state[1][i]) ^
101 mul_by_03(state[2][i]) ^ state[3][i]
102         s2 = state[0][i] ^ state[1][i] ^ mul_by_02(state[2][i]) ^
    mul_by_03(state[3][i])
103         s3 = mul_by_03(state[0][i]) ^ state[1][i] ^ state[2][i] ^
104 mul_by_02(state[3][i])
105 
106         state[0][i] = s0
107         state[1][i] = s1
108         state[2][i] = s2
109         state[3][i] = s3
110 
111     return state
112 
113 #fungsi untuk ekpansi kunci
114 def key_expansion(key):
115     # translasi key string kedalam array list
116     key_symbols = [ord(symbol) for symbol in key]
117     # membuat chiper key dengan key schedule
118     key_schedule = [[] for i in range(4)]
119     for r in range(4):
120         for c in range(nk):
121             key_schedule[r].append(key_symbols[r + 4 * c])
122 
123     # mengisi key schedule
124     for col in range(nk, nb * (nr + 1)):
125         if col % nk == 0:
126             # vriabel temporary untuk key tiap round
127             tmp = [key_schedule[row][col - 1] for row in range(1,
128 4)]
129             tmp.append(key_schedule[0][col - 1])
130 
131             # subtitusi bytes pada temporary key
132             for j in range(len(tmp)):
133                 sbox_row = tmp[j] // 0x10
134                 sbox_col = tmp[j] % 0x10
135                 sbox_elem = sbox[16 * sbox_row + sbox_col]
136                 tmp[j] = sbox_elem
137 
138             # melakukan xor untuk nb
139             for row in range(4):
140                 s = (key_schedule[row][col - 4]) ^ (tmp[row]) ^
141 (rcon[row][int(col / nk - 1)])
142                 key_schedule[row].append(s)
143 
144         else:
145             for row in range(4):
146                 s = key_schedule[row][col - 4] ^
147 key_schedule[row][col - 1]
148                 key_schedule[row].append(s)
149 
150     return key_schedule
151 
152 #fungsi untuk menambahkan kunci berdasrkn key expansion
153 def add_nr_key(state, key_schedule, round=0):
154     for col in range(nk):
155         s0 = state[0][col] ^ key_schedule[0][nb * nr + col]
156         s1 = state[1][col] ^ key_schedule[1][nb * nr + col]
157         s2 = state[2][col] ^ key_schedule[2][nb * nr + col]
158         s3 = state[3][col] ^ key_schedule[3][nb * nr + col]
159 
160         state[0][col] = s0
161         state[1][col] = s1
162         state[2][col] = s2
163         state[3][col] = s3
```

```
164
165     return state
166 #fungsi untuk mengeser baris bytes
167 def left_shift(array, count):
168     res = array[:]
169     for i in range(count):
170         temp = res[1:]
171         temp.append(res[0])
172         res[:] = temp[:]
173     return res
174
175 #fungsi untuk perkalian matrik dengan 02
176 def mul_by_02(num):
177     if num < 0x80:
178         res = (num << 1)
179     else:
180         res = (num << 1) ^ 0x1b
181     return res % 0x100
182
183 #fungsi untuk perkalian matrik dengan 03
184 def mul_by_03(num):
185     return (mul_by_02(num) ^ num)
186
187 def encrypt(key, input_bytes):
188     state = [[] for j in range(4)]
189     for r in range(4):
190         for c in range(nb):
191             state[r].append(input_bytes[r + 4 * c])
192
193     key_schedule = key_expansion(key)
194
195     state = add_nr_key(state, key_schedule)
196
197     for rnd in range(1, nr):
198         state = sub_bytes(state)
199         state = shift_rows(state)
200         state = mix_columns(state)
201         state = add_nr_key(state, key_schedule, rnd)
202
203     state = sub_bytes(state)
204     state = shift_rows(state)
205     state = add_nr_key(state, key_schedule, rnd + 1)
206
207     result = [None for i in range(4 * nb)]
208     for r in range(4):
209         for c in range(nb):
210             result[r + 4 * c] = state[r][c]
211     return result
```

Penjelasan tabel 5.2 kode program AES:

1. Baris 01-60, deklarasi variabel sbox berisi *array* subtitusi *box* algoritme AES.
2. Baris 61-70, deklarasi variabel rcon berisi *array* untuk pembuatan kunci AES.
3. Baris 71, deklarasi variabel nb untuk menentukan *state*.
4. Baris 72, deklarasi variabel nr untuk menentukan *round*.
5. Baris 73, deklarasi variabel nk untuk menentukan panjang *key*.
6. Baris 75-84, deklarasi fungsi sub_bytes untuk melakukan *subtitusi bytes*.
7. Baris 86-92, deklarasi fungsi shift_row untuk melakukan *shift row* atau mengeser *bytes*.

8. Baris 94-111, deklarasi fungsi mix_columns untuk melakukan *mix* kolom.
9. Baris 113-150, deklarasi fungsi key_expansion untuk melakukan ekpansi kunci AES.

5.2 Implementasi Sistem

Implementasi komponen komunikasi membahas mengenai instalasi dan konfigurasi yang diperlukan untuk menjalankan protokol MQTT. Pada penelitian ini protokol MQTT dikembangkan menggunakan bahasa pemrograman Python. Pembahasan pada sub-bab ini meliputi konfigurasi *publisher*, konfigurasi *broker*, dan konfigurasi *subscriber*.

5.2.1 Implementasi Komponen *Publisher*

Pada implementasi *publisher* pada *node* sensor (Raspberry Pi), perlu dilakukan pertama kali yaitu melakukan *update* dan *upgrade* untuk mendapatkan aplikasi versi terbaru. Aplikasi Python dan library paho-mqtt juga dibutuhkan oleh *publisher* sehingga dapat melakukan *publish* ke *broker*, langkah yang dibutuhkan yaitu menjalankan perintah akan dijelaskan pada Tabel 5.3.

Tabel 5.3 Perintah Instalasi MQTT *Client*

No.	Perintah Instalasi MQTT *Client*
1	$ sudo apt-get update
2	$ sudo apt-get upgrade
3	$ sudo apt-get install python
4	$ sudo apt-get install python-pip
5	$ sudo pip install paho-mqtt

Setelah berhasil melakukan instalasi aplikasi yang diperlukan, maka perlu membuat kode yang nantinya akan mengirimkan data ke *broker*, berikut ini adalah tabel 5.4 kode program *publisher*:

Tabel 5.4 Kode Program *Publisher*

NO.	Kode Program *Publisher*
1	import json
2	#Import library paho-mqtt
3	import paho.mqtt.client as mqtt
4	import random
5	dummy = ['55', '60', '65', '70', '75','80','85','90','95','100']
6	data = random.choice(dummy)
7	
8	#import MAC
9	from poly1305 import PolyRun
10	mac_pub = PolyRun().runMAC(b"80")
11	
12	# Inisiasi object mqtt
13	mqttc = mqtt.Client("publisher", clean_session=True)
14	
15	# Buat koneksi ke broker
16	mqttc.connect("192.168.56.101", 1883)
17	
18	#data dictionary
19	dictionary_pub = {
20	"mac" : mac_pub,
21	"sensor" : data}
22	#inisialisasi data dump
23	json_pub = json.dumps(dictionary_pub)

```
24      # Publish
25      mqttc.publish("/node/suhu", payload=json_pub, qos=1)
```

Penjelasan tabel 5.4 kode program *publisher*:

1. Baris 1-4, memuat modul yang diperlukan.
2. Baris 5, deklarasi variable `dummy` untuk membuat daftar inputan acak data sensor.
3. Baris 6, deklarasi variable `data` untuk memilih data sensor berdasarkan daftar data.
4. Baris 8-10, untuk menjalankan kelas PolyRun dari file Poly1305 agar mendapatkan output dari algoritme Poly1305-AES yang nantinya digunakan sebagai pengecekan integritas data.
5. Baris 12-13, inisialisasi objek mqttc sebagi mqtt client.
6. Baris 15-17, membuat koneksi ke *broker* denagn username dan password yang sudah terdaftar dan menagatur alamat ip dan port yang digunakan.
7. Baris 18-22, membuat kamus data untuk dikirimkan ke *subscriber* melewati *broker*.
8. Baris 23-24, inisialisasi data dump atau kamus data yang dibuat.
9. Baris 25-26, mempublish atau mengirimkan data dump yang dibuat dengan topik tertentu.

5.2.2 Implementasi *Broker*

Pada implementasi *Broker*, perlu dilakukan pertama kali yaitu melakukan update dan upgrade untuk mendapatkan aplikasi versi terbaru. Aplikasi mosquitto dibutuhkan oleh *broker* sehingga dapat menerima *publish* dan *subscribe* dari *publisher* dan *subscriber*, langkah yang dibutuhkan yaitu menjalankan perintah `sudo apt-get update`, `sudo apt-get upgrade` dan `sudo apt-get install mosquito`.

