ANALISIS MEKANISME KEAMANAN ANTARA TLS/SSL DAN *CRYPTO* PADA KOMUNIKASI IOT *MIDDLEWARE* DENGAN *SUBSCRIBER* BERBASIS PROTOKOL HTTP

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Deny Hari Prasetya Dewa
NIM: 135150200111043

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

ANALISIS MEKANISME KEAMANAN ANTARA TLS/SSL DAN *CRYPTO* PADA KOMUNIKASI IOT *MIDDLEWARE* DENGAN *SUBSCRIBER* BERBASIS PROTOKOL HTTP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Deny Hari Prasetya Dewa
NIM: 135150200111043

Skripsi ini telah diuji dan dinyatakan lulus pada
18 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eko Sakti P., S.Kom, M.Kom	Dany Primanita K., S.T., M.Kom
NIK: 201102 860805 1 001	NIP: 19771116 200501 2 003

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

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

Malang, 24 Januari 2018

Deny Hari Prasetya Dewa

NIM: 135150200111043

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT atas nikmat dan rahmat-Nya, serta hidayah-Nya, penulis dapat menyelesaikan penulisan skripsi dengan baik. Penulisan skripsi ini diajukan untuk memenuhi sebagian persyaratan untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya. Judul yang penulis ajukan adalah "Analisis Mekanisme Keamanan Antara TLS/SSL dan *Crypto* Pada Komunikasi IoT *Middleware* Dengan Berbasis Protokol HTTP".

Dengan kesempatan ini, penulis ingin menyampainkan banyak terima kasih kepada semua pihak yang telah mendukung serta membantu penulis selama masa pengerjaan penelitian ini, diantaranya:

1. Kedua orang tua penulis, Eko Hari Syamsudin dan Wiwik Rukmi Wuryani yang selalu memberi dukungan dan do'a agar penulis dapat menyelesaikan penelitian ini.
2. Bapak Eko Sakti P., S.Kom, M.Kom dan Ibu Dany Primanita K., S.T., M.Kom selaku dosen pembimbing I dan dosen pembimbing II yang dengan sabar dan tabah telah membimbing penulis selama penelitian ini.
3. Bapak Agi Putra Kharisma, S.T, M.T selaku dosen penasihat akademik yang telah memberikan kritik dan saran selama proses perkuliahan.
4. Bapak Wayan Firdaus Mahmudy, S. Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
5. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
6. Bapak Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
7. Semua teman FILKOM yang telah memberi dukungan serta bantuan selama penelitian. Khususnya pada tim bimbingan Bapak Eko yang telah melalui suka dan duka bersama penulis.
8. Semua pihak yang tidak dapat penulis sebutkan satu persatu yang terlibat langsung maupun tidak langsung dalam penelitian ini.

Semoga Allah SWT melimpahkan segala rahmat dan karunia-Nya kepada semua pihak yang telah meberikan bantuan kepada penulis. Tanpa bantuan dari semua pihak yang telah disebutkan di atas, penulis tidak akan mampu menyelesaikan penelitian dengan judul Analisis Mekanisme Keamanan Antara TLS/SSL dan *Crypto* Pada Komunikasi IoT *Middleware* Dengan Berbasis Protokol HTTP" ini dengan lancar.

Penulis menyadari bahwa penelitian ini masih banyak kekurangannya sehingga penulis menerima kritik dan saran yang membangun secara terbuka. Akhir kata, penulis berharap supaya penelitian ini dapat bermanfaat bagi semua pihak yang menggunakannya.

Malang, 24 Januari 2018

Penulis

denyhari.pras@gmail.com

ABSTRAK

Terdapat celah keamanan pada pengiriman pesan pada sebuah *middleware* dimana pesan yang dikirimkan masih dalam bentuk *plain text*, yang menyebabkan kerahasiaan data menjadi tidak terjamin. Aktifitas *eavesdropping* akan sangat mudah dilakukan jika tidak segera diimplementasikan sebuah mekanisme keamanan. Berdasarkan permasalahan tersebut, diimplementasikan mekanisme *end-to-end security* menggunakan mekanisme keamanan TLS/SSL dan *Crypto*. Penggunaan mekanisme keamanan tidak hanya mengamankan jaringan, tetapi juga mempengaruhi kinerja sistem tersebut. Proses enkripsi dan dekripsi memperlambat kinerja *middleware*. Penelitian ini berfokus untuk mengetahui bagaimana dampak implementasi mekanisme keamanan TLS/SSL dan *Crypto*, khususnya AES-256 terhadap keamanan data dan kinerja *middleware*. Sebelum mekanisme keamanan diimplementasikan, data yang terdapat pada jaringan sangat mudah dibaca dan diketahui isinya. Setelah kedua mekanisme keamanan diimplementasikan, data tidak dapat diketahui lagi isinya karena telah dienkripsi. TLS/SSL memberikan keamanan yang lebih kuat dengan menggunakan pertukaran sertifikat sebagai sistem otentikasi. Kedua metode yang dipasang mampu mengamankan sistem tetapi keduanya mempengaruhi kinerja *middleware*. Kedua mekanisme keamanan tidak berpengaruh besar terhadap penggunaan CPU dan *memory*. AES-256 menghasilkan nilai yang lebih baik pada parameter *packet loss,* dengan nilai 1%. Sedangkan TLS/SSL lebih unggul di parameter *delay* dan *jitter,* dengan nilai 0.089008423 detik dan 0.003208877 detik.

Kata kunci: *Internet of Things, middleware, TLS/SSL, AES-256, keamanan, delay*

ABSTRACT

There is a security hole in message delivery on a middleware where messages are still in plain text, causing data confidentiality to be insecure. Eavesdropping activities will be very easy to do if a security mechanism is not implemented immediately. Based on the problem, implemented end-to-end security mechanism using TLS / SSL and Crypto. The use of security mechanisms not only secures the network, but also affects the performance of the system. The process of encryption and decryption slows down middleware performance. This study focuses on knowing how the impact of TLS / SSL and Crypto security mechanisms, especially AES-256 on data security and middleware performance. Before the security mechanism is implemented, the data contained on the network is very readable. After both security mechanisms are implemented, the data can not be known again because the contents have been encrypted. TLS / SSL provides stronger security by using certificate exchanges as an authentication system. Both methods installed are capable of securing the system but both affect the performance of middleware. Both security mechanisms have no significant effect on CPU and memory usage. AES-256 produces better values on packet loss parameters, with a value of 1%. While TLS / SSL is superior in parameter delay and jitter, with value 0.089008423 sec and 0.003208877 sec.

Keywords: *Internet of Things, middleware, TLS/SSL, AES-256, security, delay*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar belakang

Pada penelitian sebelumnya telah dikembangkan sebuah *middleware* dengan pendekatan *event-driven* yang mampu mendukung interoperabilitas berbagai macam perangkat atau sensor (Anwari, 2017). Pada penelitian tersebut, *middleware* dapat melakukan komunikasi *end-to-end* dengan aplikasi menggunakan *subscriber* berbasis protokol HTTP. Dalam skema komunikasi tersebut, terdapat celah keamanan dimana paket data yang dikirimkan masih berupa *plain text*.

Celah keamanan ini memberikan ancaman berupa *Eavesdropping*. Rajra dan J Deepa (2015) menjelaskan bahwa *eavesdropping* adalah penangkapan komunikasi antara dua titik oleh pihak yang tidak berwenang. Untuk mengatasi hal tersebut, dibutuhkan mekanisme keamanan untuk mengatasi celah keamanan pada komunikasi *end-to-end* pada *middleware* tersebut, mekanisme keamanan tersebut adalah *end-to-end security*. *End-to-end security* bergantung pada protokol dan mekanisme yang diimplementasikan pada koneksi *endpoints*, keamanan pada *endpoints* memiliki beberapa kebutuhan, diantaranya; identitas, protokol, algoritma, implementasi dan operasi yang aman (Michael, 2009). Dalam *end-to-end security* terdapat dua metode yang dapat digunakan, yaitu TLS/SSL dan Kriptografi. Kriptografi adalah seni penulisan rahasia yang digunakan sejak zaman Romawi untuk menyembunyikan informasi rahasia atau menjaga keamanan pesan. Untuk menjaga kerahasiaan informasi, metode yang banyak digunakan adalah enkripsi / dekripsi (Maqsood F, 2017). Terdapat banyak algoritma kriptografi yang digunakan untuk mengamankan informasi seperti DES, 3DES, *Blowfish*, AES, RSA, ElGamal dan *Paillier*. Maqsood F (2017), dalam penelitiannya membandingkan kinerja dari setiap algoritma berdasarkan waktu enkripsi dan dekripsi, penelitian tersebut menghasilkan data yang menunjukkan bahwa waktu enkripsi dan dekripsi algoritma AES adalah yang paling cepat. Sedangkan penulis menggunakan algoritma AES-256 untuk penelitian ini.

TLS/SSL merupakan protokol yang paling banyak digunakan untuk memastikan autentikasi, integrasi dan *confidentiality* pada pertukaran informasi antara *web server* dan *web client* (Turner, 2014). Protokol ini memilki dua bagian, pertama adalah *handshake protokol* untuk membuat *session keys* dan *record protokol* untuk mengamankan data aplikasi dengan *session keys* dan memastikan integrasi dan keaslian dari data (Turner, 2014). Tehupeiory N (2016), dalam penelitiannya menggunakan mekanisme keamanan TLS/SSL untuk mengamankan koneksi *file* FTP. Sedangkan cara kerja AES adalah dengan mengulang langkah-langkah yang sama yang sudah ditetapkan berkali-kali, AES adalah semacam algoritma enkripsi untuk kunci rahasia dan beroperasi pada sejumlah byte yang tetap (Bhajaj, 2016). Ramdhansya (2014), dalam penelitiannya menggunakan algoritma AES untuk mengamankan sistem berbasis mikrokontroler Arduino menggunakan *Bluetooth*.

Sebelum melakukan implementasi mekanisme keamanan ke dalam *middleware,* terdapat beberapa hal yang harus diperhatikan. Mudassar Ahmad (2012), dalam penelitiannya membuktikan bahwa penggunaan mekanisme keamanan akan mengurangi kinerja jaringan. Kinerja adalah permasalahan yang utama pada implementasi mekanisme keamanan. Terdapat beberapa parameter jaringan yang harus diperhatikan untuk melihat kinerja jaringan, seperti tingkat pengiriman, *throughput*, *delay*, probabilitas tabrakan, efisiensi *bandwidth*, rasio *packet loss*, *bit error rate*, *delay* antrian, dan *jitter* (Islam N, 2016). Mekanisme keamanan yang menerapkan fungsi kriptografi terhadap paket yang dikirimkan mengakibatkan peningkatan *delay* karena proses enkripsi dan dekripsi, mekanisme keamanan yang menggunakan algoritma lebih kuat akan menghasilkan *delay* yang lebih besar (Suartana, 2017). Peningkatan *delay* juga diakibatkan oleh proses identifikasi dan otentikasi pada mekanisme keamanan yang dipakai (Suartana, 2017).

Oleh karena itu, dibutuhkan sebuah studi tentang penerapan *end-to-end security,* agar dapat mengetahui mekanisme keamanan apa yang dapat mengamankan *middleware* IoT dan mekanisme mana yang lebih efisen jika digunakan. Berdasarkan penjelasan di atas, penulis ingin menganalisis kinerja dan pengaruh antara dua mekanisme keamanan pada *middleware* IoT, yaitu TLS/SSL dan AES-256. Dengan analisis ini, diharapkan dapat mengetahui apakah mekanisme keamanan TLS/SSL dan AES-256 dapat mengamankan komunikasi dan bagaimana pengaruh mekanisme keamanan tersebut terhadap kinerja komunikasi.

1.2 Rumusan masalah

Berdasarkan uraian di atas, penulis merumuskan beberapa masalah sebagai berikut:

1. Bagaimana kinerja mekanisme kemanan antara TLS/SSL dan AES-256 pada komunikasi *middleware* dengan *subscriber* pada IoT Middleware?
2. Bagaimana pengaruh mekanisme keamanan antara TLS/SSL dan AES-256 terhadap kinerja komunikasi *middleware* dengan *subscriber* pada IoT Middleware?

1.3 Tujuan

Berdasarkan rumusan masalah di atas, tujuan dari penulisan skripsi ini adalah:

1. Mengimplementasikan mekanisme keamanan TLS/SSL dan AES-256 pada middleware IoT.
2. Mengetahui pengaruh mekanisme keamanan antara TLS/SSL dan AES-256 terhadap kinerja pengiriman data.
3. Mengetahui mekanime keamanan yang lebih aman dan efisien antara TLS/SSL dan AES-256 untuk digunakan pada middleware IoT.

1.4 Manfaat

Beberapa manfaat yang dapat diperoleh dari pembuatan skripsi ini adalah sebagai berikut:

Untuk Penulis:

1. Dapat mengimplementasikan ilmu yang telah dipelajari di universitas.
2. Menambah pengetahuan tentang keamanan jaringan.

Untuk Prodi:

1. Memberikan inspirasi untuk penelitian dan pengembangan keamanan jaringan selanjutnya agar dapat tercipta jaringan yang aman.

Untuk Masyarakat:

1. Menyediakan mekanisme kemanan jaringan IoT terutama pada titik antara middleware dengan data center.

1.5 Batasan masalah

Batasan masalah dalam penelitian ini adalah:

1. Jaringan yang diteliti adalah antara *middleware* dengan data center.
2. *Middleware* yang digunakan adalah *middleware* yang dikembangkan oleh Husnul Anwari (2017).
3. Parameter uji yang digunakan adalah penggunaan CPU dan *memory, packet loss, delay,* dan *jitter*.

1.6 Sistematika penulisan

Bagian ini berisi struktur skripsi ini mulai Bab Pendahuluan sampai Bab Penutup dan deskripsi singkat dari masing-masing bab. Diharapkan bagian ini dapat membantu pembaca dalam memahami sistematika pembahasan isi dalam skripsi ini. Sistematika penulisan skripsi ini adalah sebagai berikut :

BAB I PENDAHULUAN

Pada bab satu menjelaskan tentang latar belakang dari penelitian ini, rumusan masalah yang didapat, tujuan dari penelitian, manfaat dari penelitian, batasan masalah dari sistem yang akan dibuat dan sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAAN

Pada bab dua menguraikan kajian pustaka beserta teori, konsep, metode, atau sistem literatur ilmiah yang digunakan sebagai pendukung penelitian ini. Landasan kepustakaan ini digunakan sebagai dasar pengujian testbed, pengambilan dan pengolahan data, serta analisis dan pembahasan pada penelitian ini.

BAB III METODOLOGI PENELITIAN

Pada bab tiga ini membahas tentang metodologi yang digunakan dalam penelitian, bagaimana tahap-tahap dari penelitian ini mulai dari studi literatur, pembangunan lingkungan uji, pengujian dan pengambilan data, pengolahan data dan analisis, serta kesimpulan dan saran.

BAB IV PEMBANGUNAN LINGKUNGAN UJI

Pada bab ini penulis akan menjelaskan berbagai kebutuhan yang diperlukan untuk melakukan penelitian. Kebutuhan penelitian tersebut meliputi kebutuhan perangkat keras, kebutuhan perangkat lunak, middleware, serta implementasi mekanisme keamanan.

BAB V PENGUJIAN DAN PENGAMBILAN DATA

Bab ini membahas pengujian *middleware* dan pengambilan data yang dilakukan oleh penulis terhadap mekanisme keamanan TLS/SSL dan AES-256 pada middleware IoT. Kemudian data yang telah diambil akan diolah agar dapat dianalisa oleh penulis.

BAB VI PENGOLAHAN DATA DAN ANALISIS

Bab ini memuat pengolahan data yang digunakan untuk mendapatkan nilai parameter uji dan analisis hasil dari pengambilan data serta data yang sudah diolah oleh penulis.

BAB VII PENUTUP

Bab ini berisi kesimpulan dari penelitian yang dilakukan serta saran terkait dengan penelitian oleh penulis. Kesimpulan akan menjawab pertanyaan-pertanyaan yang terdapat pada rumusan masalah.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi tentang uraian tentang penelitian terkait serta teori-teori yang digunakan penulis sebagai dasar teori penelitian.

2.1 Kajian Pustaka

Kajian pustaka digunakan untuk perbandingan antara penelitian terkait yang sudah dilakukan dengan penelitian yang akan dilakukan, yang diuraikan dalam tabel 2.1.