5.2.3 Implemntasi Komponen *Subscriber*

Pada implementasi *subscriber*, perlu dilakukan pertama kali yaitu melakukan update dan upgrade untuk mendapatkan aplikasi versi terbaru. Aplikasi Python dan beberapa *library* pendukung juga dibutuhkan oleh *subscriber* sehingga dapat menerima data dari *broker* dan memasukkan data tersebut ke database, langkah yang dibutuhkan yaitu menjalankan perintah `sudo apt-get install python-pip`, `sudo pip install paho-mqtt` dan `sudo pip install pymongo`. Setelah berhasil melakukan instalasi aplikasi yang diperlukan, maka perlu membuat *script* yang nantinya akan melakukan *subscribe* ke *broker*, berikut ini adalah Tabel 5.5 kode program *subscriber* :

Tabel 5.5 Kode Program *Subscriber*

```
1   import json
2   # Import library paho-mqtt
3   import paho.mqtt.client as mqtt
4
5   #import MAC
6   from poly1305 import PolyRun
7   # Inisiasi object mqtt
8   mqttc = mqtt.Client("subscriber1", clean_session=True)
```

```
9
10  # Buat koneksi ke broker
11  mqttc.connect( "192.168.1.19", 1883 )
12  mqttc.username_pw_set("subscriber","subscriber")
13
14  # Konek MongoDB pada database NodeData
15  from pymongo import MongoClient
16  client = MongoClient("localhost", 27017)
17  db = client.DataSensor
18  db.authenticate('dbAdmin', 'warning1', source='admin')
19  def on_connect(mqttc, obj, flags, rc):
20          print("Connected to Broker")
21
22  def on_message(mqttc, obj, msg):
23          data_str = json.loads(msg.payload)
24          mac_sub = PolyRun().runMAC(b"80")
25          mac_pub = data_str["mac"]
26          print "Genarate MAC Poly1305-aes Subscriber : ", mac_sub
27          print "======================================"
28          if (mac_sub == mac_pub) :
29                  data_sensor = data_str["sensor"]
30                  mac_pub = data_str["mac"]
31                  db.sensor.insert_one({"sensor" : data_sensor})
32                  print "Generate MAC Poly1305-aes Publisher : ",
33  mac_pub
34                  print  "Topik  "+msg.topic+"  Payload  :  "+
35  data_sensor + " QoS "+str(msg.qos)
36                  print "======================================"
37                  print "Subsriber insert data to mongoDb"
38  # Set callback function
39  mqttc.on_connect = on_connect
40  mqttc.on_message = on_message
41  # Subscribe ke topik tertentu
42  mqttc.subscribe("/node/suhu", qos=1
43  # Loop supaya subscribernya tidak berhenti
44  mqttc.loop_forever()
```

Penjelasan tabel 5.4 kode program *subscriber* :

1. Baris 1-3, memuat modul yang diperlukan.
2. Baris 5-6, import untuk menjalankan kelas PolyRun dari file Poly1305 agar mendapatkan output dari algoritme Poly1305-AES yang nantinya digunakan sebagai pengecekan integritas data.
3. Baris 8-9, inisialisasi objek mqttc sebagi mqtt client.
4. Baris 11-13, membuat koneksi ke *broker* dengan username dan password yang sudah terdaftar dan menagatur alamat ip dan port yang digunakan.
5. Baris 15-19, membuat koneksi ke basis data MongoDB agar dapat memasukan data sensor yang dikirim kedalam basis data.
6. Baris 21-22, deklarasi fungsi on_connect sebagai penanda jika *subscriber* sudah terhubung dengan *broker*.
7. Baris 24-38, deklarasi fungsi on_message yang berfungsi untuk menerima pesan atau payload dari *broker*, pengecekan algoritme Poly1305-AES dan memasukan pesan atau payload kedalam basis data MongoDB yang dibuat.
8. Baris 39-41, inisialisasi untuk memanggil fungsi dengan variable.
9. Baris 43-44, untuk *subscribe* topik yang tersedia.
10. Baris 46-48, perulangan agar *subscriber* selau menyala.

5.2.4 Implementasi Pengecekan Integritas Data

implementasi pengecekan integritas data berisi potongan kode untuk memanggil algoritme Poly1305-AES yang berada pada *publisher* dan *subscriber* untuk pengecekan integritas data, berikut ini adalah potongan kode program algoritme Poly1305-AES dalam Tabel 5.6 integritas data *publisher* dan tabel 5.6 integritas data *subscriber* :

Tabel 5.6 Integritas Data *Publisher*

No.	Kode Integritas Data *Publisher*
1	from poly1305 import PolyRun
2	mac_pub = PolyRun().runMAC(b"80")
3	dictionary_pub = {"mac" : mac_pub,
4	"sensor" : data}
5	json_pub = json.dumps(dictionary_pub)

Penjelasan Tabel 5.5 integritas data *publisher* :

1. Baris 1-2, untuk menjalankan kelas PolyRun dari file Poly1305-AES agar mendapatkan *output* dari algoritme Poly1305-AES yang nantinya digunakan sebagai pengecekan integritas data.
2. Baris 3-4, membuat kamus data untuk dikirimkan ke *subscriber* melewati *broker*.
3. Baris 5, inisialisasi data dump atau kamus data yang dibuat.

Tabel 5.7 Integritas Data *Subscriber*

No.	Kode Integritas Data *Subscriber*
1	from poly1305 import PolyRun
2	def on_message(mqttc, obj, msg):
3	data_str = json.loads(msg.payload)
4	mac_sub = PolyRun().runMAC(b"80")
5	mac_pub = data_str["mac"]
6	print "Genarate MAC Poly1305-aes Subscriber : ", mac_sub print "===================================="
7	if (mac_sub == mac_pub) :
8	data_sensor = data_str["sensor"]
9	mac_pub = data_str["mac"]
10	db.sensor.insert_one({"sensor" : data_sensor})
11	print "Generate MAC Poly1305-aes Publisher : ",
12	mac_pub

Penjelasan tabel 5.6 interitas data *subscriber* diatas adalah :

1. Baris 1, untuk menjalankan kelas PolyRun dari *file* Poly1305.
2. Baris 2, deklarasi fungsi on_message yang berfungsi untuk menerima pesan atau *payload* dari *broker*.
3. Baris 3, inisialisasi pesan atau *payload* sebagai variable data_str.
4. Baris 4, deklarasi variabel mac_sub untuk memanggil fungsi runMAC dari kelas PolyRun untuk mendapatkan output algoritme Poly1305-AES.
5. Baris 5, deklarasi variable mac_pub untuk mendapatkan output algoritme Poly1305-AES yang dimiliki *publisher*.
6. Baris 8-12, pengkondisian jika *output* algoritme Poly1305-AES *publisher* dan *subscriber* sama maka akan menampilkan data sensor yang dikirim dan akan menyimpan data sensor kedalam basis data.

BAB 6 PENGUJIAN DAN ANALISIS

Pada bab ini akan dibahas mengenai Ppengujian berdasarkan perancangan pengujian yang sudah dibuat pada bab perancangan. Tahap pengujian terdiri dari pengujian fungsional, *test vector,* pengujian peforma sistem dan pengujian keamanan.

6.1 Pengujian Fungsional

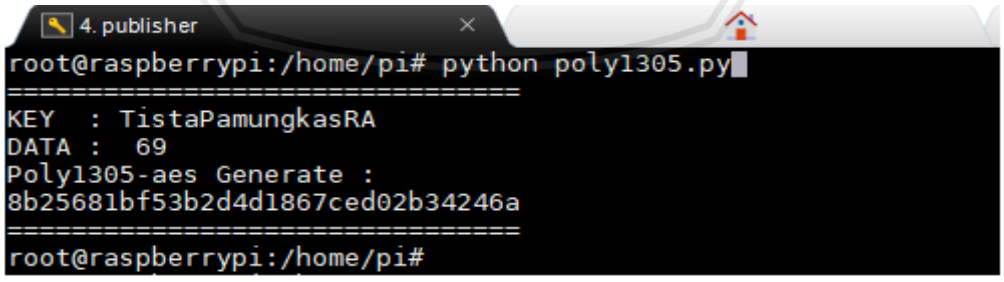
Pengujian fungsional dilakukan untuk memastikan semua fitur telah sesuai dan dapat berjalan dengan benar. Pengujian ini dilakukan dengan melibatkan interaksi antara aktor-aktor yang ada pada sistem seperti *publisher* ke *broker, broker* ke *subscriber dan subscriber* ke *database.* Pengujian ini dilakukan berdasarkan skenario yang telah dijelaskan pada perancangan pengujian fungsional.

6.1.1 Pengujian Pembuatan MAC Poly1305-AES *Publisher*

Pengujian pembuatan MAC Poly1305-AES *publisher* dilakukan untuk memastikan algoritme Poly1305-AES yang diimplementasikan pada *publisher* memproses *key* yang diberikan sebagai masukan dan membuat MAC. Langkah dan hasil pengujian ini dijelaskan melalui Tabel 6.1.

Tabel 6.1 Pengujian Pembuatan MAC Poly1305-AES *Publisher*

Kode	**PF_001**
Prosedur	1. Memasukan *key* dan data sensor pada *file* poly1306.py. 2. Menjalankan *file* poly1306.py pada direktori *publisher*.
Hasil yang diharapkan	*Publisher* dapat membuat MAC berdasarkan masukan yang diberikan.
Hasil Pengujian	*Publisher* berhasil membuat MAC berdasarkan masukan yang diberikan.



Gambar 6.1 Hasil Pengujian Pembuatan MAC Poly1305-AES *Publisher*

Gambar 6.1 merupakan hasil pembuatan MAC Poly1305-AES pada *publisher*. Dengan masukan *key* bernilai TistaPamungkasRA dan data sensor bernilai 69. Pada Gambar 6.1 *publisher* berhasil membuat MAC berdasarkan masukan yang

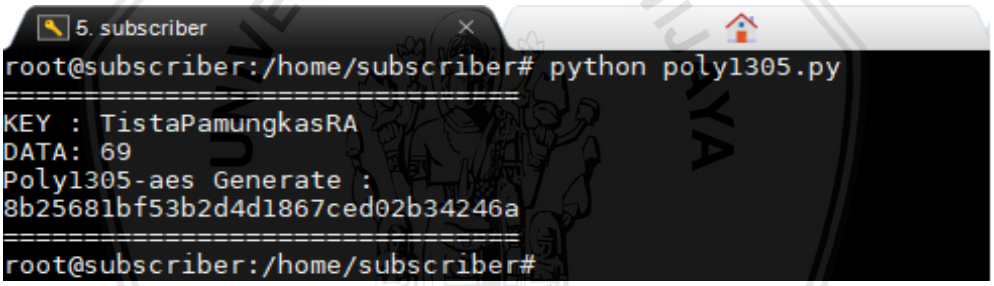
ditentukan. Hal ini menandakan bahwa algoritme Poly1305-AES yang diterapkan pada *publisher* berhasil memproses *key* dan data sensor menjadi MAC.

6.1.2 Pengujian Pembuatan MAC Poly1305-AES *Subscriber*

Pengujian pembuatan MAC Poly1305-AES *subscriber* dilakukan untuk memastikan algoritme Poly1305-AES yang diimplementasikan pada *subscriber* memproses *key* yang diberikan sebagai masukan. Langkah dan hasil pengujian ini dijelaskan melalui Tabel 6.2.

Tabel 6.2 Pengujian Pembuatan MAC Poly1305-AES *Subscriber*

Kode	**PF_002**
Prosedur	1. Memasukan *key* dan data sensor pada *file* poly1306.py. 2. Menjalankan *file* poly1306.py pada direktori *subscriber*.
Hasil yang diharapkan	*Subscriber* dapat membuat MAC berdasarkan masukan yang diberikan.
Hasil Pengujian	*Subscriber* berhasil membuat MAC berdasarkan masukan yang diberikan.



Gambar 6.2 Hasil Pengujian Pembuatan MAC Poly1305-AES *Subscriber*

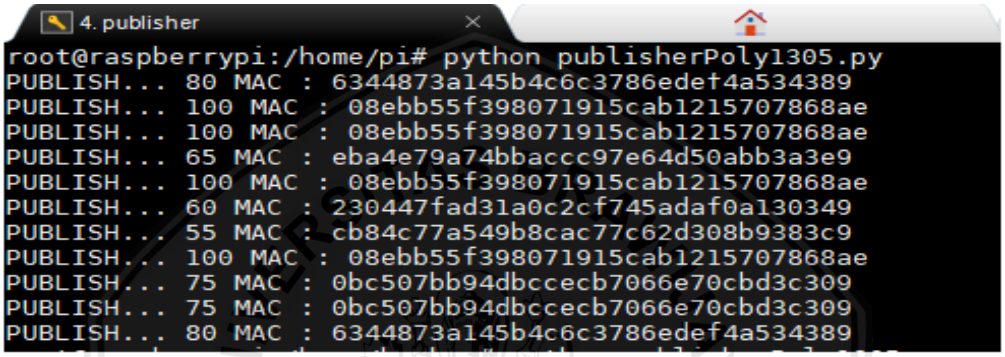
Gambar 6.2 merupakan hasil pembuatan MAC Poly1305-AES pada *subscriber*. Dengan masukan *key* bernilai TistaPamungkasRA dan data sensor bernilai 69. Pada Gambar 6.2 *subscriber* berhasil membuat MAC berdasarkan masukan yang ditentukan. Hal ini menandakan bahwa algoritme Poly1305-AES yang diterapkan pada *subscriber* berhasil memproses *key* dan data sensor menjadi MAC.

6.1.3 Pengujian MQTT *Publish*

Pengujian MQTT *publish* dilakukan untuk memastikan bahwa *publisher* melakukan *publish* dengan *payload* JSON berisi MAC Poly1305-AES dan data sensor. Langkah serta hasil pengujian ini dijelaskan melalui Tabel 6.3.

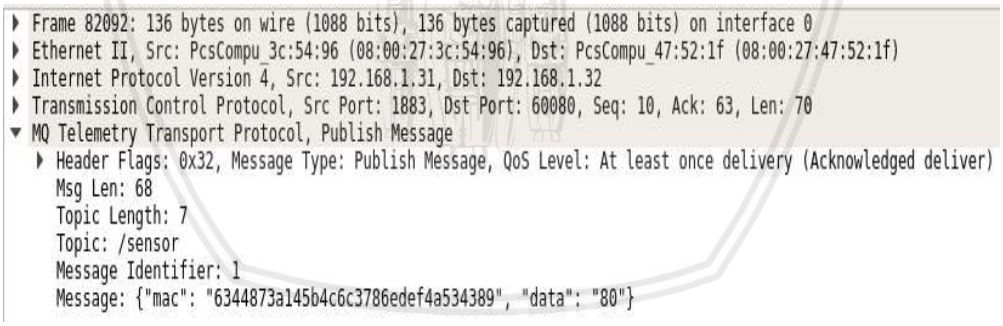
Tabel 6.3 Pengujian MQTT *Publish*

Kode	PF_003
Prosedur	1. Menjalankan hasil eksekusi kode program publisherPoly1305.py. 2. Menampilkan *payload publish* pertama dengan Wireshark.
Hasil yang diharapkan	*Publisher* dapat melakukan *publish* berupa JSON yang berisi data sensor dan MAC Poly1305-AES.
Hasil Pengujian	*Publisher* berhasil melakukan *publish* dengan *payload* yang berisi data sensor dan MAC Poly1305-AES.



Gambar 6.3 Hasil Pengujian MQTT *Publish*

Gambar 6.3 adalah hasil pengujian MQTT *publish* dengan *payload* JSON yang berisi data sensor bernilai dan MAC Poly1305-AES.



Gambar 6.4 Menampilkan *Payload* Pertama Menggunakan Wireshark

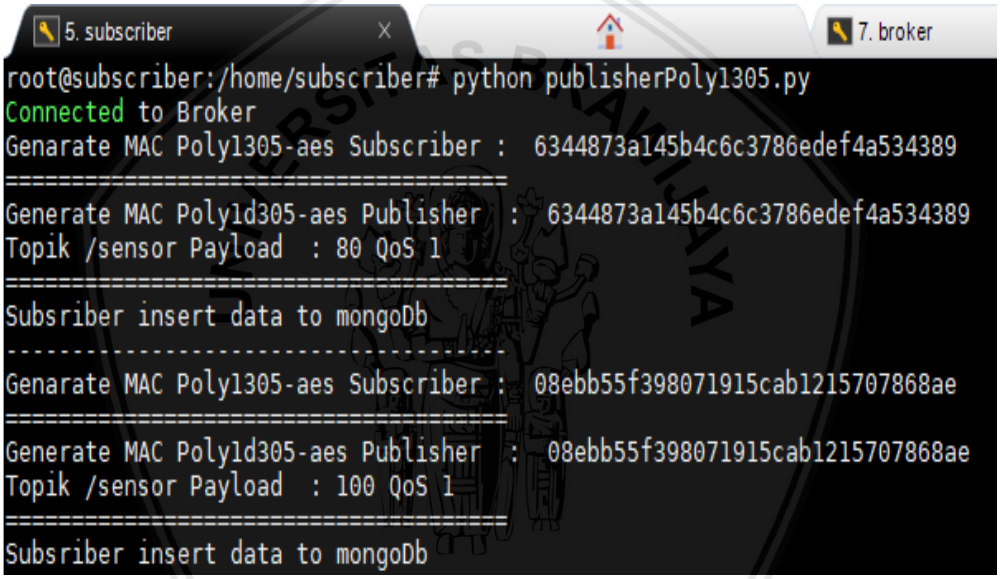
Gambar 6.4 merupakan tampilan hasil *payload* pertama pada *publisher* dengan Wireshark. Sesuai pada Gambar 6.4, dapat disimpulan bahwa pengujian MQTT publish telah berhasil karena *payload publish* pada Wireshark berupa JSON yang berisi data sensor dan MAC Poly1305-AES.

6.1.4 Pengujian MQTT *Subscribe*

Pengujian MQTT *subscribe* dilakukan untuk memastikan bahwa *subscriber* melakukan *publish* dengan *payload* JSON berisi MAC Poly1305-AES dan data sensor. Langkah serta hasil pengujian ini dijelaskan melalui Tabel 6.4.

Tabel 6.4 Pengujian MQTT *Subscribe*

Kode	**PF_004**
Prosedur	1. Menjalankan hasil eksekusi kode program subscriberPoly1305.py. 2. Menampilkan *payload* yang diterima dari *publisher.*
Hasil yang diharapkan	*Subscriber* dapat menerima dan membaca *payload* JSON dari *publisher* yang berisi MAC Poly1305-AES dan data sensor.
Hasil Pengujian	*Subscriber* berhasil menerima dan membaca *payload* JSON dari *publisher* yang berisi MAC Poly1305-AES dan data sensor.



Gambar 6.5 Hasil Pengujian MQTT *Subscribe*

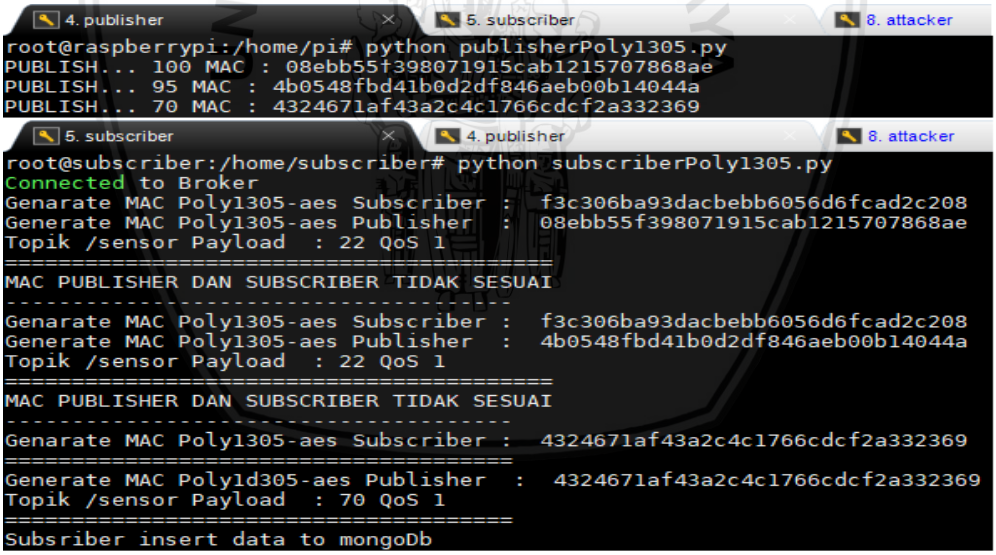
Gambar 6.5 merupakan hasil pengujian MQTT *subscribe* yang menampilkan *payload* dari *publisher* dan membaca *payload* tersebut. Karena *payload* yang diterima berhasil ditampilkan dan dibaca, dapat disimpulkan bahwa pengujian MQTT *subscribe* telah berhasil.

6.1.5 Pengujian Integritas Data Sensor

Pengujian integritas data dilakukan untuk memastikan bahwa *subscriber* menghasilkan MAC dari data sensor yang diterima kemudian membandingkan MAC yang dihasilkan dengan MAC yang diterima dari *publisher*. Prosedur, skenario, serta hasil pengujian ini dijelaskan melalui Tabel 6.5.