Tabel 2.1 Penelitian Terkait

No	Judul Jurnal	Perbedaan	
		Penelitian Terdahulu	Rencana Penelitian
1	Ramdhansya A.F, Ariyanto E, Nuha H.H., 2014. *IMPLEMENTASI ADVANCED ENCRYPTION STANDARD (AES) PADA SISTEM KUNCI ELEKTRONIK KENDARAAN BERBASIS SISTEM OPERASI ANDROID DAN MIKROKONTROLER ARDUINO*. Yogyakarta. Universitas Telkom.	Mengimplementasi algoritma AES pada sistem berbasis mikrokontroler Arduino menggunakan *bluetooth*.	Melakukan analisis mekanisme keamanan TLS/SSL dan AES-256 pada komunikasi *middleware* dan *subscriber*.
2	Tehupeiory N, Chandra D.W., 2016. *Analisis Perbandingan Mekanisme Secure Socket Layer (SSL) dan Transfer Layer Security (TLS) Pada Koneksi File Transfer Protokol (FTP) Server Ubuntu*. Salatiga. Universitas Kristen Satya Wacana.	Menganalisis perbandingan mekanisme SSL dan TLS pada koneksi *file* transfer FTP.	Melakukan analisis mekanisme keamanan TLS/SSL dan AES-256 pada komunikasi *middleware* dan *subscriber*.

2.2 *Internet of Things Middleware*

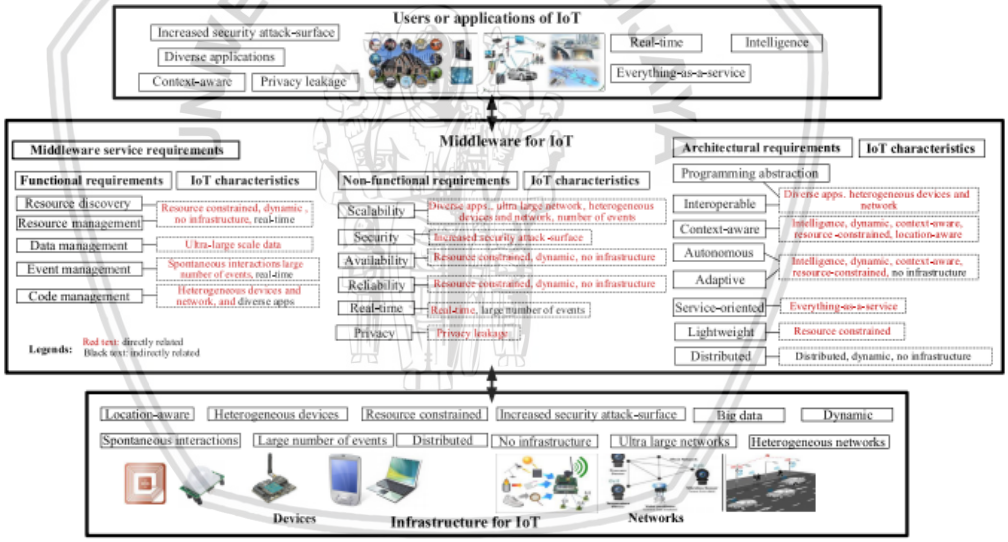
Untuk mempermudah perangkat dalam berkomunikasi dengan internet, *IoT* membutuhkan sebuah *middleware* yakni *software* atau *hardware* yang menyediakan *interface* bagi perangkat untuk mengirimkan atau mendapatkan data. Dalam *IoT*, penggunaan *middleware* sangat beragam misalkan, *middleware* untuk menghubungkan sensor dengan *cloud*, *middleware* untuk *smart-device* terhubung dengan *smart-phone* ataupun untuk komunikasi *machine-to-machine* (M2M). Pada umumnya *middleware* memiliki *interface* bagi sensor untuk mengirimkan data dan juga *interface* bagi aplikasi untuk membaca data tersebut.

Yang membedakan antara *middleware* satu dengan *middleware* lainnya adalah bagaimana *middleware* menerapkan *interface*, melakukan manajemen data, serta menjaga keamanan dan privasi data (Anwari, 2016).

Dengan kondisi IoT saat ini, *middleware* yang dikembangkan harus dapat beradaptasi dan menyelesaikan tantangan yang ada. Beberapa penelitian telah dilakukan untuk merumuskan tantangan tersebut dan bagaimana menyelesaikannya. Penelitian ini meliputi survei dan analisis tentang *middleware* yang sudah saat ini, pola-pola perancangan serta konsep arsitektur *middleware* (Anwari, 2016). Berdasarkan karakteristik infrastruktur IoT dan aplikasi yang bergantung padanya, dapat diuraikan kebutuhan untuk *middleware* yang mendukung IoT. Kebutuhan tersebut dibagi menjadi dua bagian (Abdur, 2016):

1. Layanan yang harus diberikan dari *middleware* (*Middleware Service Requirements*)
2. Arsitektur sistem harus mendukung (*Architectural Requirements*).

Hubungan infrastruktur *IoT* dengan aplikasi dan *middleware*-nya dapat dilihat pada gambar 2.1 sebagai berikut:



Gambar 2.1 Hubungan antara infrastruktur IoT, aplikasi dan *middleware*

Sumber: Abdur (2016)

2.3 Arsitektur *Middleware*

Beberapa survei dan penelitian telah dilakukan untuk mengetahui apa saja jenis *middleware* yang ada saat ini dan bagaimana kinerja tiap *middleware* dalam mengatasi tantangan di atas. (Anne, et al., 2016) membagi *middleware* menjadi tiga jenis.

1. *Service-based middleware*

Arsitektur *service-based middleware* terdiri dari tiga layer yakni *Physical Plane* (sensors dan actuator), *Virtualized Plane* (server atau layanan cloud) serta *Application Plane* (*utility*). Unit komputasi berada pada layer *Virtualized Plane.* Fungsi pada *middleware* ini cukup beragam mulai dari *access control*, *storage management* dan *event processing engine*, akan tetapi tidak untuk analisis data. *Middleware* ini mempunyai performa yang tinggi dan membutuhkan banyak tenaga untuk mengoperasikannya, biasanya *middleware* jenis ini di-*deploy* di beberapa *server* dalam *cloud*. *Middleware* ini juga tidak didesain untuk *constrained devices* seperti *smartphone* dan tidak mendukung komunikasi *Machine-to-Machine* (M2M).

2. *Cloud-based middleware*

Fungsi yang ditawarkan *middleware* ini terbatas pada apa yang ada pada *cloud*. Fungsi dapat berupa layanan penyimpanan data, komputasi tingkat lanjut atau analisis data. Biasanya, fungsi tersebut diekspose melalui serangkaian API yang hanya dapat diakses oleh aplikasi dari penyedia cloud atau komputer di *cloud* lainnya yang mendukung RESTful API.

3. *Actor-based middleware*

Actor-based middleware pertama kali diperkenalkan pada sebuah proyek oleh universitas, pemerintah dan perusahaan pribadi di USA. Arsitektur *middleware* ini terbagi menjadi tiga layer. Lapisan luar yakni *Sensory Swarm* (sensor dan aktuator), lapisan tengah yakni *Mobile Access* (*gateway* seperti *smartphone*, Raspberry Pi, Swarmbox, Laptop) dan lapisan dalam yakni *cloud*. *Middleware* dengan arsitektur ini didesain sangat ringan dan dapat dideploy pada semua lapisan (sensory layer, mobile access layer dan cloud). Unit komputasi (actor) pada *middleware* ini terdistribusi dalam jaringan, misalkan *middleware* yang diimplementasikan pada *smartwatch* mungkin tidak menyertakan fungsi storage service. Akan tetapi fungsi tersebut dapat diambil alih oleh perangkat lain dengan mengunduh *actor* dari *repository* yang disediakan.

2.4 *End-to-end Security*

End-to-end security adalah metode dalam komunikasi yang aman yang dapat mencegah supaya pihak ketiga tidak dapat mengakses data saat data tersebut dipindahkan dari satu *endpoint* atau dari perangkat ke perangkat lain. *End-to-end Security* bergantung pada protokol dan mekanisme yang diterapkan secara eksklusif pada titik akhir koneksi, contoh yang paling umum adalah koneksi HTTPS (berbasis *Transport Layer Protocol* (TLS)) ke sebuah server web; *IP Security* (IPSec) juga bisa digunakan untuk *end-to-end security*, seperti saat pada awalnya diusulkan sebagai mekanisme koneksi untuk IPv6 (Michael, 2013).

Definisi dari sebuah *endpoint* adalah klien atau server. Dengan definisi tersebut, *end-to-end security* dimulai pada klien dan berakhir di server. Mengingat banyaknya jumlah aplikasi yang berjalan secara paralel pada sebuah sistem operasi, dan mengingat meningkatnya virtualisasi, definisi ini biasanya tidak lagi cukup tepat (Michael, 2013). Sistem operasi dapat membentuk asosiasi keamanan

baik pada *session* atau tingkat aplikasi, hal ini juga dapat diakhiri di *front end*, dikarenakan banyaknya server (Michael, 2013).

2.4.1 Komponen Utama *End-to-end Security*

Keamanan pada *endpoint* (client-server, atau client-client untuk peer-to-peer) adalah persyaratan mutlak untuk komunikasi yang aman. Solusi semacam ini memiliki komponen berikut (Michael, 2013):

1. Identitas

Komponen ini mencakup identitas entitas yang diketahui dan dapat diverifikasi pada kedua ujungnya, identitas bisa bersifat sementara untuk koneksi. Misalnya, pengguna diidentifikasi menggunakan *username* dan *password*, sedangkan server dapat diidentifikasi melalui sertifikat server (Michael, 2013).

2. Protokol

Protokol digunakan untuk bertukar kunci *session* secara dinamis, dan untuk menyediakan fungsi keamanan yang diperlukan (misalnya, enkripsi dan verifikasi integritas atau keaslian) untuk sebuah koneksi, contohnya TLS dan IPsec. Protokol menggunakan algoritma untuk mengimplementasikan fungsi-fungsi tersebut (Michael, 2013).

3. Algoritma

Algoritma ini menggunakan *session key* yang telah disebutkan sebelumnya untuk melindungi data dalam perjalanan, misalnya melalui enkripsi atau pemeriksaan integritas, contohnya adalah AES, 3DES, dan SHA-1 (Michael, 2013).

4. Implementasi yang aman

Endpoint (client atau server) yang menjalankan salah satu protokol yang disebutkan diatas harus bebas dari *bug* yang bisa membahayakan keamanan. Keamanan browser web sangat relevan disini. Juga malware bisa membahayakan keamanan, misalnya dengan membukukan kunci pada PC (Michael, 2013).

5. Operasi yang aman

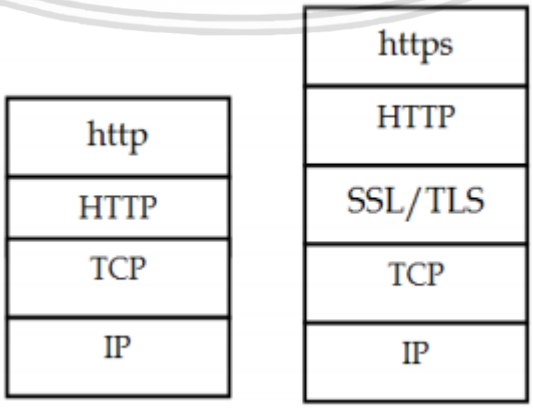
Pengguna dan operator harus memahami mekanisme keamanan, dan bagaimana cara menangani pengecualiannya. Misalnya, *web browser* memberi peringatan tentang sertifikat server yang tidak valid, namun pengguna dapat menghiraukan peringatan tersebut dan tetap melakukan koneksi. Hal ini adalah masalah yang nonteknis, namun sangat memprihatinkan (Michael, 2013).

Untuk *end-to-end security* penuh, semua komponen diatas harus aman. Dalam jaringan dengan *end-to-end security*, kedua *endpoint* biasanya (tergantung pada protokol dan algoritma yang digunakan) bergantung pada fakta bahwa komunikasi mereka tidak terlihat oleh orang lain, dan tidak ada orang lain yang dapat memodifikasi data saat dikirim (Michael, 2013).

2.5 HTTP (*Hypertext Transfer Protocol*) dan HTTPS (HTTP *Secure*)

Cosmas Eko Suharyanto dan Pastima Simanjuntak (2017), dalam penelitiannya menjelaskan bahwa *Hypertext Transfer Protocol* (HTTP) adalah protokol aplikasi untuk sistem informasi terdistribusi, kolaboratif, *hypermedia*. HTTP telah digunakan oleh inisiatif informasi global *World Wide Web* sejak tahun 1990. Versi pertama HTTP, yang disebut HTTP / 0.9, adalah protokol sederhana untuk transfer data mentah ke Internet. Versi protokol saat ini adalah HTTP / 1.1, yang menambahkan beberapa fitur tambahan ke versi 1.0 sebelumnya. HTTP menyediakan *framework* umum untuk kontrol akses dan otentikasi, melalui skema otentikasi *request-response* yang dapat diperluas, yang dapat digunakan oleh server untuk meminta *request* klien dan oleh klien untuk memberikan informasi autentikasi. HTTP mendefinisikan metode untuk menentukan tindakan yang diinginkan oleh sumber yang diidentifikasi. Sumber mewakili apakah data awal atau data yang dibuat sebelumnya secara dinamis, tergantung pada penerapan server. Spesifikasi HTTP / 1.0 mendefinisikan metode GET, POST dan HEAD dan spesifikasi HTTP / 1.1. menambahkan 5 metode baru yakni OPSI, PUT, DELETE, TRACE and CONNECT.

Cosmas Eko Suharyanto dan Pastima Simanjuntak (2017), dalam penelitiannya juga menjelaskan bahwa HTTP (RFC2616) pada awalnya digunakan secara umum di Internet. Namun, peningkatan penggunaan HTTP untuk aplikasi yang sensitif membutuhkan tindakan pengamanan. SSL, dan penggantinya TLS (RFC2246) dirancang untuk menyediakan *channel* keamanan. Koneksi HTTPS terutama digunakan untuk transaksi pembayaran di *World Wide Web*, e-mail dan untuk transaksi yang sensitif dalam informasi sistem perusahaan. Pada akhir tahun 2000an dan awal 2010an, HTTPS mulai meluas penggunaannya untuk melindungi keaslian halaman pada semua jenis situs web, mengamankan akun dan menjaga komunikasi pengguna, identitas dan *web browsing*. Untuk membuat pesan HTTPS, pengirim mengintegrasikan preferensi pengirim dengan preferensi penerima. Hasilnya adalah daftar algoritma kriptografi yang akan diterapkan dan kunci yang digunakan untuk menerapkannya.



Gambar 2.2 Perbedaan HTTP dan HTTPS

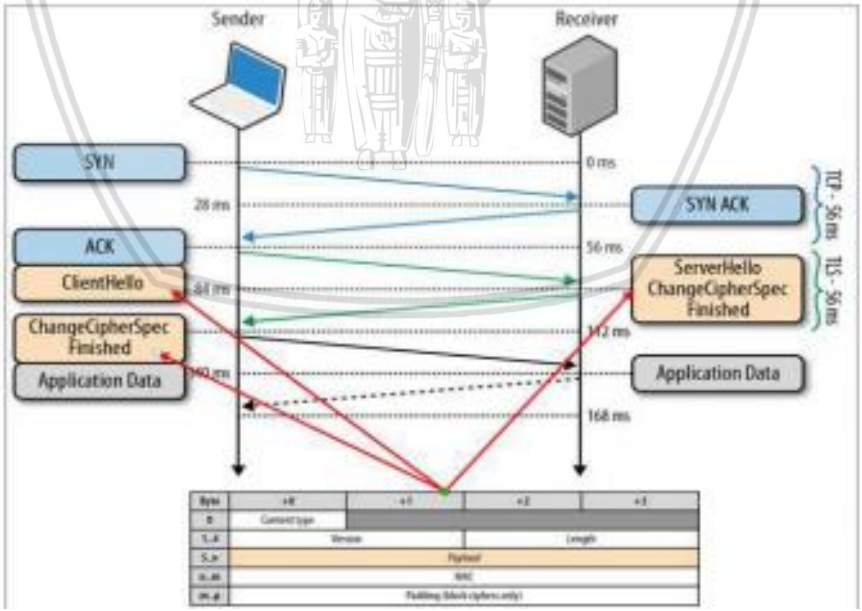
Sumber: Singh (2016)

Gambar 2.2 di atas menunjukkan perbedaan protokol HTTP dan HTTPS. Terlihat bahwa terdapat penggunaan SSL/TLS pada protokol HTTPS.