Tabel 6.5 Pengujian Integritas Data Sensor

Kode	PF_005
Prosedur	1. *Publisher* megirim *payload* berisi MAC Poly1305-AES dan data sensor. 2. Membaca *payload* JSON kemudian menjadikan data sensor sebagai masukan pembuatan MAC Poly1305-AES pada *subscriber*. 3. Menampilkan *payload* yang diterima dari *publisher*. 4. Membandingkan MAC yang dihasilkan *subscriber* dengan MAC yang diterima dari *publisher*. 5. Menampilkan hasil perbandingan MAC melalui peringatan apakah MAC sesuai atau tidak sesuai.
Hasil yang diharapkan	*Subscriber* dapat mengetahui data sensor yang tidak diubah dan data sensor yang diubah melalui perbandingan MAC dari *publisher* dan MAC dari *subscriber*.
Hasil Pengujian	*Subscriber* berhasil membedakan data sensor yang tidak diubah dan data sensor yang diubah.



Gambar 6.6 Hasil Pengujian Integritas Data Sensor

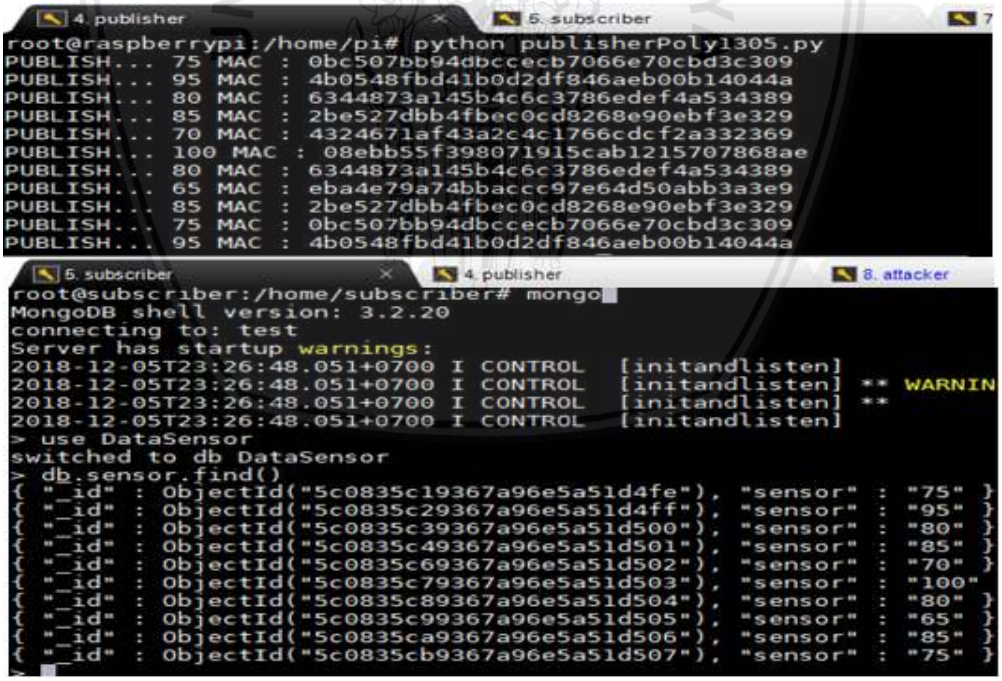
Gambar 6.6 merupakan hasil pengujian integritas data sensor. Pada gambar ini dapat dilihat data sensor dengan nilai 100 dan 95 diubah nilainya ketika sampai di *subscriber* dan sistem secara otomatis akan mencocokan MAC dari *publisher* dan *subscriber* sehingga akan menampilkan peringatan MAC tidak sesuai. Sedangkan data sensor dengan nilai yang tidak diubah akan dimasukkan ke dalam bsis data tanpa ada peringatan MAC tidak sesuai. Berdasarkan hasil yang ditunjukkan Gambar 6.6, pengujian pengecekan integritas data telah berhasil.

6.1.6 Pengujian Penyimpanan Data Sensor

Pengujian menyimpan data sensor dilakukan untuk memastikan bahwa *subscriber* menyimpan data sensor yang diterima pada MongoDB setelah dipastikan integritas datanya. Langkah serta hasil pengujian ini dijelaskan melalui Tabel 6.6.

Tabel 6.6 Pengujian Penyimpanan Data Sensor

Kode	**PF_006**
Prosedur	1. Melakukan *pulish* data. 2. Melakukan *subscribe* dan menampilkan *payload* yang diterima dari *publisher*. 3. Menyimpan data sensor ke MongoDB setelah dipastikan integritasnya. 4. Menampilkan isi koleksi MongoDB
Hasil yang diharapkan	*Subscriber* dapat menyimpan data sensor ke MongoDB setelah melauli proses pengecekan integritas data.
Hasil Pengujian	*Subscriber* berhasil menyimpan data sensor ke MongoDB setelah melauli proses pengecekan integritas data.



Gambar 6.7 Hasil Pengujian Menyimpan Data Sensor

Gambar 6.7 adalah hasil pengujian menyimpan data sensor ke MongoDB. Data sensor yang telah dipastikan integritasnya akan disimpan pada basis data MongoDB. Pada Gambar 6.7 menunjukkan setelah *publish* data sensor yang

dilakukan *publisher, subscriber* akan melakukan peroses pengecekan integritas data yang diterima dari *publisher*. Jika data sensor sudah dipastikan integritasnya maka akan disimpan pada basis data MongoDB. Bisa dilihat pada Gambar 6.7 *subscriber* menampilkan koleksi basis data yang tersedia dan bisa dilihat data yang disimpan pada basis data sama dengan data *publish* yang artinya sudah tersimpan pada basis data dan menandakan bahwa pengujian menyimpan data sensor telah berhasil.

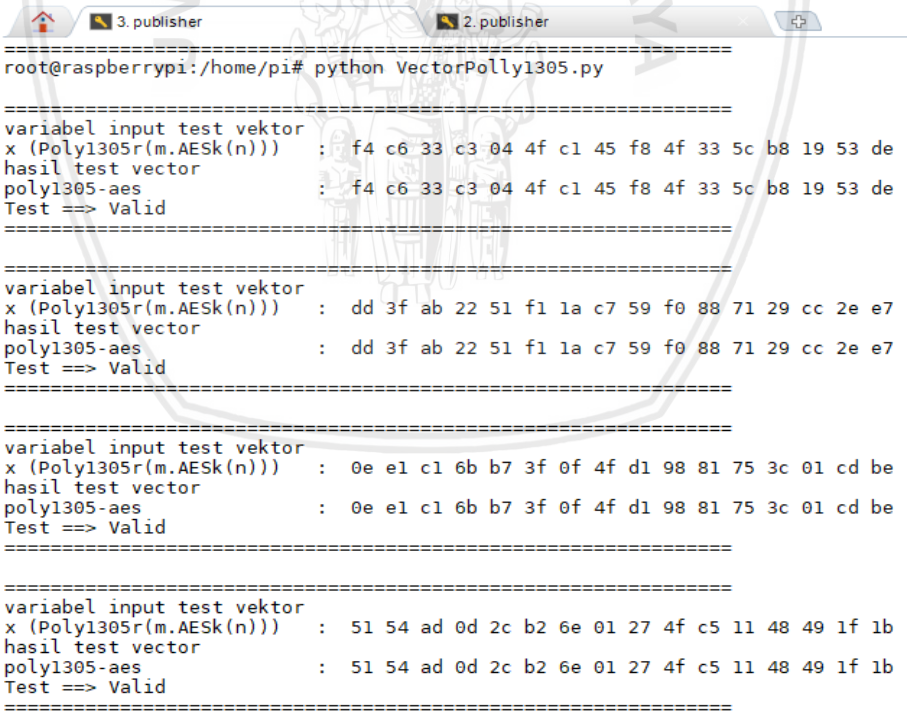
6.2 Pengujian *Test Vektor*

Pengujian *Test Vector* dilakukan untuk memastikan algoritme Poly1305-AES yang diterapkan pada penelitian ini sudah benar dan sesuai dengan ketentuan, maka digunakan *test vector* untuk algoritme Poly1305-AES, yang merujuk pada penelitian The Poly1305-AES message-authentication code oleh Daniel J. Bernstein(2005). Message authentication code yang dihasilkan algoritme Poly1305-AES akan dibandingkan dengan *test vector*. Hasil pengujian *test vector* akan dijelaskan pada Tabel 6. 7 dan Gambar 6.8.

Tabel 6.7 Pengujian *Test Vector*

***Test vector* 1**	
Parameter	***Input***
m (Pesan)	f3 f6
r (Little endian)	85 1f c4 0c 34 67 ac 0b e0 5c c2 04 04 f3 f7 00
k (Key)	ec 07 4c 83 55 80 74 17 01 42 5b 62 32 35 ad d6
n (Nonce)	fb 44 73 50 c4 e8 68 c5 2a c3 27 5c f9 d4 32 7e
x (Poly1305r (m. AESk(n)))	f4 c6 33 c3 04 4f c1 45 f8 4f 33 5c b8 19 53 de
***Test vector* 2**	
Parameter	***Input***
m (Pesan)	
r (Little endian)	a0 f3 08 00 00 f4 64 00 d0 c7 e9 07 6c 83 44 03
k (Key)	75 de aa 25 c0 9f 20 8e 1d c4 ce 6b 5c ad 3f bf
n (Nonce)	61 ee 09 21 8d 29 b0 aa ed 7e 15 4a 2c 55 09 cc
x (Poly1305r (m. AESk(n)))	dd 3f ab 22 51 f1 1a c7 59 f0 88 71 29 cc 2e e7
***Test vector* 3**	
Parameter	***Input***
m (Pesan)	66 3c ea 19 0f fb 83 d8 95 93 f3 f4 76 b6 bc 24

	d7 e6 79 10 7e a2 6a db 8c af 66 52 d0 65 61 36
r (Little endian)	48 44 3d 0b b0 d2 11 09 c8 9a 10 0b 5c e2 c2 08
k (Key)	6a cb 5f 61 a7 17 6d d3 20 c5 c1 eb 2e dc dc 74
n (Nonce)	ae 21 2a 55 39 97 29 59 5d ea 45 8b c6 21 ff 0e
x (Poly1305r (m. AESk(n)))	0e e1 c1 6b b7 3f 0f 4f d1 98 81 75 3c 01 cd be
***Test vector* 4**	
Parameter	***Input***
m (Pesan)	ab 08 12 72 4a 7f 1e 34 27 42 cb ed 37 4d 94 d1 36 c6 b8 79 5d 45 b3 81 98 30 f2 c0 44 91 fa f0 99 0c 62 e4 8b 80 18 b2 c3 e4 a0 fa 31 34 cb 67 fa 83 e1 58 c9 94 d9 61 c4 cb 21 09 5c 1b f9
r (Little endian)	12 97 6a 08 c4 42 6d 0c e8 a8 24 07 c4 f4 82 07
k (Key)	e1 a5 66 8a 4d 5b 66 a5 f6 8c c5 42 4e d5 98 2d
n (Nonce)	9a e8 31 e7 43 97 8d 3a 23 52 7c 71 28 14 9e 3a
x (Poly1305r (m. AESk(n)))	51 54 ad 0d 2c b2 6e 01 27 4f c5 11 48 49 1f 1b



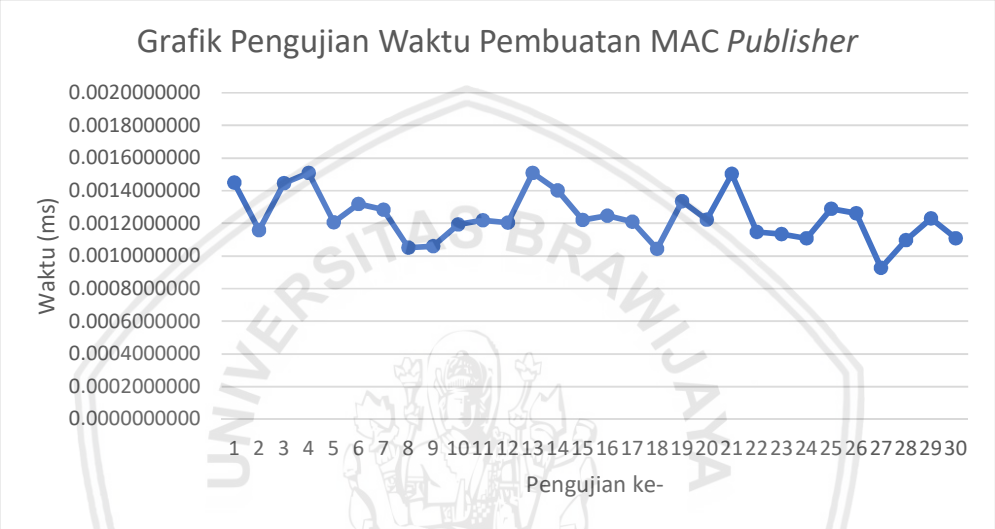
Gambar 6.8 Hasil Pengujian *Test Vector*

Gambar 6.8 menunjukkan hasil pengujian yang dilakukan berdasarkan skenario pengujian yang dijelaskan pada bab perancangan pengujian *test vector*.

Oleh sebab seluruh skenario yang dilakukan memberi hasil yang sama dengan *output vector*, maka dapat disimpulkan bahwa algoritme Poly1305-AES yang diterapkan telah sesuai dan benar.

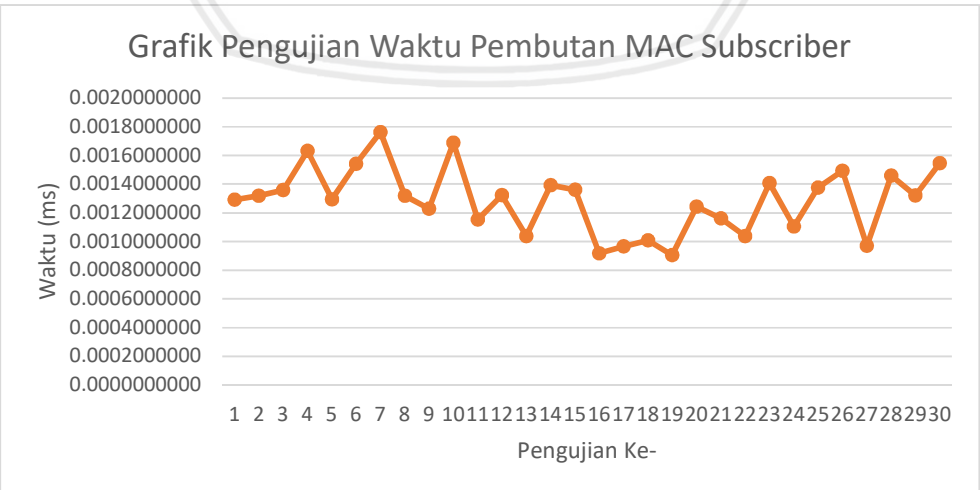
6.3 Pengujian Waktu Eksekusi Algoritme

Pengujian waktu eksekusi algoritme dilakukan untuk tujuan mengetahui berapa lama waktu yang dibutuhkan oleh algoritme Poly1305-AES untuk menghasilkan kode MAC. Berdasarkan skenario yang dibuat pada perancangan pengujian waktu eksekusi algoritme. Berikut ini hasil pengujian waktu eksekusi algoritme *publisher* akan dijelaskan pada Gambar 6.9.



Gambar 6.9 Grafik Pengujian Waktu Eksekusi Algoritme *Publisher*

Berdasarkan grafik yang ditunjukkan pada Gambar 6.9, dapat dilihat bahwa pengujian waktu eksekusi algoritme Poly1305-AES pada *publisher* didapat rata-rata waktu 0,00123 ms, waktu maksimum 0,00150 detik dan waktu minimum 0,00092 ms. Sedangkan hasil pengujian waktu eksekusi *subscriber* dapat dilihat pada Gambar 6.10.



Gambar 6.10 Grafik Pengujian Waktu Eksekusi Algoritme *Subscriber*

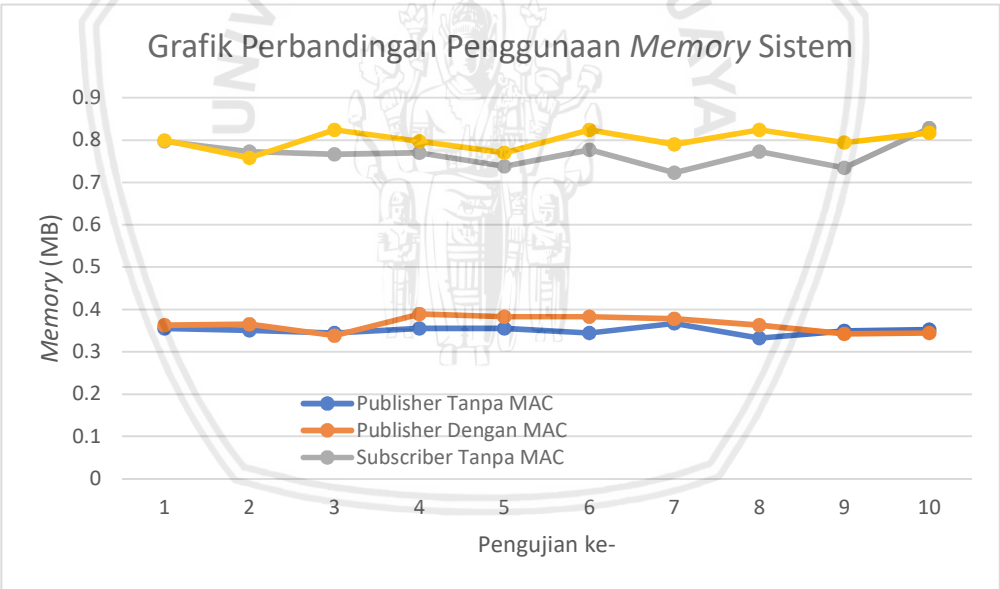
Pada Gambar 6.10 dapat dilihat hasil pengujian waktu eksekusi algoritme *subscriber* memiliki rata-rata waktu eksekusi 0,00128 ms, waktu maksimal 0,00176 ms dan waktu minimal 0,00090 ms. Maka dapat disimpulkan hasil keseluruhan pengujian waktu eksekusi algoritme terdapat perbedaan yang tidak signifikan antara waktu eksekusi algoritme Poly1305-AES pada *publisher* dan *subscriber*.

6.4 Pengujian Peforma Sistem

Pengujian performa sistem dilakukan untuk mengetahui perbandingan kinerja sistem dan kinerja jaringan dalam pemrosesan algoritme berdasarkan skenario yang dibuat pada perancangan pengujian performa sistem.

6.4.1 Pengujian Penggunaan *Memory* Sistem

Pengujian penggunaan *memory* sistem dilakukan untuk mengetahui perbedaan penggunaan *memory* sistem saat menggunakan algoritme dan tidak menggunakan algoritme pembuatan MAC. Sesuai dengan penjelasan skenario [PS_01] pada bab sebelumnya, berikut grafik perbandingan dari pengujian penggunaan *memory* sistem ditunjukkan pada Gambar 6.11.



Gambar 6.11 Grafik Pengujian Penggunaan *Memory* Sistem

Perbandingan penggunaan memory saat sistem dijalankan dengan dan tanpa adanya metode pengecekan integritas data menunjukkan seberapa besar algoritme Poly1305-AES membebani sistem. Berdasarkan grafik yang ditunjukkan pada Gambar 6.11, dapat dilihat bahwa ada peningkatan penggunaan *memory* antara *publisher* dan *subscriber*. Perbedaan peningkatan penggunaan memory ini disebabkan oleh karena subscriber menjalankan lebih banyak proses daripada publisher, salah satunya adalah koneksi pada basis data MongoDB. Hasil pengujian

penggunaan *memory* pada *publisher* dan *subscriber* sebelum dan setelah penerapan Poly1305-AES ditunjukkan melalui Tabel 6.8.