2.6 TLS/SLL (*Transport Layer Security / Secure Socket Layer*)

Protokol TLS/SSL memiliki dua bagian, yang pertama adalah *handshaking protocol*, yang kedua adalah *record protocol* . *Handshaking protocol* menegoisasi suite cipher, mengotentikasi server dan secara opsional mengotentikasi klien dan menetapkan *session keys* (Turner, 2014). Sedangkan *record protocol* mengamankan data aplikasi dengan *session keys* yang dibuat pada *record protocol* dan memverifikasi keaslian dan integritas aplikasi (Turner, 2014).

Handshaking protocol sebenarnya memiliki tiga sub-protokol. *Handshaking protocol* melakukan angkat berat dengan menegosiasikan versi protokol, sesi identifier, kunci publik peer (opsional), kompresi Metode, spesifikasi cipher, dan *master secret* (Turner, 2014). The *alert protocol*, yang biasanya tidak dikirim selama *handshaking protocol*, memberitahu rekan tentang penyebab kegagalan protokol (Turner, 2014). Peringatan dikategorikan baik sebagai peringatan kegagalan, dalam hal sesi dihentikan, atau peringatan *warning*, dalam hal penerima mendapat pilihan untuk mengakhiri sesi (Turner, 2014). Peringatan fatal menghasilkan koneksi dihentikan yang tidak dapat dilanjutkan (Turner, 2014). *Change cipher specification protocol* menginformasikan peer bahwa pengirim ingin mengubah ke satu set kunci baru, yang diciptakan dari informasi dipertukarkan oleh *handshaking protocol* (Turner, 2014). Proses *handshake* yang terjadi pada protokol TLS/SSL dapat dilihat pada gambar 2.3.

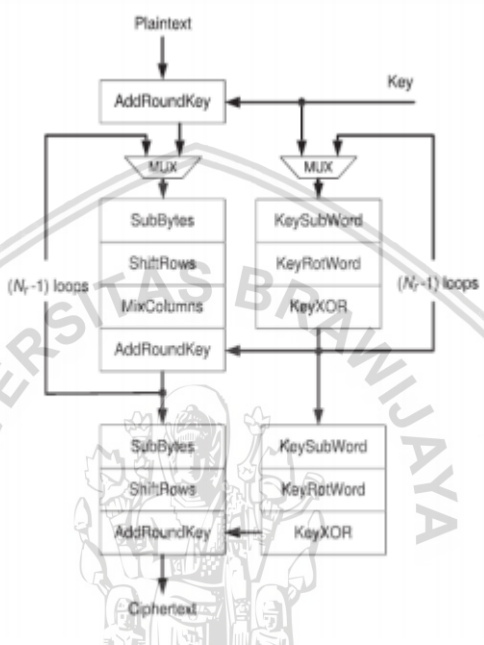


Gambar 2.3 SSL *handshake*

Sumber: Behren (2017)

2.7 AES (Advanced Encryption Standard)

AES adalah algoritma kriptografi dan standar keamanan komputer yang dirancang untuk tujuan keamanan dan untuk melindungi data-data digital (Bhajaj, 2016). Cara kerja AES adalah dengan mengulang langkah-langkah yang sama yang sudah ditetapkan berkali-kali, AES adalah semacam algoritma enkripsi untuk kunci rahasia dan beroperasi pada sejumlah byte yang tetap, kunci ini diperbesar menjadi sub-kunci untuk setiap putaran operasi (Bhajaj, 2016). Proses ini dinamakan *key expansion*.



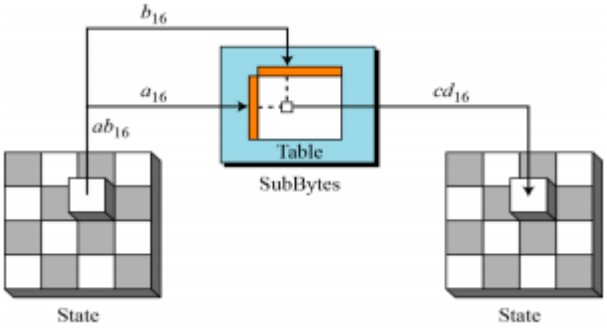
Gambar 2.4 Diagram block dari algoritma AES

Sumber: Bhajaj (2016)

Gambar 2.4 di atas menggambarkan cara kerja algoritma AES dalam mengenkripsi data. AES mengambil input sebagai blok data berukuran 128-bit dan melakukan putaran transformasi untuk menghasilkan *cipher text* sebagai *output.* Blok input data diproses oleh array byte *4-by-4.* Ukuran kunci putaran bisa berukuran 128, 192 atau 256 bits (Bhajaj, 2016). Panjang kunci putaran akan menentukan jumlah putaran yang diulang pada AES. Kunci putarannya adalah 10, 12 atau 14 untuk masing-masing panjang kunci 128, 192 atau 256 bits (Bhajaj, 2016). Untuk enkripsi data input, terdapat 4 transformasi yang dilakukan pada AES, yaitu:

2.7.1 *SubBytes*

Transformasi SubBytes adalah subtitusi byte nonlinier. Setiap byte dari input diganti dengan byte yang lain sesuai dengan kotak subtitusi, kotak subtitusi ini disebut dengan S-box (Bhajaj, 2016). Proses subtitusi digambarkan pada gambar 2.5.

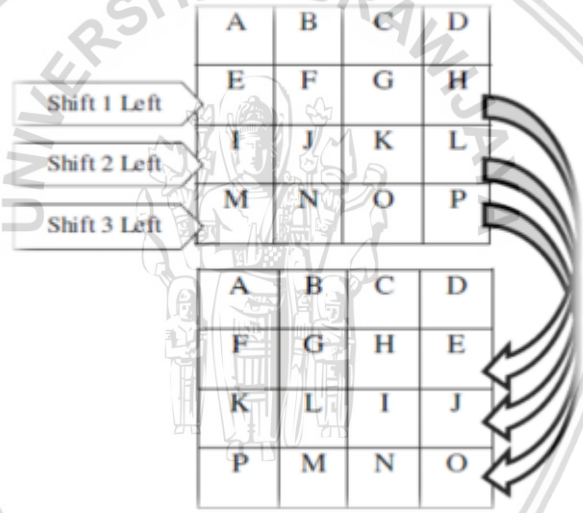


Gambar 2.5 Subtitusi

Sumber: Bhajaj (2016)

2.7.2 *ShiftRows*

Pada proses *shiftrows*, baris pertama pada array disimpan. Tetapi, baris kedua, ketiga dan seterusnya secara siklus digeser satu, dua, dan tiga byte ke kiri (Bhajaj, 2016). Proses tersebut dapat dilihat pada gambar 2.6.

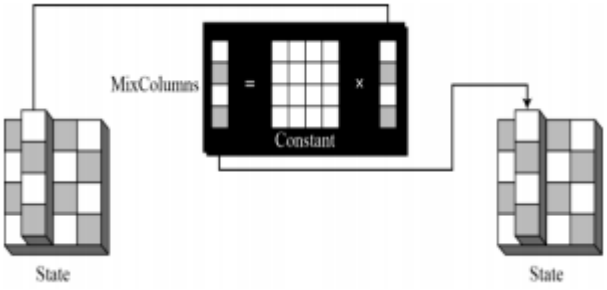


Gambar 2.6 *ShiftRows*

Sumber: Bhajaj (2016)

2.7.3 *MixColumns*

Transformasi *MixColumns* mencakup, kolom dianggap sebagai polinomial di atas GF (28) dan dikalikan dengan modulo x4 + 1 dengan polynomial c(x) tetap, diberikan oleh: c (x) = {03} x3 + {01} x2 + {01} x + {02} [8] (Bhajaj, 2016). Proses *MixColumns* digambarkan pada gambar 2.7.

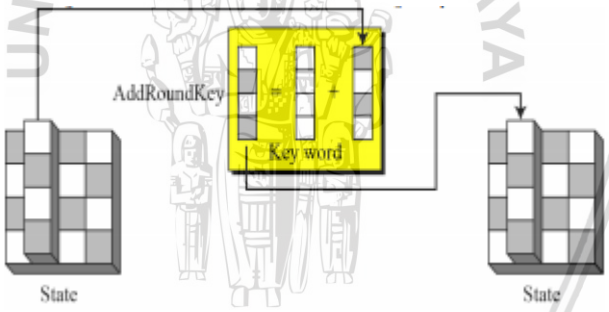


Gambar 2.7 *MixColumns*

Sumber: Bhajaj (2016)

2.7.4 *AddRoundKey*

Dengan menggunakan operasi bitwise exclusive-or (XOR), sebuah kunci putaran ditambahkan ke array. Kunci putaran ini dihitung dalam proses *key expansion* (Bhajaj, 2016). Jika kunci putaran dihitung dengan cepat untuk setiap blok data, maka disebut AES dengan *key expansion* online. Di sisi lain, pada sebagian besar aplikasi, kunci enkripsi tidak berubah sesering data. Akibatnya, sebelum kunci putaran proses enkripsi dapat dihitung dan disimpan selama periode waktu di register lokal atau memori. Perhitungan kunci putaran ini disebut AES dengan *key expansion* offline (Bhajaj, 2016).



Gambar 2.8 *AddRoundKey*

Sumber: Bhajaj (2016)

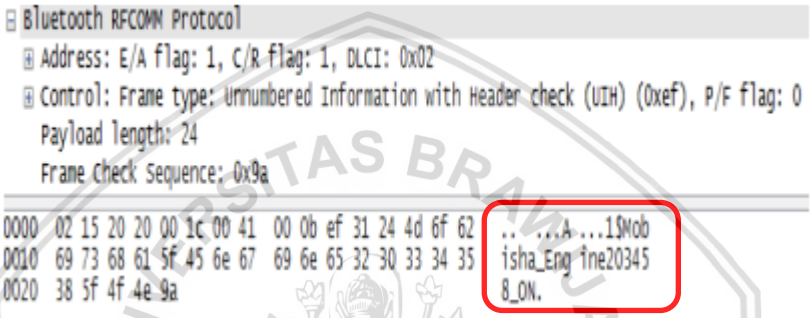
Gambar 2.8 di atas menggambarkan proses *AddRoundKey*. Sedangkan terdapat tiga langkah untuk setiap putaran *key expansion* (Bhajaj, 2016):

1. KeySubWord: Proses keysubword membutuhkan input kata berukuran 4-byte dan output kata dihasilkan dengan mensubtitusikan setiap byte pada input ke byte yang lain berdasarkan S-box.
2. KeyRotWord: Proses keyrotword mengambil kata [a3, a2, a1, a0], melakukan permutasi siklus dan mengembalikan kata [a2, a1, a0, a3] sebagai output.
3. KeyXOR: Pada proses keyXOR, setiap kata w [1] sama dengan XOR dari kata sebelumnya, w[i-1], dan kata Nk posisi sebelumnya, w[i-Nk]. Nk sama dengan 4, 6 atau 8 untuk masing-masing panjang kunci 128, 192 atau 256 bits.

2.8 Pengambilan Data

Pengambilan data bertujuan untuk memperolah data-data mentah yang nantinya diperlukan untuk pengolahan data dan analisis, serta untuk mengetahui bagaimana perbedaan sebelum dan setelah diimplementasikan mekanisme keamanan. Pengambilan data dilakukan dengan cara *sniffing* terhadap pengiriman data pada sistem (Ramdhansya, 2014). Sistem operasi yang digunakan harus dilengkapi dengan *libcap* (*system interface* yang digunakan untuk *packet capture*) (Ramdhansya, 2014). Namun, penelitian ini akan menggunakan *tcpdump*.

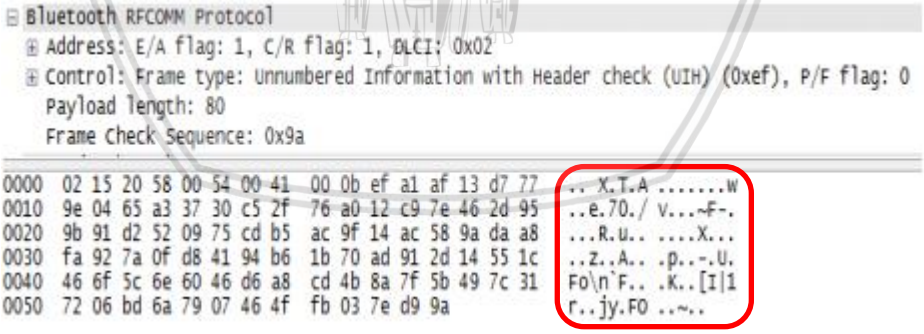
Hasil dari proses *packet capture* akan dianalisa menggunakan aplikasi wireshark, kemudian dapat terlihat perbedaan paket sebelum dan sesudah diimplementasikan mekanisme keamanan.



Gambar 2.9 Contoh paket data sebelum diimplementasi AES

Sumber: Ramdhansya (2014)

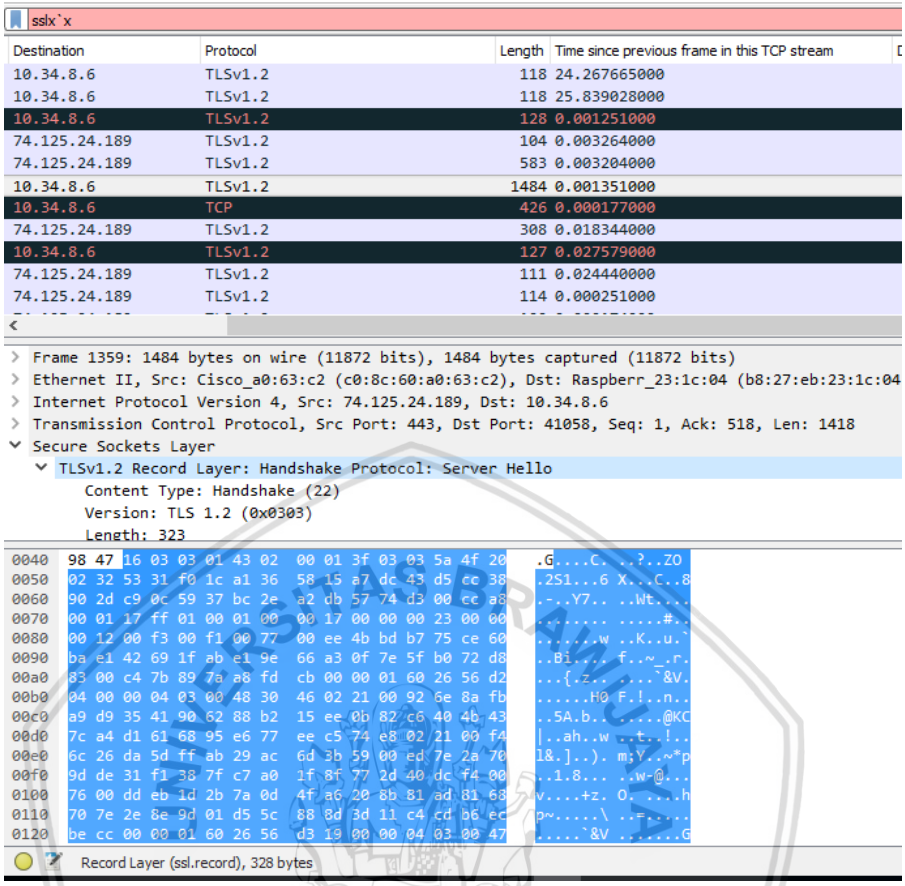
Dari gambar 2.9 diatas dapat terlihat pada bagian yang ditandai bahwa paket yang diuji dapat dengan mudah dibaca, sehingga *replay-attack* dapat dilakukan untuk membobol sistem (Ramdhansya, 2014).



Gambar 2.10 Contoh paket data sebelum diimplementasi AES

Sumber: Ramdhansya (2014)

Dari gambar 2.10 diatas dapat terlihat pada bagian yang ditandai bahwa paket yang diuji tidak mudah dibaca karena pasan-pesan dan paketnya telah dienkripsi oleh algoritma AES (Ramdhansya, 2014). Berdasarkan data tersebut, dapat disimpulkan bahwa pengiriman data telah menjadi lebih aman dan tidak dapat diketahui oleh pihak lain (Ramdhansya, 2014).