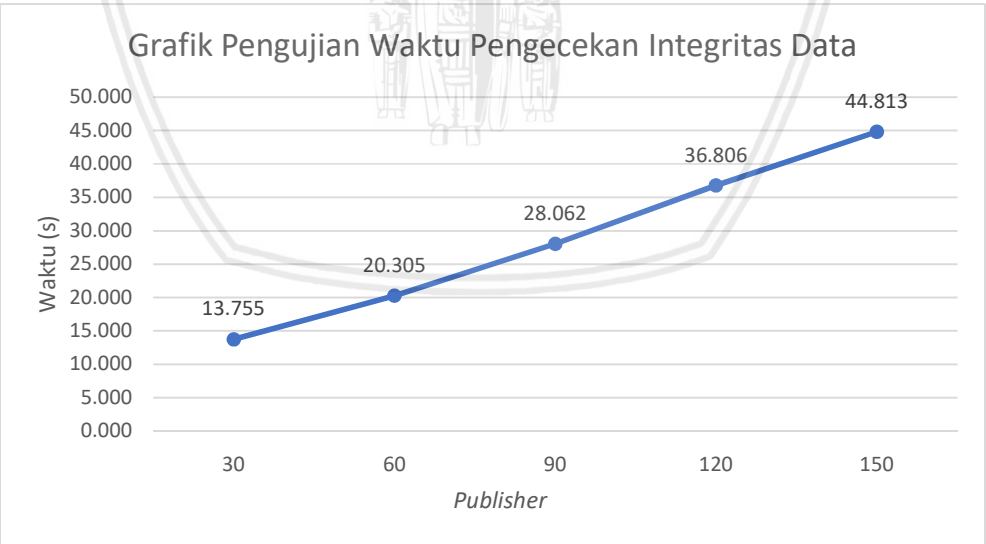
Tabel 6.8 Hasil Pengujian Penggunaan Memory

Keteranagan	Publisher (MB)		Subscriber (MB)	
	Tanpa MAC	Dengan MAC	Tanpa MAC	Dengan MAC
Rata-Rata	0,350	0,363	0,771	0,799
Nilai Maksimum	0,332	0,338	0,723	0,758
Nilai Minimum	0,367	0,389	0,832	0,824

Hasil pengujian penggunaan *memory* berdasarkan pada Tabel 6.8 didapat rata-rata penggunaan *memory* pada *publisher* tanpa menggunakan MAC sebesar 0,350 MB dan 0,363 MB pada *publisher* yang menggunakan MAC. Dan rata-rata penggunaan *memory* pada *subscriber* tanpa menggunakan MAC sebesar 0,771 MB dan 0,799 MB pada *subscriber* yang menggunakan MAC. Maka dapat disimpulkan dari hasi pengujian penggunaan *memory* algoritme Poly1305-AES menggunakan *memory* sebesar 0,0134 MB pada *publisher* dan *memory* sebesar 0,0284 MB pada *subscriber*.

6.4.2 Pengujian Waktu Pengecekan Integritas Data

Pengujian waktu pengecekan integritas data dilakukan untuk mengetahui waktu *subscriber* saat melakukan pengecekan integritas data sensor yang diterima dari *publisher* sesuai dengan skenario pengujian yang dibuat pada bab perancangan. Pengujian ini melakukan *publish* data dari 30, 60, 90, 120, dan 150 *publisher* dalam satu waktu bersamaan. Berikut hasil grafik pengujian waktu pengecekan integritas data akan ditunjukkan melalui grafik pada Gambar 6.12.



Gambar 6.12 Pengujian Waktu Pengecekan Integritas Data

Hasil grafik pengujian waktu pengecekan integritas data yang ditunjukkan pada Gambar 6.12 adalah rata-rata waktu yang diperoleh dari 30 kali *publish* data

dan 10 kali perulangan pada setiap skenario. Sedangkan Tabel 6.9 merupakan hasil pengujian waktu pengecekan integritas data dengan nilai rata-rata , maksimum , dan nilai minimum.

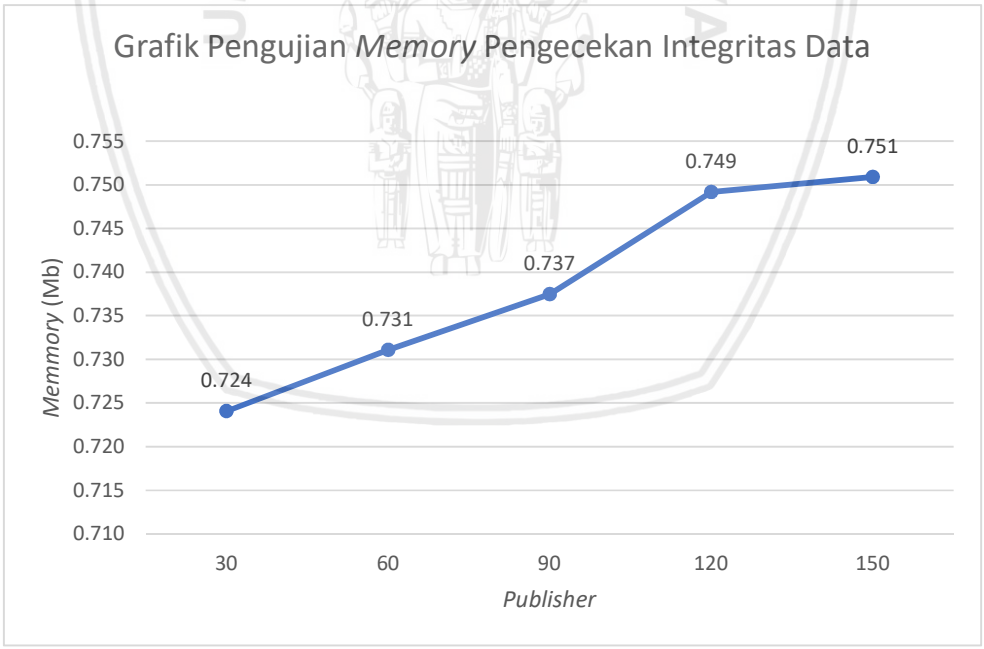
Tabel 6.9 Hasil Pengujian Waktu Pengecekan Integritas Data

Keteterangan	30	60	90	120	150
Rata-Rata	13,755	20,305	28,062	36,806	44,813
Nilai Maksimum	18,870	29,150	34,080	44,330	53,143
Nilai Minimum	10,010	15,620	22,490	30,570	37,090

Berdasarkan nilai rata-rata pada Tabel 6.10 dan grafik pada Gambar 6.12, dapat diambil kesimpulan bahwa waktu pengecekan integritas data meningkat sekitar 7,764 detik setiap kelipatan 30 *publisher*

6.4.3 Pengujian *Memory* Pengecekan Integritas Data

Pengujian *memory* pengecekan integritas data dilakukan untuk mengetahui besar penggunaan *memory subscriber* saat melakukan pengecekan integritas data sensor yang diterima dari *publisher* sesuai dengan skenario pengujian yang dibuat pada bab perancangan. Pengujian ini melakukan *publish* data dari 30, 60, 90, 120, dan 150 *publisher* dalam satu waktu bersamaan. Berikut hasil grafik pengujian *memory* pengecekan integritas data akan ditunjukkan melalui grafik pada Gambar 6.13.



Gambar 6.13 Grafik Pengujian *Memory* Pengecekan Integritas Data

Hasil grafik pengujian *memory* pengecekan integritas data yang ditunjukkan pada Gambar 6.13 adalah rata-rata waktu yang diperoleh dari 30 kali *publish* data dan 10 kali perulangan pada setiap skenario jumlah *publisher*. Sedangkan Tabel

6.10 adalah hasil pengujian *memory* pengecekan integritas data dengan nilai rata-rata , maksimum , dan nilai minimum.

Tabel 6.10 Hasil Pengujian *Memory* Pengecekan Integritas Data

Keterangan	30	60	90	120	150
Rata-Rata	0,724	0,731	0,737	0,749	0,751
Nilai Maksimum	0,777	0,832	0,828	0,832	0,832
Nilai Minimum	0,676	0,680	0,680	0,680	0,680

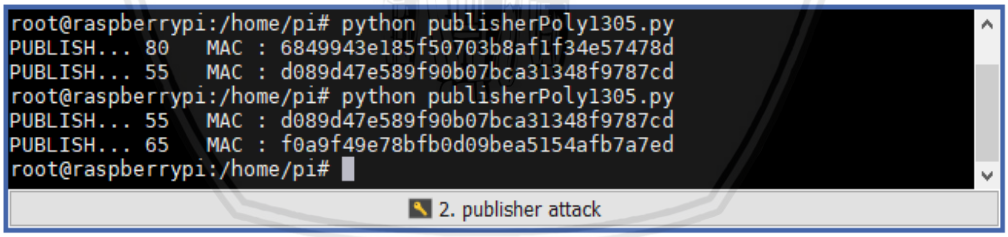
Berdasarkan nilai rata-rata pada Tabel 6.10 dan grafik pada Gambar 6.12, dapat diambil kesimpulan bahwa waktu pengecekan integritas data meningkat sekitar 0,0067 MB setiap kelipatan 30 *publisher.*

6.5 Pengujian Keamanan

Pengujian keamanan dilakukan untuk mengetahui bangaimana kemanan sistem yang dibuat sudah berjalan dengan benar. Pengujian ini terfokus untuk menguji integritas data yang ada pada sistem berdasarkan skenario yang dibuat pada perancangan pengujian keamanan.

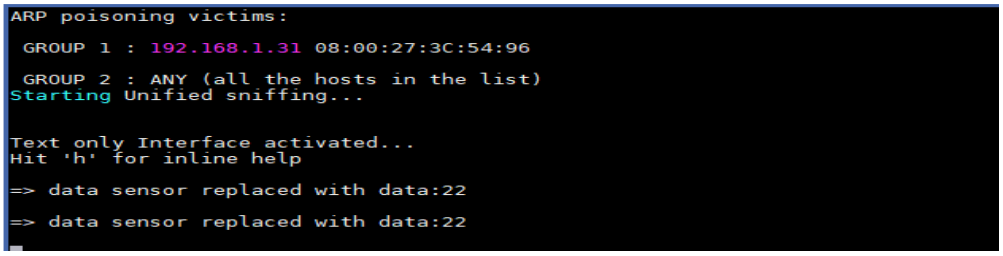
6.5.1 Pengujian Penyerangan Perubahan Data

Pengujian penyerangan perubahan data dilakukan untuk mengetahui bagaimana sistem dapat menangani penyerangan perubahan data. Sesuai dengan penjelasan skenario [PK_01] pada bab sebelumnya, berikut ini hasil pengujian penyerangan perubahan data ditunjukkan pada Gambar 6.14 sampai Gambar 6.16.



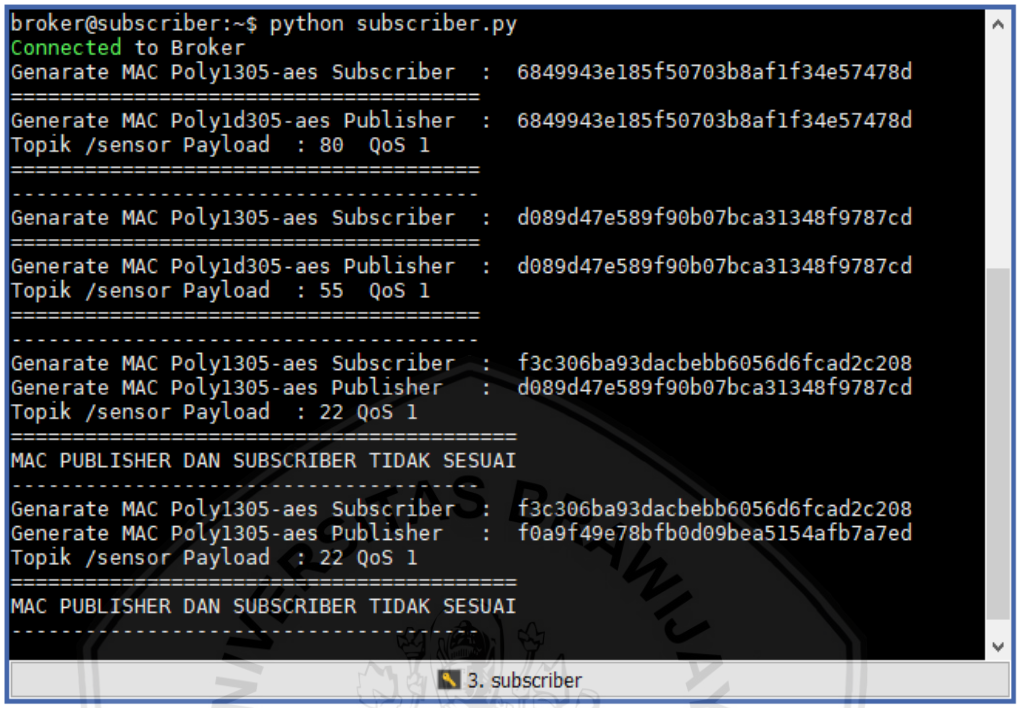
Gambar 6.14 *Publisher* Perubahan Data

Publisher melakukan *publish* data ke *broker* sebanyak dua kali dimana *publish* data pertama adalah data yang tidak diserang dan *publish* data kedua adalah data yang diserang.



Gambar 6.15 Penyerangan Perubahan Data

Pada Gambar 6.15 ditunjukan bahwa penyerang melakukan ARP Spoofing pada *broker* dan data yang telah di-*publish* akan diubah dengan nilai 22.

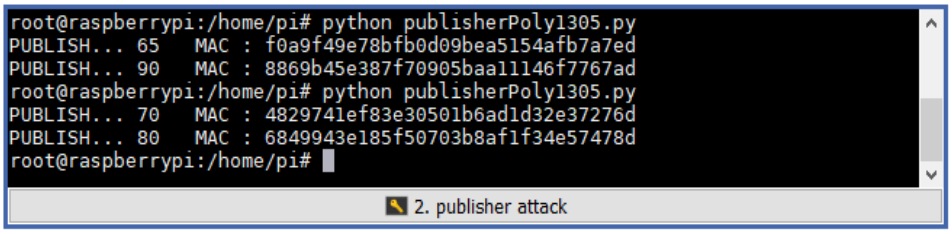


Gambar 6.16 *Subscriber* Perubahan Data

Pada Gambar 6.16 ditunjukkan bahwa *subscriber* telah menerima data dari *broker.* Penyerangan perubahan data dilakukan dengan merubah data sensor dari *publisher* yang bernilai awal 55 dan 65 menjadi 22. S*ubscriber* menampilkan pesan MAC *publisher* dan *subscriber* tidak sesuai karena *subscriber* memiliki mekanisme pengecekan integritas data dengan membadingkan nilai MAC dari *publisher* dan *subscriber.* Oleh karena itu, dapat diambil kesimpulan bahwa *subscriber* mampu mengatasi serangan perubahan data dan dapat menjamin integritas data yang ditransmisi.

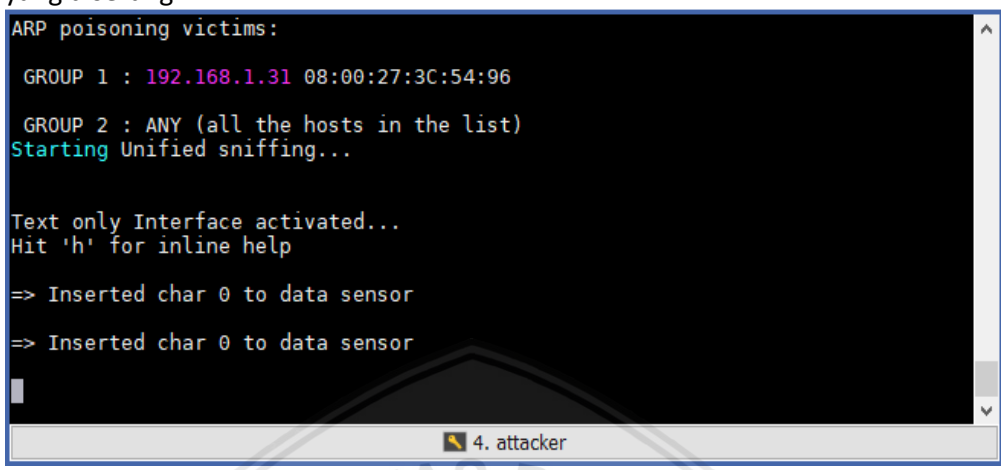
6.5.2 Pengujian Penyerangan Penyisipan Data

Pengujian penyerangan penyisipan data dilakukan untuk mengetahui bagaimana sistem dapat menangani penyerangan penyisipan data. Sesuai dengan penjelasan skenario [PK_02] pada bab sebelumnya, berikut ini hasil pengujian penyerangan penyisipan data ditunjukkan pada Gambar 6.17 sampai Gambar 6.19.



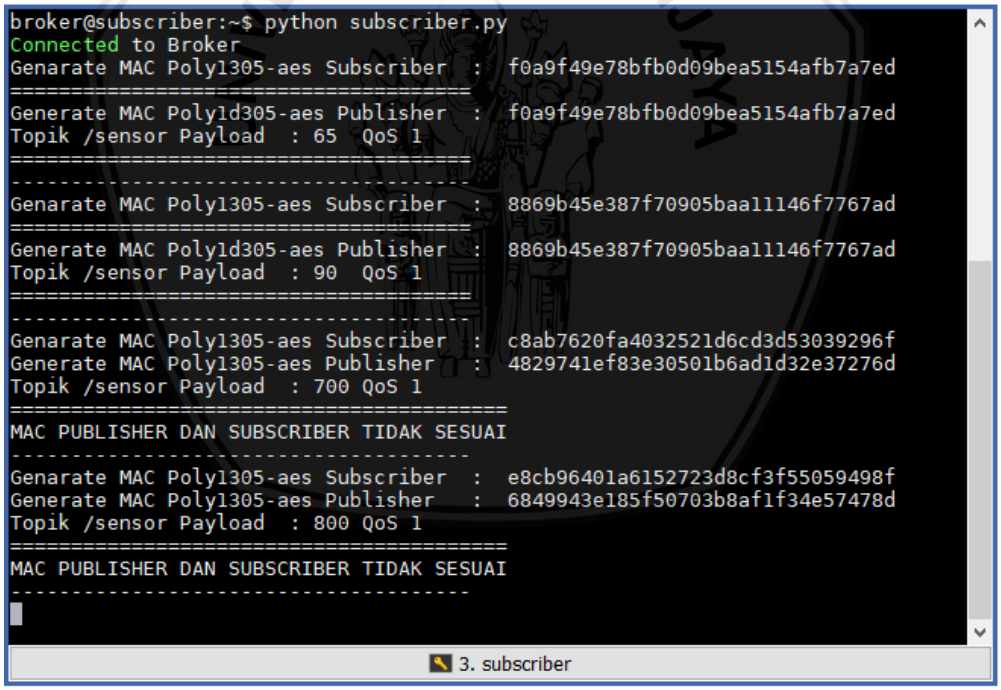
Gambar 6.17 *Publisher* Penyisipan Data

Publisher melakukan *publish* data ke *broker* sebanyak dua kali dimana *publish* data pertama adalah data yang tidak diserang dan *publish* data kedua adalah data yang diserang.



Gambar 6.18 Penyerangan Penyisipan Data

Pada Gambar 6.18 ditunjukan bahwa penyerang melakukan ARP Spoofing pada *broker* dan telah berhasil menyisipkan nilai 0 pada bagian belakang data.