Gambar 2.11 Contoh paket data setelah diimplementasi TLS/SSL

Sumber: Tehupeiory (2016)

Gambar 2.11 di atas merupakan contoh hasil pengambilan data pada sistem yang sudah diimplementasikan mekanisme keamanan TLS/SSL.

2.9 Pengujian Kinerja

2.9.1 Kinerja mekanisme keamanan

Pengujian kinerja mekanisme keamanan bertujuan untuk menguji apakah mekanisme keamanan yang digunakan dapat mengamankan komunikasi pada lingkungan uji penelitian. Dalam penelitiannya, Fauzi Ahmad (2016) melakukan ekperimen dengan melakukan pengujian terhadap metode AES-256. Pengujian tersebut terdapat 2 tahap, pengujian menggunakan enkripsi dan pengujian tanpa enkripsi. Pengujian tersebut menggunakan aplikasi *wireshark* untuk melakukan kegiatan *sniffing*. Aplikasi wireshark menangkap data-data yang ada pada pengujian tersebut untuk kemudian di analisa.

Analisa data pada penelitian tersebut dilakukan dengan cara melakukan proses enkripsi dan dekripsi dengan mengolah pesan teks yang dimasukkan oleh *user*. Pesan teks akan dienkripsi dengan algorima *Advanced Encryption Standard* (AES) 256 bit, maka akan menghasilkan pesan sandi (*ciphertext*). Pesan teks

tersebut bersifat *ciphertext* karena pesan tidak dapat dimengerti maknanya. Pesan sandi (*ciphertext*) tersebut dikirim ke *server* untuk proses penyimpanan data. Pada saat inilah proses analisa data dikerjakan karena pada proses ini akan terlihat apakah pesan tersebut sudah berbentuk pesan sandi (*ciphertext*) atau masih pesan asli (*plaintext*) (Fauzi, 2016).

Ahmad Fali Oklilas dan Budi Irawan, dalam penelitiannya (2014), menggunakan mekanisme keamanan TLS untuk mengamankan transfer data. Pengujian dilakukan dengan melakukan *login* ke FTP *server* menggunakan *username* dan *password*. Pengujian tersebut dijalankan bersama dengan *tool* wireshark. Untuk mengetahui apakah data yang dikirimkan aman, dibuka proses *tcp stream* pada wireshark.

Gambar 2.12 Tcp stream pada wireshark

Gambar 2.12 di atas menunjukkan bahwa transfer data belum aman dikarenakan data sangat mudah dibaca.

Gambar 2.13 TCP Stream pada wireshark

Gambar 2.13 di atas menunjukkan isi dari proses *tcp stream* yang sudah diimplementasikan mekanisme enkripsi. Data yang terlihat tidak bisa dibaca dengan jelas. Hal ini berarti data yang dikirimkan telah aman (Fali, 2014).

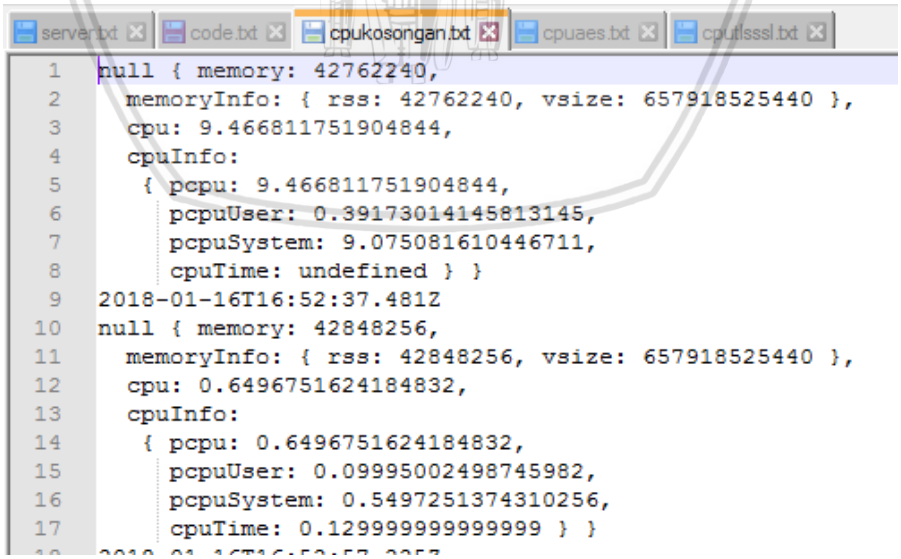
2.9.2 Kinerja penggunaan CPU dan *Memory*

Pengujian kinerja penggunaan CPU dan *memory* bertujuan untuk mengetahui dampak dari implementasi mekanisme keamanan terhadap penggunaan CPU dan *memory* raspberry pi. Cara mengambil data CPU dan *Memory* dari middleware adalah dengan menjalankan program yang digunakan oleh Fahrur (2017) dalam penelitiannya.

```
var usage = require('usage');
var pid = 24571
setInterval(function() {
          var options = { keepHistory:true};
          usage.lookup(pid, options, function(err, stat) {
          console.log(err, stat);
          console.log(new Date().toISOString());
});
},15000);
```

Gambar 2.14 Kode program uji penggunaan CPU dan *memory*

Gambar 2.14 di atas merupakan program yang digunakan pada penelitian M. Fahrur Rozi (2017) untuk mengukur penggunaan CPU dan *memory* pada raspberry. Program tersebut berjalan setiap 15 detik sekali dan mengambil data penggunaan CPU dan *Memory middleware* berdasarkan PID dari *file server.js* yang merupakan berkas utama dari *middleware*. Baris program untuk mengambil data penggunaan CPU dan Memory pada `baris usage.lookup(pid, options, function(err, stat)`. Data *CPU* dan *Memory* dicatat dalam sebuah log file dengan format txt (Fahrur, 2017).



Gambar 2.15 Hasil *log file*

Pada Gambar 2.15 di atas merupakan hasil dari proses pengambilan data penggunaan CPU dan *Memory*. Pada *file* tersebut, terdapat nilai variabel *memory*

dan *cpu.* Variabel inilah yang digunakan untuk mengetahui nilai penggunaan CPU dan *Memory* (Fahrur, 2017).

2.9.3 *Packet loss*

Pengujian *packet loss* digunakan untuk mengetahui pengaruh penggunaan implementasi keamanan terhadap nilai *packet loss* pada komunikasi antara *middleware* dan data center. Dalam penelitian yang dilakukan oleh Fahrur Rozi (2017), performansi *middleware* terhadap *packet loss* dilakukan dengan cara membandingkan data yang seharusnya terkirim dengan data yang benar benar diterima oleh *middleware*.

Data yang diterima *middleware* didapatkan dengan menghitung *packet* yang tercacat pada *log* wireshark. Pengambilan data *packet loss* dilakukan secara manual yaitu dengan menghitung jumlah paket yang ada pada *capture* wireshark. Proses pengiriman data oleh *publisher* dilakukan secara terjadwal yaitu setiap 30 detik sekali sehingga memudahkan untuk mengetahui berapa data yang seharusnya terkirim, dalam satu menit terdapat 2 data yang seharusnya terkirim, dalam satu jam terdapat 120 data yang seharusnya terkirim, dalam 3 jam terdapat 360 data yang seharusnya terkirim oleh 1 *publisher.* Data packet loss didapatkan dengan membandingkan jumlah data yang seharusnya terkirim dengan data yang berhasil terkirim.

2.9.4 Delay

Pengujian *delay* digunakan untuk mengetahui pengaruh penggunaan implementasi keamanan terhadap nilai *delay* pada komunikasi antara *middleware* dan data center. Dalam penelitian yang dilakukan oleh Fahrur Rozi (2017), wireshark memiliki fitur untuk mencatat *delay* yang didapatkan dari *packet capture* yang tercatat dalam sebuah *log* dengan format *pcap*. Didalam log tersebut tercatat waktu kapan paket dikirimkan dan kapan paket menenerima ACK sehingga *delay* yang dicatat pada wireshark diambil dari rentan waktu paket dikirimkan sampai menerima ACK, dalam aplikasi wireshark, nilai tersebut diberi label sebagai *delta time*. *Delta Time* ini adalah sebuah nilai pada aplikasi wireshark yang menunjukkan jeda waktu saat paket dikirim sampai paket menerima ack. Proses perhitungan jeda waktunya yaitu waktu saat ack diterima dikurangi waktu saat paket dikirimkan. *Delta Time* inilah yang menjadi sumber data untuk menghitung rata rata *delay* di setiap pengujian skenario berlangsung.

	Delta Time	Info
	0.000309	3000 → 33742
	0.000256	3000 → 33742
	0.007596	3000 → 33742
	0.000199	3000 → 33742
67c8de…	0.006046	3000 → 33726
dc0b09…	0.002951	3000 → 33726
d3dcc7…	0.001801	3000 → 33726
8da5ae…	0.003779	3000 → 33726
	0.000322	3000 → 33746
	0.000144	3000 → 33746
	0.004773	3000 → 33746

Gambar 2.16 Delta time

Gambar 2.16 di atas merupakan nilai *delta time* yang terdapat pada kolom di wireshark. Nilai tersebut yang digunakan sebagai nilai *delay*.

2.9.5 Jitter

Pengujian *jitter* digunakan untuk mengetahui pengaruh penggunaan implementasi keamanan terhadap nilai *jitter* pada komunikasi antara *middleware* dan data center. *Jitter* atau variasi kedatangan paket diakibatkan oleh variasi-variasi dalam panjang antrian, dalam waktu pengolahan data, dan juga dalam waktu penghimpunan ulang paket-paket di akhir perjalanan *jitter*. *Jitter* lazimnya disebut variasi *delay* berhubungan erat dengan *latency*, yang menunjukkan banyaknya variasi *delay* pada taransmisi data di jaringan. *Delay* antrian pada *router* dan *switch* dapat menyebabkan jitter (Setiawan, 2017). Penghitungan nilai jitter dilakukan pada aplikasi Microsoft Excel menggunakan rumus berikut:

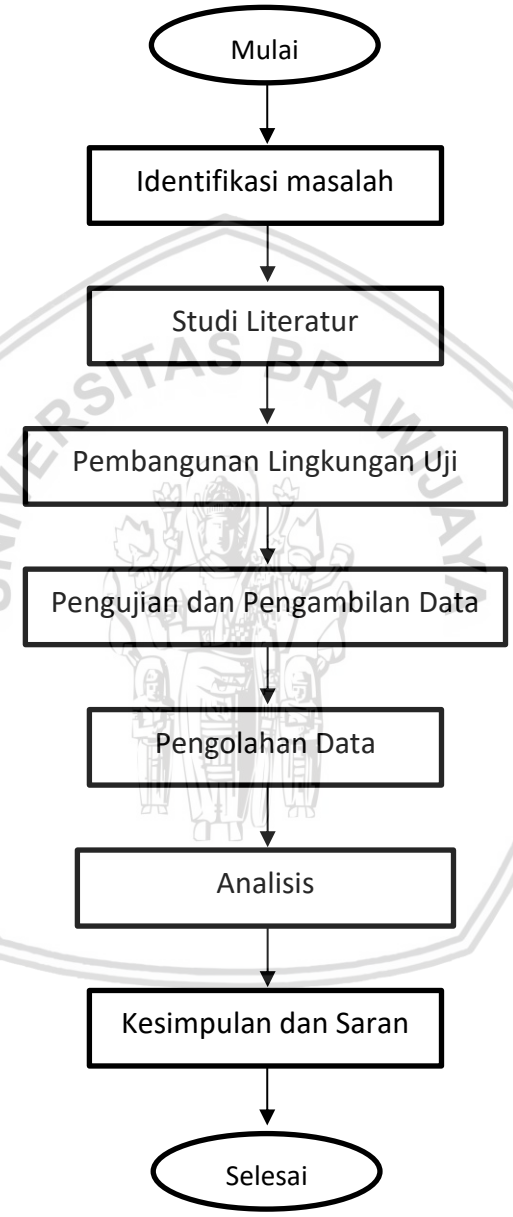
$$jitter = \frac{total\ variasi\ delay}{total\ paket\ yang\ diterima - 1} \quad (2.1)$$

Total variasi delay merupakan jumlah dari selisih tiap nilai *delay*, dengan rumus perhitungan:

$$\text{Total variasi delay} = (\text{delay 2} - \text{delay 1}) + (\text{delay 3} - \text{delay 2}) + \cdots + (\text{delay n} - \text{delay (n}-\text{)}) \quad (2.2)$$

BAB 3 METODOLOGI PENELITIAN

Pada bab metodologi penelitian ini akan dibahas tentang studi literatur, persiapan testbed, pengujian dan pengolahan data, analisa dan pembahasan, kesimpulan dan saran. Diagram alir penelitian ini digambarkan pada Gambar 3. 1.



Gambar 3.1 Diagram Alir Penelitian

3.1 Identifikasi Masalah

Identifikasi masalah dilakukan untuk menentukan penelitian apa yang akan dilakukan. Dengan melihat pada kondisi saat ini serta melihat masalah apa yang terjadi dan yang mungkin terjadi di lingkungan. Dengan adanya indentifikasi masalah, penelitian akan menjadi fokus dan memiliki tujuan serta manfaat yang jelas.

3.2 Studi Literatur

Studi literatur digunakan untuk mempelajari tentang landasan kepustakaan yang digunakan untuk mendukung penulisan penelitian. Landasan kepustakaan tersebut didapatkan dari beberapa sumber seperti: buku, *e-book*, skripsi, jurnal, artikel dan *website*. Landasan kepustakaan yang digunakan untuk mendukung penulisan penelitian ini sebagai berikut:

1. *Internet of Things* (IoT)
2. *Middleware* IoT
3. *End-to-end Security*
4. HTTP dan HTTPS
5. TLS/SSL
6. AES-256
7. Pengambilan Data
8. Pengujian Kinerja

3.3 Pembangunan Lingkungan Uji

Pembangunan lingkungan uji merupakan tahap yang digunakan penulis untuk menyiapkan berbagai kebutuhan yang diperlukan untuk melakukan penelitian. Kebutuhan-kebutuhan tersebut dapat dibagi menjadi kebutuhan perangkat keras, kebutuhan perangkat lunak dan *middleware*. Kebutuhan perangkat keras berisikan daftar perangkat-perangkat yang digunakan dalam penelitian, terutama perangkat fisik. Kebutuhan perangkat lunak berisikan daftar program yang digunakan dalam penelitian. Implementasi *middleware* juga dilakukan pada tahap ini.

Sebelum melakukan implementasi *middleware*, diperlukan instalasi paket-paket yang mendukung *middleware* tersebut terlebih dahulu. Instalasi tersebut meliputi instalasi Redis dan Node.js. Setelah instalasi kedua paket tersebut selesai, maka implementasi *middleware* baru dilakukan. *Middleware* harus dipastikan sudah berjalan dengan benar sebelum manjutkan ke tahap selanjutnya, yaitu implementasi mekanisme keamanan. Untuk itu, pengujian dilakukan dengan cara *middleware* mengirimkan data ke data center, jika data berhasil diterima oleh data center, maka *middleware* sudah berjalan dengan benar dan bisa dilanjutkan ke implementasi mekanisme keamanan.

Implementasi mekanisme keamanan dibagi menjadi dua bagian, yaitu implementasi TLS/SSL dan Implementasi AES-256. Implementasi dilakukan dengan menambahkan *source code* ke dalam *file* pada *middleware* dan data center.

1. Implementasi TLS/SSL

   Mekanisme keamanan RLS/SSL menggunakan sertifikat dalam prosesnya, maka perlu membuat sertifikat terlebih daulu. Penulis menggunakan *self-signed certificate* yang merupakan sertifikat yang dibuat sendiri. Setelah sertifikat berhasil dibuat, maka dilanjutkan ke tahap implementasi. Implementasi TLS/SSL mengubah *source code* pada *file server.js* pada *middleware* dan *file client.js* pada data center.

2. Implementasi AES-256

   Implementasi AES-256 mengubah *source code* pada *file server.js* pada *middleware* dan *file client.js* pada data center.

3.4 Pengujian dan Pengambilan Data

Pengujian dilakukan untuk mengetahui apakah *middleware* mampu berfungsi dengan benar berdasarkan skenario-skenario yang telah dipersiapkan oleh penulis. Pengambilan data dilakukan untuk mengetahui apakah mekanisme yang dipasang, yaitu TLS/SSL dan AES-256 berhasil mengamankan pengiriman data antara middleware dengan data center. Pengambilan data dilakukan menggunakan program *tcpdump* dengan fungsi *capture packet*-nya. Dengan wireshark, peneliti dapat mengetahui paket-paket yang dikirimkan dan diterima dari/ke middleware. Data-data yang didapat dari wireshark akan dicatat untuk dianalisis di tahap berikutnya.