Gambar 6.19 *Subscriber* Penyisipan Data

Pada Gambar 6.19 ditunjukan bahwa *subscriber* telah menerima data dari *broker.* Penyerangan penyisipan data dilakukan dengan menyisipkan nilai 0 pada bagian belakang data sensor yang dikirimkan *broker.* Data sensor dari *publisher* yang bernilai awal 70 dan 80 berubah menjadi 700 dan 800 setelah penyerangan. *Subscriber* menampilkan pesan MAC *publisher* dan *subscriber* tidak sesuai karena

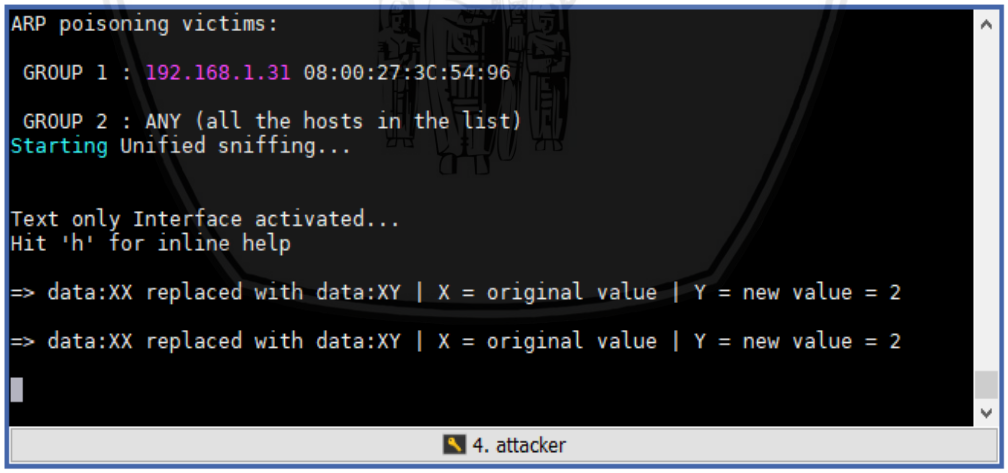
subscriber memiliki mekanisme pengecekan integritas data dengan membadingkan nilai MAC dari *publisher* dan *subscriber.* Oleh karena itu, dapat diambil kesimpulan bahwa *subscriber* mampu mengatasi serangan penyisipan data dan dapat menjamin integritas data yang ditransmisi.

6.5.3 Pengujian Penyerangan Pensubtitusian Data

Pengujian penyerangan pensubtitusian data dilakukan untuk mengetahui bagaimana sistem dapat menangani penyerangan pensubtitusian data. Sesuai dengan penjelasan skenario [PK_03] pada bab sebelumnya, berikut ini hasil pengujian penyerangan pensubtitusian data ditunjukan pada Gambar 6.20 sampai Gambar 6.22.

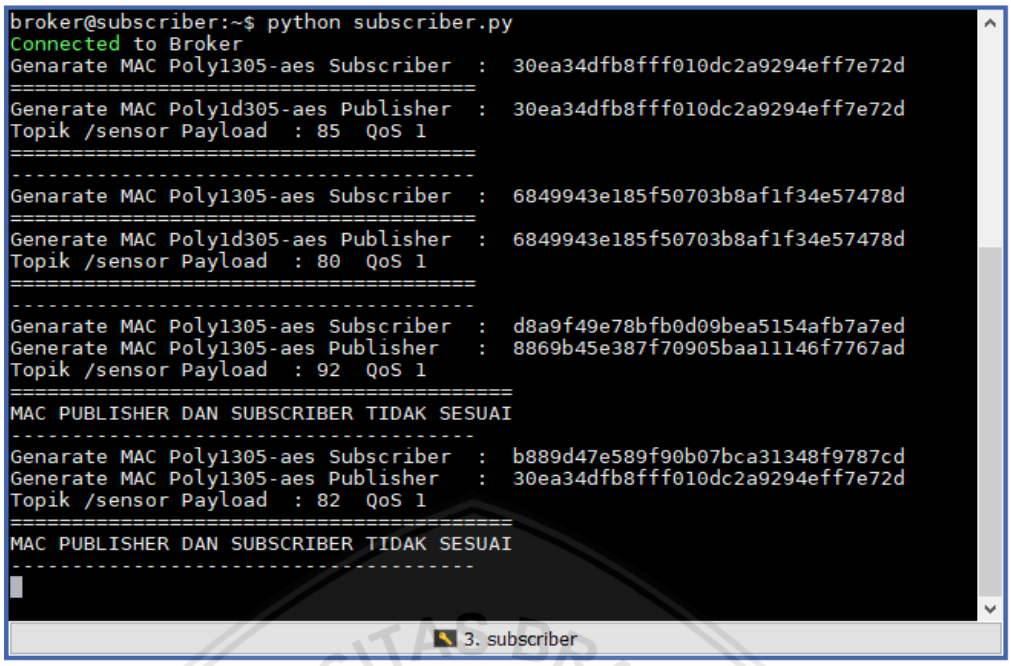
Gambar 6.20 *Publisher* Pensubtitusian Data

Publisher melakukan *publish* data ke *broker* sebanyak dua kali dimana *publish* data pertama adalah data yang tidak diserang dan *publish* data kedua adalah data yang diserang.



Gambar 6.21 Penyerangan Pensubtitusian Data

Pada Gambar 6.21 ditunjukan bahwa penyerang melakukan ARP Spoofing pada *broker* dan telah berhasil melakukan subtitusi nilai data sensor pada angka terakhir dengan nilai 2.



Gambar 6.22 *Subscriber* Pensubtitusian Data

Pada Gambar 6.22 ditunjukan bahwa *subscriber* telah menerima data dari *broker.* Penyerangan pensubtitusian data dilakukan dengan melakukan subtitusi angka terakhir pada data sensor dengan nilai 2. Data sensor dari *publisher* yang bernilai awal 90 dan 85 berubah menjadi 92 dan 82 setelah penyerangan. S*ubscriber* menampilkan pesan MAC *publisher* dan *subscriber* tidak sesuai karena s*ubscriber* memiliki mekanisme pengecekan integritas data dengan membadingkan nilai MAC dari *publisher* dan *subscriber.* Oleh karena itu, dapat diambil kesimpulan bahwa *subscriber* mampu mengatasi serangan pensubtitusian data dan dapat menjamin integritas data yang ditransmisi.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan masalah yang telah dirumuskan serta hasil yang diperoleh dari pengujian, maka kesimpulan yang didapat adalah sebagai berikut:

1. *Subscriber* dapat mengetahui integritas data sensor dengan cara menghitung nilai MAC dari data sensor yang diterima. MAC dapat dihitung dengan inputan *key, integer, nonce* dan data sensor yang diterima. Jika *output* MAC yang dihasilkan *subscriber* sama dengan MAC yang dihasilkan *publisher*, maka data sensor yang diterima telah dipastikan integritasnya dan tidak mengalami perubahan, penyisipan dan pensubtitusian saat transmisi data oleh *broker*.
2. Algoritme Poly1305-AES saat digunakan untuk pengecekan integritas data pada protokol MQTT memiliki rata-rata waktu 0,00123 ms dan untuk menghasilkan MAC pada *publisher* dan pada *subscriber* sebesar 0.00128 ms. Sedangkan peningkatan penggunaan *memory* ketika algoritme Poly1305-AES diterapkan pada sistem *publisher* sebesar 0,013 MB dan 0,028 MB pada *subscriber*. Pengecekan integritas data yang dilakukan *subscriber* meningkat 7,764 detik dan penigkatan penggunaan *memory* 0,0067 MB setiap kelipatan 30 *publisher.*

7.2 Saran

Saran yang dapat diberikan untuk penelitian selanjutnya terkait topik tentang pengecekan integritas data sensor pada protokol MQTT adalah:

1. Melakukan penambahan aspek keamanan lainnya seperti aspek kerahasiaan data menggunakan algoritme enkripsi.
2. Menambahkan mekanisme *key management* dalam dalam mengatur pertukaran *key* antara *publisher* dan *subscriber.* Sehingga lebih efektif saat diterapkan pada sistem berskala besar.
3. Menambahkan perangkat keras sensor agar data yang diperoleh lebih nyata dan akurat.

DAFTAR PUSTAKA

Al-Fuqaha, A. et al., 2015. Internet of Things: A Survey on Enabling Technologies, Protocols and Applications. *IEEE Communication Surveys & Tutorial,* 17(4), pp. 2347-2376.

Andy, S., B. R. & B. H., 2017. Attack scenarios and security analysis of MQTT communication protocol in IoT System.

Atmoko, R. A., R. R. & M. K. H., 2017. IoT real time data acquisition using MQTT protocol.

Aumasson, et al., 2013. BLAKE2: simpler, smaller, fast as MD5.

Bernstein, D. J., 2005. The Poly1305-AES message-authentication code.

Guoqiang, S., Yanming, C., Chao, Z. & Yanxu, Z., 2013. Design and Implementation of a Smart IoT Gateway. *2013 IEEE International Confrence on Green Computing and Communications and IEEE Cyber, Physical and Social Computing,* pp. 720-723.

Hunkeler, U., Truong, H. L. & Clark, A. S., 2015. A Publish/Subscribe Protocol For Wireless Sensor Network. *IEEE.*

Lobna Yehia, A. K. A. D., 2015. Hybrid Security Techniques for Internet of Things Healthcare Applications. *SciRes.*

Mahmoud Elkhodr, S. S. a. H. C., 2016. THE INTERNET OF THINGS: NEW INTEROPERABILITY, MANAGEMENT AND SECURITY CHALLENGES.

MongoDB, 2015. *PyMongo 3.4.0 Documentation.* [Online] Available at: https://api.mongodb.com/python/current/ [Accessed 11 April 2017].

MQTT, 2016. *Frequently Asked Question.* [Online] Available at: http://www.mqtt.org/faq [Accessed 8 Oktober 2016].

Munir, R., 2005. *Kriptografi.* s.l.:s.n.

Python Software Foundation, 2017. *Paho-mqtt 1.2.2.* [Online] Available at: https://pypi.python.org/pypi/paho-mqtt [Accessed 11 April 2017].

Python Software Foundation, 2017. *The Python Tutorial.* [Online] Available at: https://docs.python.org/2.7/tutorial/index.html [Accessed 24 Mei 2017].

Syaiful Andy, Budi Rahardjo, Bagus Hanindhito, September 2017. Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System. *EECSI.*

Thangavel, D. et al., , 2014. Performance Evaluation of MQTT and CoAP via a Common Middleware. IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing..

Tutorialspoint, 2016. *MongoDB - Overview.* [Online] Available at: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm [Accessed 8 Oktober 2016].

Tutorialspoint, 2016. *Python Overview.* [Online] Available at: https://www.tutorialspoint.com/python/python_overview.htm [Accessed 8 Oktober 2016].

Wangbong Lee, K. N. H.-G. R. S.-H. K., 2016. A Gateway based Fog Computing Architecture for Wireless Sensors and Actuator Networks.

Yindong , C., Liping, L., Ziran, C., 2017. An Approach to Verifying Data Integrity for Cloud Storage.