3.4.1 Pengujian Pengiriman Pesan

Pengujian dilakukan dengan *middleware* mengirimkan pesan ke data center. Jika *middleware* dapat mengirim pesan dan data center dapat menerimanya, maka pengujian dianggap berhasil. Pesan yang dikirimkan adalah pesan *dummy* atau pesan palsu. Pesan palsu tersebut dibuat di dalam *file websocket_api.js* dengan cara menambahkan variabel baru.

```
msg = '{"sensor”: {"module":"dht22", "tipe":"esp8266",
"index":1751566, "ip":"192.168.222.222"}, "protocol":"coap",
"topic":"home\/kitchen", "humidity”: {"value":75,"unit":"%"},
"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature”:
{"value":30,"unit":"celcius"}}'
```

Gambar 3.2 Pesan palsu

Alasan penulis menggunakan pesan palsu adalah untuk mengurangi factor-faktor yang dihasilkan dari pengiriman data dari sensor ke *middleware*, seperti jeda pengiriman data, serta faktor lain seperti kemungkinan bahwa sensor bisa mati atau berhenti mengirimkan data. Pesan palsu tersebut dibuat berdasarkan pesan yang dikirimkan dari sensor, tetapi penulis membuat perbedaan pada bagian *IP*, yaitu mengubah *IP* pada pesan palsu menjadi 192.168.222.222. Pengujian ini dilakukan menggunakan tiga skenario.

1. *Skenario 1: Tanpa mekanisme keamanan*

Pengujian skenario 1 adalam pengiriman pesan tanpa menggunakan mekanisme keamanan, yang merupakan kondisi awal dari *middleware*. Pengiriman pesan dilakukan oleh *middleware*, jika data center mampu menerima dan menampilkan data yang dikirim oleh *middleware* sebagai bukti data seperti pesan palsu di atas, maka pengujian dianggap berhasil.

2. *Skenario 2: Menggunakan mekanisme keamanan AES-256*

Pengujian skenario 2 adalam pengiriman pesan menggunakan mekanisme keamanan AES-256. Pengiriman pesan dilakukan oleh *middleware*, jika data center mampu menerima dan menampilkan data yang dikirim oleh *middleware* dan pesan yang ditampilkan merupakan pesan enkripsi, maka pengujian dianggap berhasil.

3. *Skenario 3: Menggunakan mekanisme keamanan TLS/SSL*

Pengujian skenario 2 adalam pengiriman pesan menggunakan mekanisme keamanan TLS/SSL. Pengiriman pesan dilakukan oleh *middleware*, jika data center mampu menerima dan menampilkan data yang dikirim oleh *middleware* dan pesan yang ditampilkan merupakan pesan enkripsi, maka pengujian dianggap berhasil.

3.4.2 Pengambilan Data

Pengambilan data ini bertujuan untuk mengetahui apakah mekanisme keamanan yang digunakan berhasil diimplementasikan dan untuk mengetahui perbedaan hasil dari pengiriman data berdasarkan beberapa skenario. Metode pengambilan data dilakukan dengan cara *middleware* dibuat untuk mengirimkan pesan secara terus menerus ke data center. Pesan yang dikirimkan merupakan pesan *dummy* atau pesan palsu. Menggunakan pesan palsu bersama dengan fungsi *setInterval*, penulis dapat membuat kondisi dimana *middleware* akan mengirimkan data terus-menerus dengan jeda waktu antar pengiriman yang sama.

```
setInterval(function(){
       //define process
},10000)
```

Gambar 3.3 Fungsi setInterval

Proses pengambilan data dilakukan berdasarkan skenario yang sama dengan pengujian. Pada bagian ini, penulis melakukan pengambilan data yang dibutuhkan untuk analisis. Pengambilan data dilakukan menggunakan program bernama *tcpdump*, yang merupakan program untuk melakukan *capture packet* yang ada pada suatu sistem atau jaringan. Proses *capture packet* akan menghasilkan *file* berformat *pcap*, yang kemudian dapat dibuka pada *wireshark*. Pengambilan data dilakukan mengikuti skenario-skenario yang telah ditentukan pada bab sebelumnya. Adapun perintah yang digunakan untuk melakukan *capture packet* menggunakan program *tcpdump* adalah:

```
$ sudo timeout 600 tcpdump -i eth0 -w kosong1.pcap
```

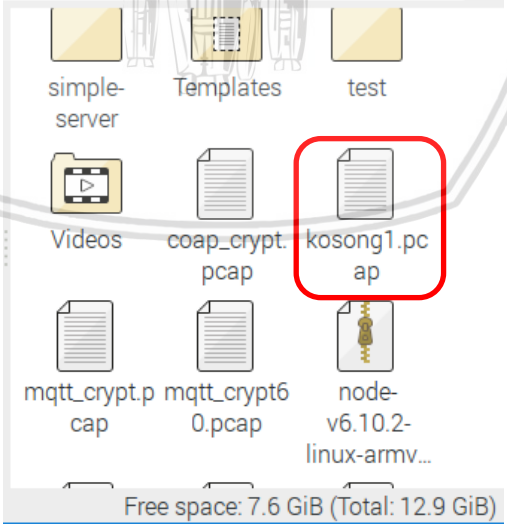
Gambar 3.4 Perintah *capture packet* menggunakan *tcpdump*

Gambar 3.4 di atas menunjukkan perintah yang digunakan untuk melakukan *capture packet* menggunakan program *tcpdump*. Terdapat beberapa bagian pada perintah tersebut. Bagian pertama adalah *sudo*, merupakan perintah yang digunakan untuk melakukan akses *root* serta menjalankan perintah selanjutnya sebagai *superuser*. Bagian kedua adalah *timeout 600*, merupakan perintah untuk menentukan berapa lama operasi akan berjalan menggunakan satuan detik. Jadi berdasarkan perintah di atas, operasi akan berjalan selama 10 menit. Bagian selanjutnya adalah *tcpdump,* merupakan deklarasi program atau operasi apa yang akan dijalankan. Bagian *-i eth0* merupakan perintah untuk menentukan interface mana yang akan di-*capture* paketnya, penelitian ini menggunakan interface *eth0* yakni *Ethernet*. Bagian terakhir adalah *-w kosong1.pcap* merupakan perintah untuk menyatakan nama *file* yang akan dibuat dari proses *capture packet* tersebut. Gambar di bawah ini menunjukkan contoh proses *capture packet* yang dilakukan penulis.

```
pi@raspberrypi:~ $ sudo timeout 600 tcpdump -i eth0 -w kosong1.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
es
```

Gambar 3.5 *Capture packet*

Pada Gambar 3.5 di atas, terlihat komentar bahwa program *tcpdump* sedang mendengarkan pada interface *eth0*. Pada saat ini program sedang melakukan penangkapan pesan yang melewati interface tersebut selama 10 menit. Ketika sudah lewat 10 menit, maka proses akan berhenti.



Gambar 3.6 *File* hasil *capture packet*

Gambar 3.6 di atas menunjukkan *file* hasil dari *capture packet* menggunakan program *tcpdump* yang dijalankan pada *middleware*. File tersebut kemudian dapat dibuka menggunakan aplikasi *wireshark*.

1. *Skenario 1: Pengiriman data tanpa mekanisme keamanan*

Skenario pertama adalah pengiriman data tanpa menggunakan mekanisme keamanan atau pengiriman data menggunakan sistem dasar dari arsitektur middleware yang diteliti. Pengambilan data dilakukan dengan mengirimkan data terus-menerus selama 10 menit bersama dengan program *tcpdump* untuk menyimpan data-data aktivitas selama pengiriman data. Hasil dari *tcpdump* akan dibuka menggunakan wireshark, data di dalam wireshark akan di-*filter*, kemudian diolah datanya.

2. *Skenario 2: Pengiriman data menggunakan mekanisme keamanan AES-256*

Skenario kedua adalah pengiriman data menggunakan mekanisme keamanan TLS/SSL. Pengambilan data dilakukan dengan mengirimkan data terus-menerus selama 10 menit bersama dengan program *tcpdump* untuk menyimpan data-data aktivitas selama pengiriman data. Hasil dari *tcpdump* akan dibuka menggunakan wireshark, data di dalam wireshark akan di-*filter*, kemudian diolah paket datanya.

3. *Skenario 3: Pengiriman data menggunakan mekanisme keamanan TLS/SSL*

Skenario kedua adalah pengiriman data menggunakan mekanisme keamanan TLS/SSL. Pengambilan data dilakukan dengan mengirimkan data terus-menerus selama 10 menit bersama dengan program *tcpdump* untuk menyimpan data-data aktivitas selama pengiriman data. Hasil dari *tcpdump* akan dibuka menggunakan wireshark, data di dalam wireshark akan di-*filter*, kemudian diolah paket SSL-nya serta paket datanya.

4. *Pengambilan Data Penggunaan CPU dan Memory*

Cara mengambil data CPU dan Memory dari middleware adalah dengan menjalankan program berikut:

```
var usage = require('usage');
var pid = 24571
setInterval(function() {
          var options = { keepHistory:true};
          usage.lookup(pid, options, function(err, stat) {
          console.log(err, stat);
          console.log(new Date().toISOString());
});
},20000);
```

Gambar 3.7 Kode program uji penggunaan CPU dan memory

Program tersebut berjalan setiap 20 detik sekali dan mengambil data penggunaan CPU dan *Memory middleware* berdasarkan PID dari *file server.js* yang merupakan berkas utama dari *middleware*. Baris program untuk mengambil data penggunaan CPU dan Memory pada `baris usage.lookup(pid, options, function(err, stat)`. Data *CPU* dan *Memory* dicatat dalam sebuah log file dengan format txt. Pengambilan data penggunaan CPU dan *memory* dilakukan sebanyak 5 kali untuk setiap skenario.

3.5 Pengolahan Data dan Analisis

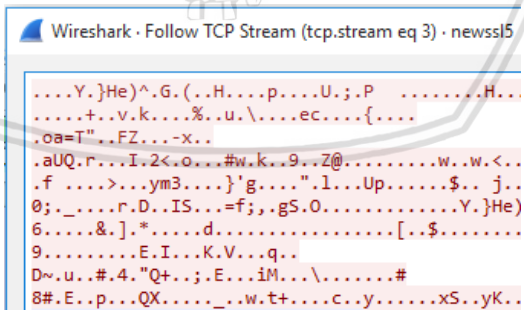
Agar data yang didapatkan dari kegiatan di atas dapat dianalisa, maka data tersebut perlu diolah sedemikian rupa agar mudah dibaca oleh penulis. Hasil *packet capture* saat pengambilan data akan dibuka pada aplikasi wireshark, kemudian penulis menggunakan fungsi filter untuk menampilkan data yang dibutuhkan untuk analisis saja. Hasil dari filter tersebut akan disimpan dengan format file *Microsoft excel* yang selanjutnya akan diolah sesuai dengan kebutuhan penelitian. Pada bagian ini juga akan dilakukan penghitungan penggunan CPU dan *memory, packet loss*, *delay, dan jitter*.

3.5.1 Kinerja Mekanisme Keamanan

Pengolahan data kinerja mekanisme keamanan dilakukan menggunakan fungsi *follow tcp stream* pada wireshark. Fungsi ini berguna untuk melihat isi dari pesan yang dikirimkan.

Gambar 3.8 TCP Stream

Gambar 3.8 di atas menunjukkan hasil *follow tcp stream*. Berdasarkan gambar tersebut, data terlihat mudah dibaca. Hal ini berarti komunikasi datanya belum aman.

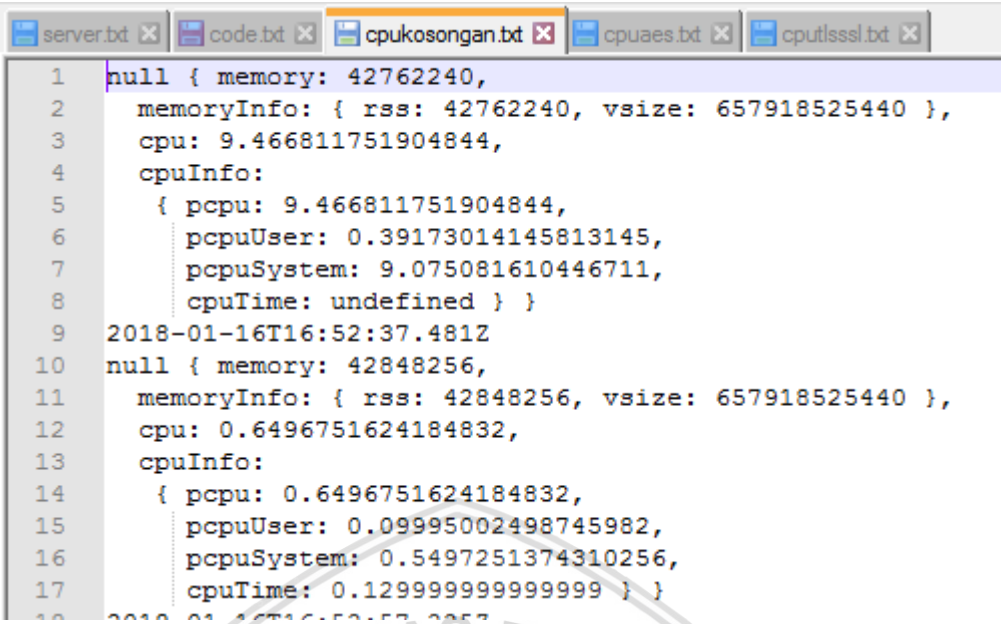


Gambar 3.9 TCP Stream

Gambar 3.9 di atas menunjukkan hasil *follow tcp stream* yang lain. Berdasarkan gambar tersebut, data tidak dapat dibaca atau dipahami maksudnya. Hal ini berarti komunikasi datanya sudah aman.

3.5.2 Penggunaan CPU dan Memory

Pengolahan data penggunaan CPU dan Memory dilakukan dengan membuka *file* hasil dari pengambilan data.



Gambar 3.10 Hasil pengambilan data penggunaan CPU dan Memory

Pada Gambar 3.10 di atas, terlihat beberapa variabel yang memiliki nilai tertentu. Variabel yang dibutuhkan adalah *memory* dan CPU, kedua nilai pada variabel tersebut diambil untuk dianalisa pada tahap selanjutnya.

3.5.3 Packet Loss

Pengujian *packet loss* digunakan untuk mengetahui pengaruh penggunaan implementasi keamanan terhadap nilai *packet loss* pada komunikasi antara *middleware* dan data center. Pengambilan nilai *packet loss* dilakukan secara manual yaitu dengan menghitung dari packet yang ada pada *capture* wireshark. Proses pengiriman data oleh *middleware* dilakukan secara terjadwal yaitu setiap 10 detik sekali sehingga memudahkan untuk mengetahui berapa data yang seharusnya terkirim, dalam satu menit terdapat 6 data, dan dalam sepuluh menit seharusnya terdapat 60 data yang seharusnya terkirim oleh *middleware*. Data packet loss didapatkan dengan membandingkan jumlah data yang seharusnya terkirim dengan data yang berhasil terkirim. Rumus yang digunakan untuk menghitung *packet loss* adalah:

$$Packet\ loss = \frac{x-y}{x}\ x\ 100 \qquad (3.1)$$

Variabel x merupakan jumlah pesan yang seharusnya dikirimkan. Variabel y adalah jumlah pesan yang diterima.

3.5.4 Delay

Delay adalah parameter yang intrinsic pada suatu jaringan, mengingat *end-point* dari suatu jaringan pasti berjarak dan informasi akan membutuhkan waktu untuk sampai ke ujung lain dari sebuah jaringan. Pengolahan data untuk *delay* dilakukan dengan mengambil nilai *Delta time* pada hasil *capture packet* di dalam *wireshark*. Didalam log tersebut tercatat waktu kapan paket dikirimkan dan

kapan paket menenerima ACK sehingga *delay* yang dicatat pada wireshark diambil dari rentan waktu paket dikirimkan sampai menerima ACK. Hasil *capture* aplikasi wireshark di setiap skenario terdapat kolom *Delta Time*, *Delta Time* ini adalah fitur dari aplikasi wireshark untuk menghitung jeda waktu saat packet dikirim sampai packet menerima ack. Setelah nilai *delta time* didapatkan, maka akan dicari rata-ratanya.

3.5.5 Jitter

Jitter adalah bentuk variasi dari *delay* yang erat kaitannya dengan kedatangan paket. *Jitter* disebabkan oleh panjang queue dalam satu pengolahan data dan reassemble data di akhir pengiriman. Nilai *jitter* didapatkan dengan rumus berikut:

$$jitter = \frac{total\ variasi\ delay}{total\ paket\ yang\ diterima - 1} \quad (3.2)$$

Total variasi *delay* merupakan jumlah dari selisih tiap nilai *delay*, dengan rumus perhitungan:

$$\text{Total variasi delay} = (\text{delay } 2 - \text{delay } 1) + (\text{delay } 3 - \text{delay } 2) + \cdots + (\text{delay n} - \text{delay } (n-)) \quad (3.3)$$

3.5.6 Analisis

Setelah proses pengambilan data selesai dan telah didapatkan data yang dibutuhkan, maka akan dilakukan analisis hasil dan pembahasan. Berdasarkan analisis tersebut, peneliti dapat menentukan apakah mekanisme keamanan TLS/SSL dan AES-256 berhasil mengamankan pengiriman data antara *middleware* dan *data center* serta manakah yang memiliki efisiensi lebih baik dibandingkan yang lain.

3.6 Kesimpulan dan Saran

Bagian terakhir dari penelitian ini adalah kesimpulan dan saran. Kesimpulan dibuat untuk memberi jawaban terhadap rumusan masalah yang tertulis pada penelitian ini. Penarikan kesimpulan dilakukan berdasarkan analisis dan pembahasan yang dilakukan sebelumnya. Dan peneliti akan memberikan saran yang dapat dijadikan rujukan untuk memperbaiki atau mengembangkan penelitian ini maupun penelitian yang sejenis.

BAB 4 PEMBANGUNAN LINGKUNGAN UJI

Pembangunan lingkungan uji merupakan tahap yang digunakan penulis untuk menyiapkan berbagai kebutuhan yang diperlukan untuk melakukan penelitian. Penelitian ini dilakukan di Lab Jaringan Komputer, Fakultas Ilmu Komputer. Kebutuhan-kebutuhan penelitian dapat dibagi menjadi kebutuhan perangkat keras, kebutuhan perangkat lunak dan middleware.

4.1 Kebutuhan Perangkat Keras

Bagian ini menjelaskan berbagai macam perangkat keras yang digunakan pada penelitian. Kebutuhan perangkat keras pada penelitian ini adalah sebagai berikut:

1. Laptop dengan spesifikasi:

Merk	: ASUS-N43SL
RAM	: 4 GB
Sistem Operasi	: Windows 10 Pro 64-bit (10.0, Build 14393)

2. Raspberry Pi 2

Chipset	: Broadcom BCM2836
CPU	: 900 MHz quad-core ARM Cortex A7
RAM	: 1 GB
Sistem Operasi	: Raspbian Jessie

3. Server

OS	: Linux
IP	: 10.34.0.21
Lokasi	: Fakultas Ilmu Komputer Universitas Brawijaya

4.2 Kebutuhan Perangkat Lunak

Bagian ini menjelaskan berbagai macam perangkat lunak yang digunakan pada penelitian. Kebutuhan perangkat lunak pada penelitian ini adalah sebagai berikut:

1. Wireshark v2.2.7

   Aplikasi yang digunakan untuk menganalisa paket data yang direkam pada middleware menggunakan *tcpdump.*

2. Putty v0.69

   Aplikasi ini digunakan untuk lakukan koneksi ssh ke middleware dan data center.

3. Node.js v6

Program yang berfungsi untuk menjalankan kode-kode *javascript* pada middleware dan data center.

4. Npm

   Sebuah *package manager* yang digunakan untuk melakukan instalasi Nodejs v6.

5. Tcpdump

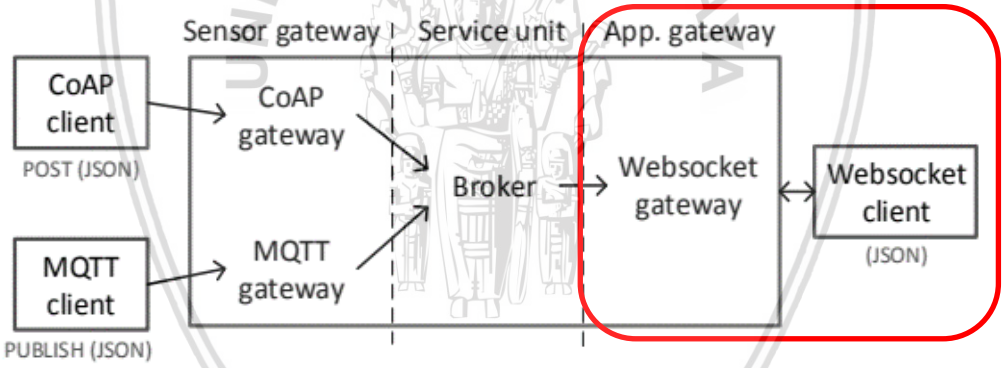
   Program yang digunakan untuk merekam paket pada middleware selama penelitian dilakukan.

6. Microsoft Excel 2016

   Aplikasi yang digunakan untuk mengolah data berupa angka.

4.3 Middleware

Tujuan utama dari penelitian ini adalah menyediakan mekanisme keamanan pada protokol HTTP untuk IoT *Middleware*. *Middleware* yang digunakan pada penelitian ini adalah *middleware* yang dikembangkan oleh Husnul Anwari dalam penelitiannya (Anwari, 2017). Arsitektur dari *middleware* tersebut adalah sebagai berikut:



Gambar 4.1 Arsitektur Middleware

Sumber: Anwari (2016)

Seperti yang terlihat pada Gambar 4.1 diatas, arsitektur *middleware* yang dipakai oleh penulis menggunakan pola *publish-subscribe*. *Middleware* tersebut memiliki tiga komponen utama, yaitu *sensor gateway, service unit,* dan *application gateway*. Sensor *gateway* berfungsi untuk menyediakan interface bagi sensor dalam mengirimkan data ke *middleware*. *Service unit* merupakan komponen yang menyediakan tiga fungsi utama, yaitu *data management, service delivery* dan *interface definition*. Komponen ini bekerja bersamaan dengan *Redis* untuk menjadi broker, menjembatani komunikasi antara *application gateway* dan sensor. Service unit juga menyediakan *interface* pada *application gateway* dan sensor untuk melakukan *publish* dan *subscribe* serta berinteraksi dengan Redis. *Application gateway* berfungsi untuk menyediakan *interface* bagi aplikasi supaya bisa terhubung dengan *middleware* serta membaca data yang telah dikirimkan

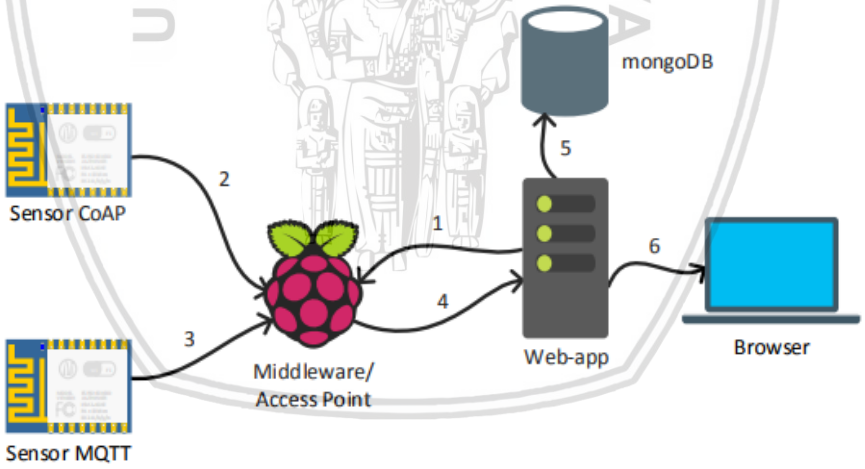
dari sensor melalui protokol Websocket. *Interface* yang disediakan oleh *application gateway* adalah *interface* untuk *connect* ke *middleware*, serta *interface* yang digunakan untuk *subscribe* ke suatu topik.

Implementasi *middleware* yang digunakan dalam penelitian ini dilakukan dengan beberapa tahap. Dimulai dengan instalasi Redis dan instalasi Node.js sebagai paket yang mendukung middleware, ketika instalasi kedua paket tersebut berhasil, maka bisa dilanjutkan dengan *cloning middleware*.

Bagian yang ditandai pada Gambar 4.1 di atas merupakan batasan lingkungan uji yang diteliti. Penulis meneliti jaringan antara *app gateway* dan *websocket client*. Instalasi *middleware* dapat dilihat pada Lampiran A.1.

4.3.1 Alur Komunikasi Middleware

Dalam sistem ini terdapat tiga komponen yang saling berinteraksi yakni sensor suhu dan kelembapan sebagai *publisher*, Raspberry Pi sebagi *middleware* dan laptop untuk menjalankan aplikasi web sebagai *subscriber*. Interaksi antar komponen dalam sistem dibagi menjadi dua yakni pengiriman data dari sensor ke *middleware*. Interaksi kedua yakni pengiriman data dari *middleware* ke aplikasi web. Setiap kali sensor mengirimkan data, maka Websocket akan menerima data tersebut secara *real-time*. Perancangan alur sistem menggunakan *middleware* multi-protokol yang dikembangkan dapat dilihat pada Gambar 4.2 berikut:



Gambar 4.2 Alur komunikasi middleware

Keterangan pada Gambar 4.2 di atas adalah :

1. Aplikasi *subscribe* topik `home/kitchen` dan `home/garage` ke middleware.
2. Sensor CoAP mengirimkan data suhu dan kelembapan dengan topik `home/kitchen`.
3. Sensor MQTT mengirimkan data suhu dan kelembapan dengan topik `home/garage`.
4. Aplikasi menerima data suhu dan kelembapan untuk topik `home/kitchen` dan `home/garage` yang dikirimkan oleh sensor
5. Aplikasi menyimpan data tersebut pada basis data MongoDB.

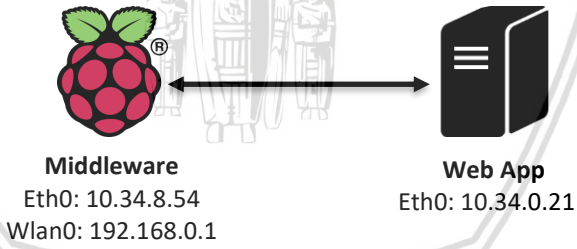
6. Aplikasi diakses menggunakan browser pada sebuah laptop.

Dalam sistem ini terdapat dua jenis sensor yang digunakan yakni sensor berbasis CoAP dan sensor berbasis MQTT. Sensor CoAP akan mengirimkan data suhu dan kelembapan ke *middleware* dengan topik `home/kitchen` dengan mengirimkan *POST request* ke *midleware*. Sedangkan sensor MQTT akan mengirimkan data suhu dan kelembapan dengan topik `home/garage` dengan melakukan publish ke *middleware*. Sebelum sensor dapat mengirimkan data, masing-masing sensor harus terhubung dengan *Access point* dari Raspberry Pi terlebih dahulu. Setiap kali sensor mengirimkan data, *middleware* akan mengirimkan data tersebut ke aplikasi web secara *real-time* menggunakan protokol Websocket. Sebelum aplikasi web dapat menerima data, aplikasi web harus terhubung terlebih dahulu dengan *middleware* melalui jaringan dan *subscribe* pada topik yang dikirimkan oleh sensor. Setelah data diterima oleh aplikasi web, data tersebut selanjutnya disimpan dalam basis data MongoDB.

4.4 Topologi Jaringan

Agar sistem dapat berjalan sesuai dengan harapan, maka perlu dibuat topologi jaringan antar komponen dalam sistem. Topologi jaringan ini menggambarkan bagaimana *middleware* berkomunikasi ke *web-app*. Topologi jaringan dapat dilihat pada Gambar 4.3.

Diperlukan *wireless adapter* yang difungsikan sebagi *access point* dan juga USB to LAN sehingga Raspberry Pi memiliki 2 buah *interface* yaitu wlan0 dengan IP 192.168.0.1 dan eth0 yang terhubung dengan jaringan LAN dengan IP 10.34.8.54. *Middleware* yang dikembangkan menggunakan port: 3000 untuk websocket.



Gambar 4.3 Topologi Jaringan

Subscriber yang digunakan dalam penelitian ini yaitu sebuah aplikasi web. Aplikasi web ini dijalankan pada server dengan IP 10.34.0.21 yang terhubung dengan *interface* eth0 pada *middleware*.

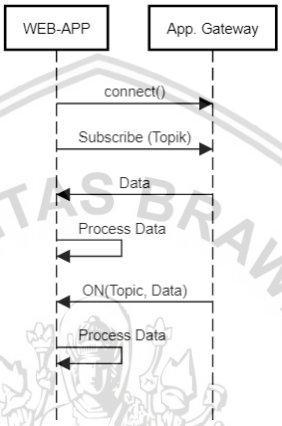
4.5 Aliran Data

Pada penelitian ini, aliran data terjadi antara *web-app* dan *application gateway*. *Application gateway* menyediakan *interface* bagi aplikasi untuk membaca data dari *middleware* melalui protokol Websocket. *Interface* yang disediakan yakni *interface* untuk connect ke *middleware* dan interface untuk *subscribe* suatu topik. Penjelasan tentang masing-masing *interface* tersebut dijabarkan di bawah ini :

Tabel 4.1 API pada application gateway

API	Parameter	Keterangan
`Connect()`		Terhubung dengan application gateway menggunakan websocket
`Subscribe(topik)`	Topik: nama topik	Subscribe ke salah satu topik tertentu

Interaksi yang terjadi antara *application gateway* dengan *service unit* pada penelitian ini digambarkan pada diagram berikut :



Gambar 4.4 Sequence diagram *application gateway*

Ketika sebuah aplikasi hendak meminta data, pertama aplikasi tersebut harus terhubung dengan *middleware* melalui *application gateway* menggunakan *interface connect*. Selanjutnya, ketika aplikasi *subscribe* ke sebuah topik, karena pada penelitian ini menggunakan data palsu, maka *application gateway* akan langsung mengirimkan data. Aplikasi akan menerima data secara terus menerus sesuai dengan *interval* yang sudah ditentukan. Data tersebut berikutnya dapat di proses lebih lanjut oleh aplikasi.

4.6 Implementasi Mekanisme keamanan

Bagian ini dilakukan setelah *middleware* sudah berjalan dengan benar. Implementasi mekanisme keamanan ini berupa mengubah source code pada *middleware* dan data center. Mekanisme keamanan yang digunakan adalah TLS/SSL dan AES-256.

4.6.1 Implementasi TLS/SSL

Implementasi TLS/SSL mengubah source code *middleware* pada file *server.js* dan data center pada file *client.js*. Adapun perubahan yang dilakukan adalah:

1. Server.js

   File server.js berada di dalam *middleware*. *Source code* tambahan diberikan untuk implementasi TLS/SSL dapat dilihat pada Gambar 4.5.

```
Server.js
1  https = require('https')
2
3  var options = {
4      key:    fs.readFileSync('/home/pi/qoap-middleware/server-
5  key-deny.pem'),
6      cert:   fs.readFileSync('/home/pi/qoap-middleware/server-
7  cert-deny.pem'),
8      ca: fs.readFileSync('/home/pi/qoap-middleware/server-csr-
9  deny.pem'),
10     rejectUnauthorized: true,
11   }
12
13 server = https.createServer(options, app)
```

Gambar 4.5 *Source code* implementasi TLS/SSL pada *server.js*

Langkah pertama yang dilakukan adalah memanggil *library https* menggunakan kode `https = require('https')`. Hal ini memungkinkan *script server.js* untuk membuat atau menyambungkan koneksi menggunakan protokol https. Variabel *options* merupakan sebuah array digunakan untuk memanggil kunci serta sertifikat yang dibutuhkan untuk membuat koneksi menggunakan TLS/SSL. Untuk kunci menggunakan file *server-key-deny.pem*, untuk sertifikat menggunakan file *server-cert-deny.pem*, file *server-csr-deny.pem* digunakan untuk meminta dan memvalidasi sertifikat. Sedangkan variabel *server* digunakan untuk membuat koneksi baru. Koneksi dibuat menggunakan *https* yang merupakan protokol *http* yang diamankan menggunakan SSL.

2. Client.js

File client.js berada di dalam *data center*. *Source code* tambahan diberikan untuk implementasi TLS/SSL dapat dilihat pada Gambar 4.6.

```
Client.js
1  https = require('https')
2
3  const options = {
4      key:    fs.readFileSync('/home/pi/qoap-middleware/server-
5  key.pem'),
6      cert:   fs.readFileSync('/home/pi/qoap-middleware/server-
7  cert.pem'),
8      ca:    [fs.readFileSync('/home/pi/qoap-middleware/server-
9  csr.pem')]
10           ,checkServerIdentity: function (host, cert){
11           return undefined;
12   }
13
14 https.globalAgent.options.rejectUnauthorized = true;
15 const client = io.connect('https://10.34.8.6:3000/', {agent:
16 https.globalAgent});
```

Gambar 4.6 *Source code* implementasi TLS/SSL pada *client.js*

Langkah pertama yang dilakukan adalah memanggil *library https* menggunakan kode `https = require('https')`. Hal ini memungkinkan *script client.js* untuk membuat atau menyambungkan koneksi menggunakan protokol https. Variabel *options* digunakan untuk memanggil kunci serta sertifikat yang dibutuhkan untuk membuat koneksi menggunakan TLS/SSL. Untuk kunci menggunakan file *server-key.pem*,

untuk sertifikat menggunakan file *server-cert.pem*, file *server-csr.pem* digunakan untuk meminta dan memvalidasi sertifikat. Sedangkan variabel *client* digunakan untuk menyambungkan koneksi dengan *middleware*. *GlobalAgent* merupakan sebuah fungsi yang digunakan sebagai alat bantu untuk menyambungkan koneksi menggunakan *https*.

4.6.2 Implementasi AES-256

Implementasi AES-256 mengubah source code *middleware* pada *file websocket_api.js* yang terletak pada folder *controller* dan *data center* pada *file client.js*. Sebelum melakukan implementasi, sistem harus sudah terinstall *Node.js*. Implementasi AES-256 dilakukan menggunakan *library* yang ada di dalam Node.js, yaitu *crypto.* Menggunakan *crypto,* penulis dapat menggunakan berbagai macam algoritma enkripsi, salah satunya adalah AES-256. Adapun implementasi yang dilakukan adalah:

1. Websocket_api.js

   File websocket_api.js berada di dalam *middleware* pada folder *controller* dan digunakan untuk mengatur pengiriman data dari *middleware* ke data center. *Source code* tambahan diberikan untuk implementasi TLS/SSL dapat dilihat pada Gambar 4.7.

Websocket_api.js

```
1  const crypto = require('crypto');
2
3  function encrypt(key, data) {
4              var cipher = crypto.createCipher('aes256', key);
5              var crypted = cipher.update(data, 'utf-8', 'hex');
6              crypted += cipher.final('hex');
7          return crypted;
8          }
9
10 let  key  =  new  Buffer('85CE6CCF67FBBAA8BB13479C3A6E084D',
11 'hex');
12
13 let encryptedText = encrypt(key, stringValue);
```

Gambar 4.7 *Source code* implementasi AES-256 pada *websocket_api.js*

   Fungsi *encrypt* berfungsi untuk mengenkripsi data yang dimasukkan ke dalam fungsi itu sendiri. Variabel *key* digunakan sebagai kunci rahasia untuk mengenkripsi dan mendekripsi data. Kemudian data yang masuk dari sensor (*stringValue*) dimasukkan ke fungsi *encrypt* yang pada akhirnya menghasilkan variabel *encryptedText* yang berupa hasil enkripsi data dari sensor. Pada akhirnya data yang dikirim adalah *encryptedText* dan bukan *stringValue*.

2. Client.js

   File client.js berada di dalam *data center*. *Source code* tambahan diberikan untuk implementasi AES-256 dapat dilihat pada Gambar 4.8.

Client.js

```
1  var  key  =  new  Buffer('85CE6CCF67FBBAA8BB13479C3A6E084D',
2  'hex');
3  function decrypt(key, data) {
```

```
4       var decipher = crypto.createDecipher('aes256', key);
5       var decrypted = decipher.update(data, 'hex', 'utf-8');
6       decrypted += decipher.final('utf-8');
7       return decrypted;
8   }
9
10  let data = raw
11  var decryptedText = decrypt(key, data.toString('utf-8'))
12  data = new Buffer.from(JSON.stringify(decryptedText))
```

Gambar 4.8 *Source code* implementasi AES-256 pada *client.js*

Variabel *key* digunakan sebagai kunci rahasia untuk mengenkripsi dan mendekripsi data sehingga nilainya sama dengan *key* pada *websocket_api.js*. Fungsi *decrypt* digunakan untuk mendekripsi data. Data yang dihasilkan dari proses dekripsi adalah data *buffer*, sehingga harus diubah menjadi data yang dapat dimasukkan ke dalam *database*, yaitu dengan format *JSON*.

BAB 5 PENGUJIAN DAN PENGAMBILAN DATA

Bab ini berisikan tentang pengujian dan pengambilan data yang dilakukan oleh penulis.

5.1 Pengujian Pengiriman Pesan

Pengujian dilakukan untuk memastikan bahwa sistem *middleware* yang digunakan oleh penulis sudah bekerja dengan benar. Pengujian dilakukan dengan membuat *middleware* mengirimkan pesan ke data center, data center menggunakan fungsi *console log* untuk melihat apakah pesan tersebut berhasil diterima serta untuk melakukan validasi pesan dengan membandingkan pesan yang dikirimkan dari *middleware* dengan pesan yang diterima oleh data center. Jika kedua pesan sama, maka pengujian dianggap berhasil dan data yang diterima adalah valid. Pesan yang dikirim merupakan pesan palsu atau *dummy* yang dibuat oleh penulih. Adapun pesan palsu yang digunakan adalah sebagai berikut:

```
msg = '{"sensor”: {"module":"dht22", "tipe":"esp8266", "index":1751566,
"ip":"192.168.222.222"}, "protocol":"coap", "topic":"home\/kitchen",
"humidity”: {"value":75,"unit":"%"}, "timestamp":"Wed, 15 Nov 2017
07:21:52 GMT","temperature”: {"value":30,"unit":"celcius"}}'
```

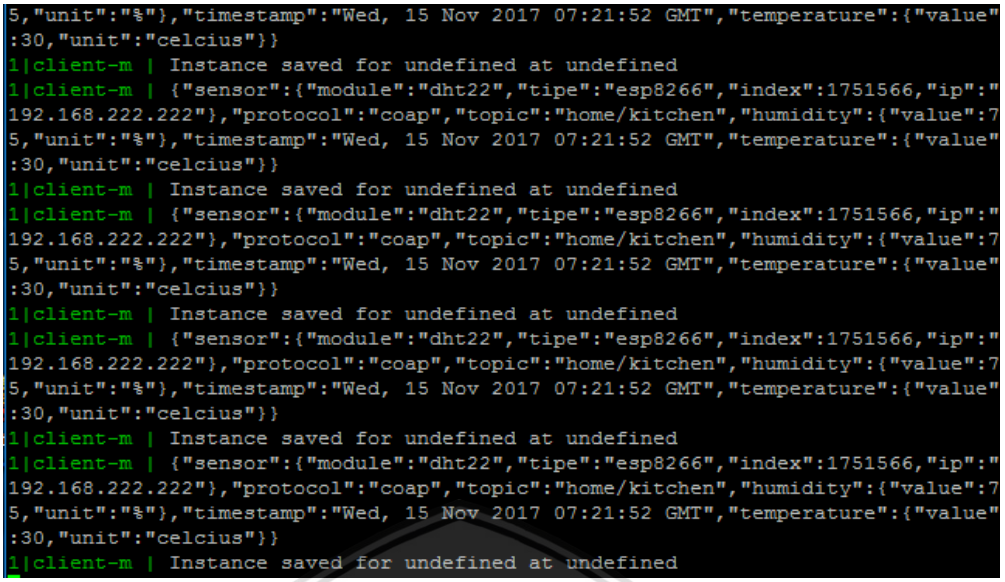
Gambar 5.1 Pesan palsu

Pada Gambar 5.1 di atas, dapat dilihat bahwa isi dari pesan terdapat jenis modul, tipe sensor, index, *IP*, dll. Yang perlu diperhatikan adalah nilai dari variabel *IP*, pada variabel ini penulis memberikan nilai 192.168.222.222. Untuk melakukan validasi pesan, dapat dilakukan dengan melihat apakah variabel *IP* pada pesan yang diterima data center bernilai 192.168.222.222, jika iya maka pesan dianggap valid, jika tidak maka dianggap tidak valid.

Dalam pengujian ini telah dirancang tiga skenario untuk melihat apakah *middleware* telah berjalan dengan benar. Skenario-skenario tersebut dibagi berdasarkan mekanisme keamanan yang digunakan.

5.1.1 Skenario 1

Skenario 1 pengujian adalah pengiriman pesan tanpa menggunakan mekanisme keamanan. Berikut merupakan hasil pengujian skenario 1:



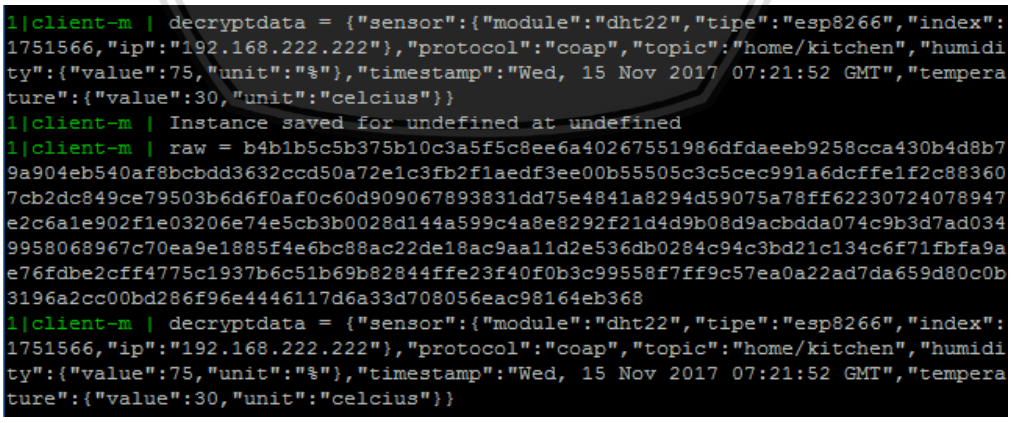
```
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
```

Gambar 5.2 Hasil pengujian skenario 1

Gambar 5.2 di atas merupakan hasil yang ditampilkan oleh fungsi *console log* pada data center, yang merupakan pesan palsu yang dikirmkan oleh *middleware* dan telah diterima oleh data center. Pesan yang diterima perlu dilakukan validasi untuk memastikan apakah pesan tersebut benar-benar pesan yang dikirimkan oleh *middleware*. Validasi pesan dilakukan dengan melihat pada variabel *IP* di dalam pesan. Berdasarkan Gambar 5.2 di atas, dapat dilihat nilai bentuk dari pesan masih berupa *plain text* dan nilai yang dimiliki oleh variabel *IP* adalah 192.168.222.222. Jadi, pesan tersebut dianggap valid.

5.1.2 Skenario 2

Skenario 2 pengujian adalah pengiriman pesan menggunakan mekanisme keamanan AES-256. Berikut merupakan hasil pengujian skenario 2:



```
1|client-m | decryptdata = {"sensor":{"module":"dht22","tipe":"esp8266","index":
1751566,"ip":"192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidi
ty":{"value":75,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","tempera
ture":{"value":30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | raw = b4b1b5c5b375b10c3a5f5c8ee6a40267551986dfdaeeb9258cca430b4d8b7
9a904eb540af8bcbdd3632ccd50a72e1c3fb2f1aedf3ee00b55505c3c5cec991a6dcffe1f2c88360
7cb2dc849ce79503b6d6f0af0c60d909067893831dd75e4841a8294d59075a78ff62230724078947
e2c6a1e902f1e03206e74e5cb3b0028d144a599c4a8e8292f21d4d9b08d9acbdda074c9b3d7ad034
9958068967c70ea9e1885f4e6bc88ac22de18ac9aa11d2e536db0284c94c3bd21c134c6f71fbfa9a
e76fdbe2cff4775c1937b6c51b69b82844ffe23f40f0b3c99558f7ff9c57ea0a22ad7da659d80c0b
3196a2cc00bd286f96e4446117d6a33d708056eac98164eb368
1|client-m | decryptdata = {"sensor":{"module":"dht22","tipe":"esp8266","index":
1751566,"ip":"192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidi
ty":{"value":75,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","tempera
ture":{"value":30,"unit":"celcius"}}
```

Gambar 5.3 Hasil pengujian skenario 2

Gambar 5.3 di atas merupakan hasil yang ditampilkan oleh fungsi *console log* pada data center, yang merupakan pesan palsu yang dikirmkan oleh *middleware* dan telah diterima oleh data center. Pesan yang diterima perlu dilakukan validasi untuk memastikan apakah pesan tersebut benar-benar pesan yang dikirimkan

oleh *middleware*. Validasi pesan dilakukan dengan melihat pada variabel *IP* di dalam pesan. Berdasarkan Gambar di atas, bahwa terdapat dua variabel yang ditampilkan, yaitu *raw* dan *decryptdata*. Dikarenakan penggunaan mekanisme keamanan AES-256, maka pesan yang dikirimkan oleh *middleware* dienkripsi terlebih dahulu, kemudian hasil enkripsi tersebut didekripsi oleh data center. Variabel *raw* merupakan pesan asli yang diterima oleh data center, berupa hasil enkripsi dari pesan palsu yang dibuat oleh penulis. Sedangkan variabel *decryptdata* merupakan hasil dekripsi dari variabel *raw*. Untuk membuktikanpesan yang diterima data center adalah valid, perlu dilihat nilai *IP* pada variabel *decryptdata*. Berdasarkan Gambar 5.3 di atas, terlihat bahwa nilai dari *IP* adalah 192.168.222.222, sama dengan nilai *IP* pada pesan palsu. Jadi, pesan tersebut dianggap valid.

5.1.3 Skenario 3

Skenario 3 pengujian adalah pengiriman pesan menggunakan mekanisme keamanan TLS/SSL. Berikut merupakan hasil pengujian skenario 3:

```
0|monitori | Bad biscuits make the baker broke, bro.

[STREAMING] Now streaming realtime logs for [all] processes
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
1|client-m | Instance saved for undefined at undefined
1|client-m | {"sensor":{"module":"dht22","tipe":"esp8266","index":1751566,"ip":"
192.168.222.222"},"protocol":"coap","topic":"home/kitchen","humidity":{"value":7
5,"unit":"%"},"timestamp":"Wed, 15 Nov 2017 07:21:52 GMT","temperature":{"value"
:30,"unit":"celcius"}}
```

Gambar 5.4 Hasil pengujian skenario 3

Gambar 5.4 di atas merupakan hasil yang ditampilkan oleh fungsi *console log* pada data center, yang merupakan pesan palsu yang dikirmkan oleh *middleware* dan telah diterima oleh data center. Pesan yang diterima perlu dilakukan validasi untuk memastikan apakah pesan tersebut benar-benar pesan yang dikirimkan oleh *middleware*. Hasil *console log* pada skenario 3 terlihat sama dengan hasil pengujian skenario 1. Hal ini dikarenakan pada mekanisme TLS/SSL, dilakukan proses dekripsi pesan terlebih dahulu sebelum memasukkan pesan ke data center. Berbeda dengan mekanisme keamanan AES-256, pesan dimasukkan ke data center terlebih dahulu sebelum didekripsi, sehingga pada skenario 2, *raw data* yang diterima adalah berupa pesan enkripsi. Validasi pesan dilakukan dengan melihat pada variabel *IP* di dalam pesan. Berdasarkan Gambar 5.4 di atas, dapat

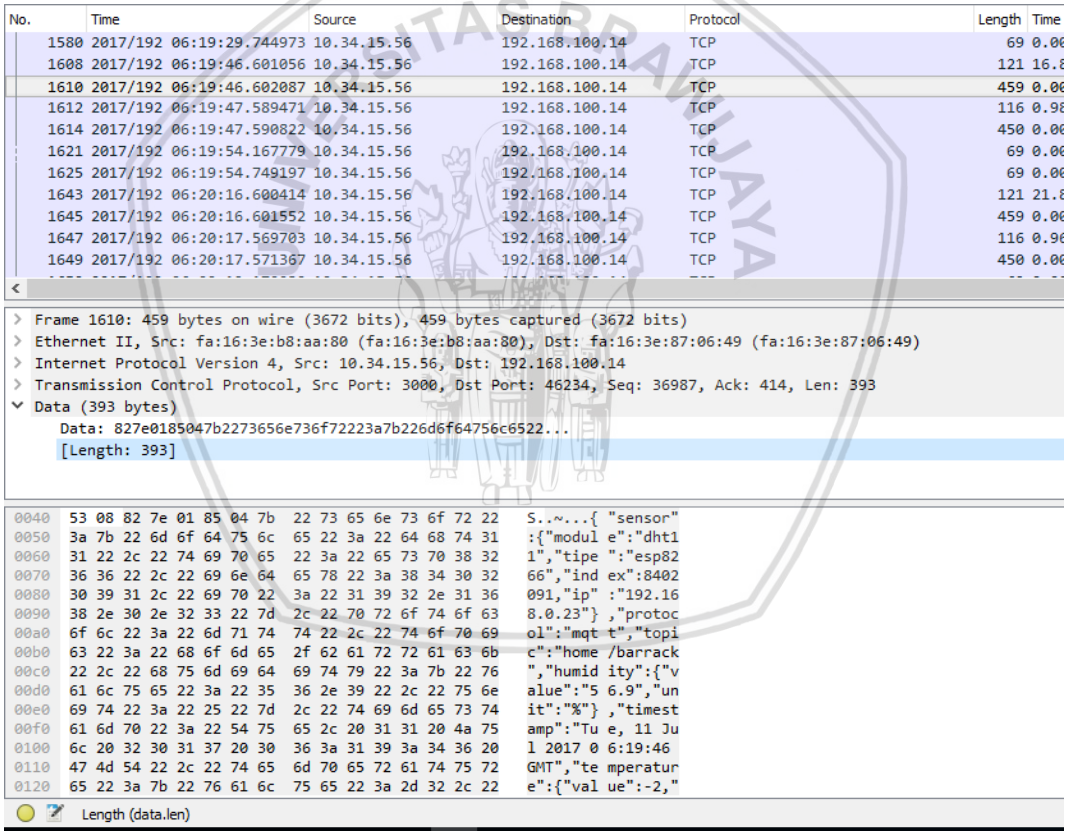
dilihat nilai bentuk dari pesan masih berupa *plain text* dan nilai yang dimiliki oleh variabel *IP* adalah 192.168.222.222. Jadi, pesan tersebut dianggap valid.

5.2 Pengambilan Data

Pada bagian pengambilan data ini terdapat tiga skenario, yaitu pengambilan data tanpa metode keamanan, pengambilan data menggunakan AES-256, dan pengambilan data menggunakan TLS/SSL. Setiap skenario akan dilakukan pengambilan data sebanyak lima kali. Berikut merupakan hasil pengambilan data pada penelitian ini.

5.2.1 Skenario 1

Skenario 1 pada pengambilan data adalah pengiriman data tanpa menggunakan mekanisme keamanan. Selanjutnya, program *tcpdump* akan menghasilkan *file* berformat *pcap. File* tersebut dibuka di *wireshark* untuk diolah datanya. Berikut merupakan hasil dari pengambilan data skenario 1:

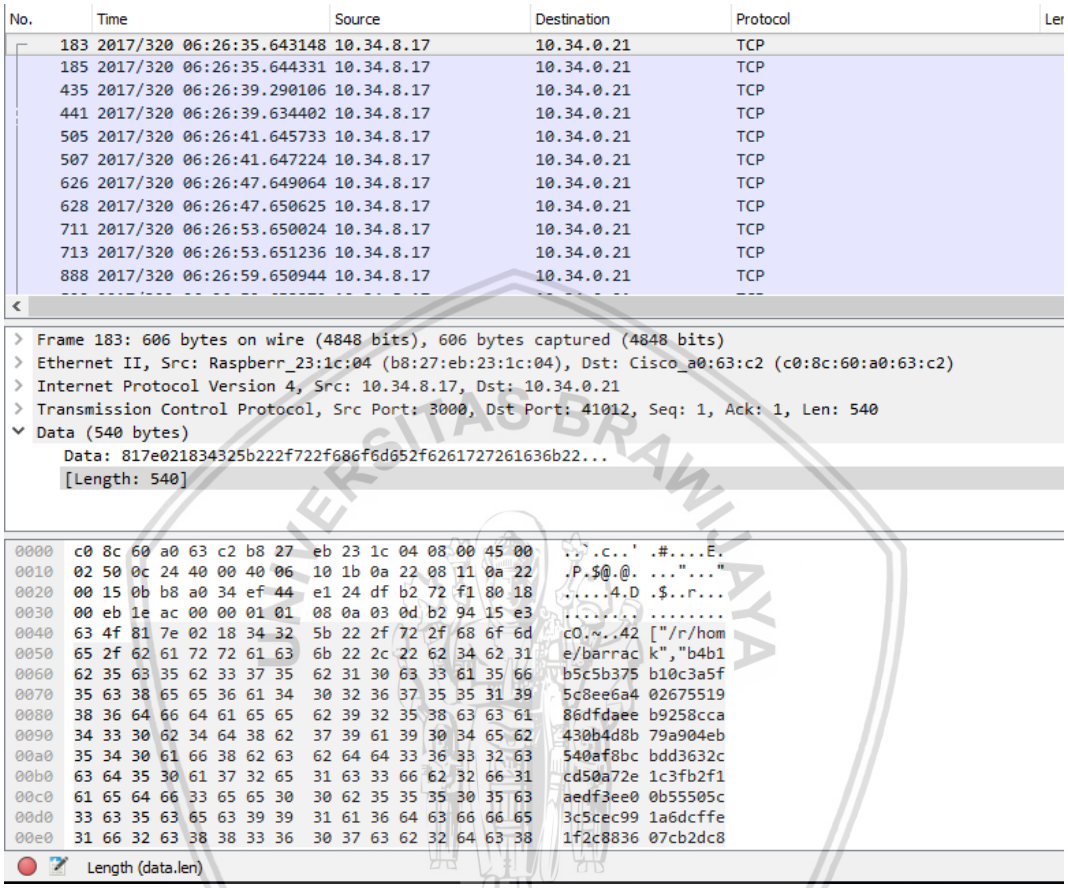


Gambar 5.5 Hasil pengambilan data skenario 1 dalam *wireshark*

Pada Gambar 5.5 di atas menunjukkan isi paket dari salah satu *file* hasil *capture packet,* yang sudah dibuka menggunakan aplikasi *wireshark*. Berdasarkan gambar di atas, dapat dilihat pesan memiliki paket *Frame, Ethernet II,* dll. Pada bagian bawah gambar, terlihat isi data dari salah satu paket.

5.2.2 Skenario 2

Skenario 2 pada pengambilan data adalah pengiriman data menggunakan mekanisme keamanan AES-256. Selanjutnya, program *tcpdump* akan menghasilkan *file* berformat *pcap. File* tersebut dibuka di *wireshark* untuk diolah datanya. Berikut merupakan hasil dari pengambilan data skenario 2:

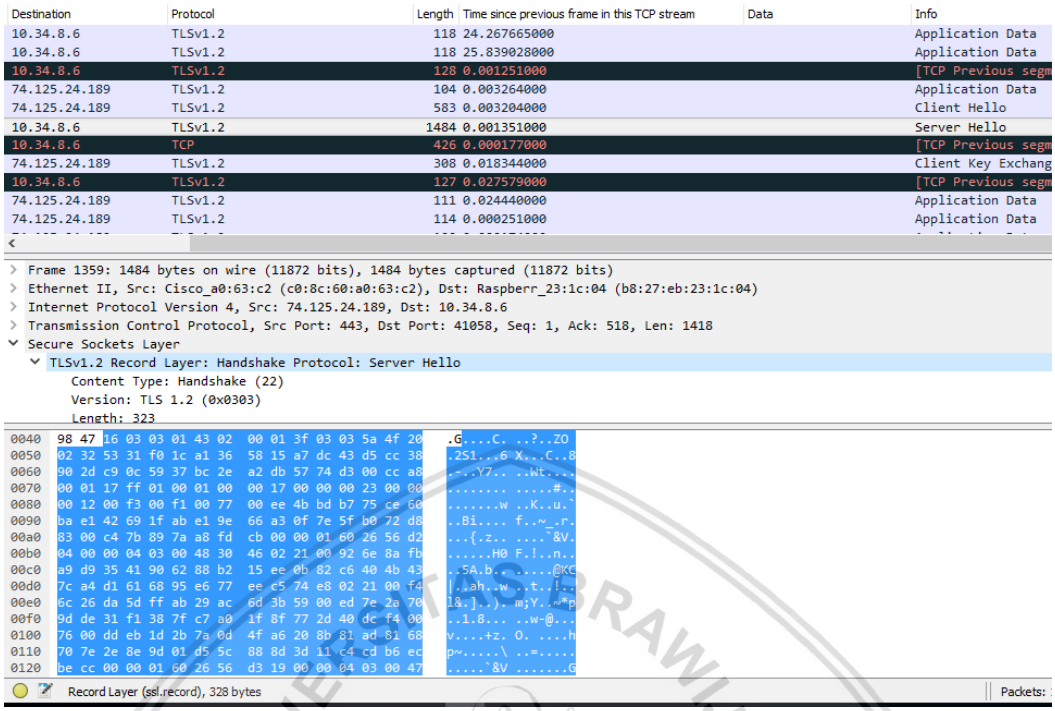


Gambar 5.6 Hasil pengambilan data skenario 2 dalam *wireshark*

Pada Gambar 5.6 di atas menunjukkan isi paket dari salah satu *file* hasil *capture packet,* yang sudah dibuka menggunakan aplikasi *wireshark*. Berdasarkan gambar di atas, dapat dilihat pesan memiliki paket *Frame, Ethernet II,* dll. Pada bagian bawah gambar terlihat isi data dari salah satu paket. Jika diperhatikan, bentuk dari isi data sudah berbeda dengan isi data dari skenario 1.

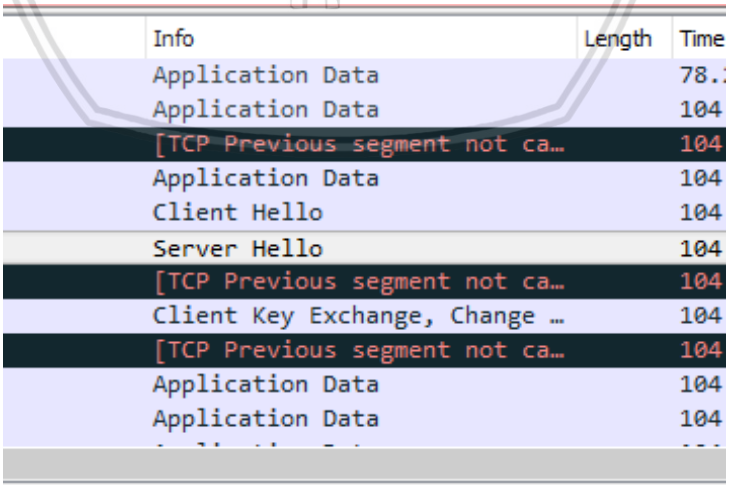
5.2.3 Skenario 3

Skenario 3 pada pengambilan data adalah pengiriman data menggunakan mekanisme keamanan TLS/SSL. Selanjutnya, program *tcpdump* akan menghasilkan *file* berformat *pcap. File* tersebut dibuka di *wireshark* untuk diolah datanya. Berikut merupakan hasil dari pengambilan data skenario 3:



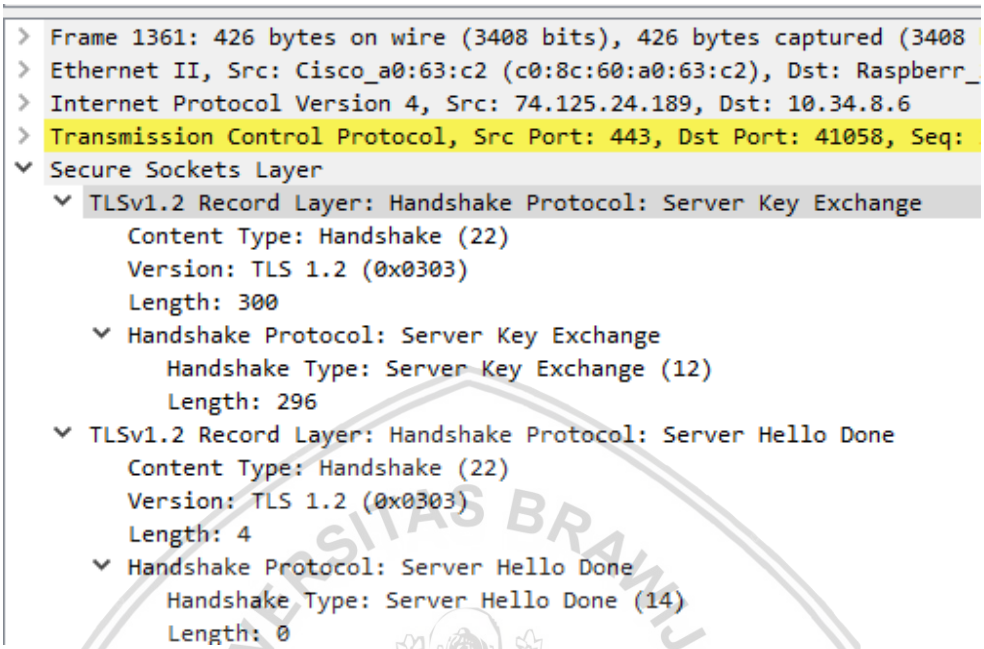
Gambar 5.7 Hasil pengambilan data Skenario 3

Pada Gambar 5.7 di atas menunjukkan isi paket dari salah satu *file* hasil *capture packet,* yang sudah dibuka menggunakan aplikasi *wireshark*. Berdasarkan gambar di atas, dapat dilihat pesan memiliki paket *Frame, Ethernet II,* dll. Pada bagian bawah gambar terlihat isi data dari salah satu paket. Jika diperhatikan, isi dari *file* hasil *capture packet* skenario 3 memiliki banyak perbedaan dengan skenario lainnya. Dapat dilihat bahwa protokol yang digunakan terlihat berbeda, serta informasi pada setiap paket data yang ada pada pesan.



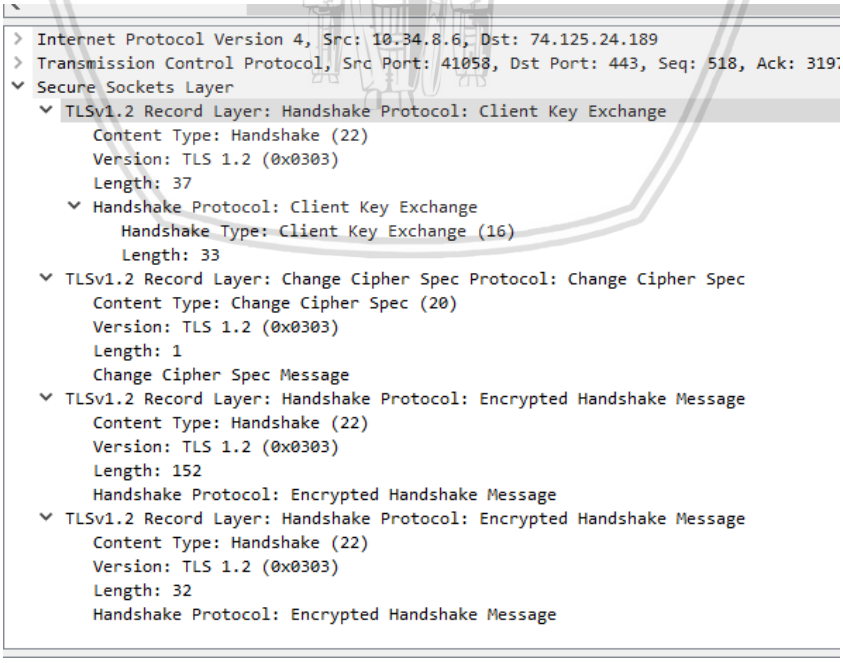
Gambar 5.8 Hasil pengambilan data skenario 3 dalam *wireshark*

Pada Gambar 5.8 di atas menunjukkan salah satu informasi yang terletak pada *file* hasil *capture packet* skenario 3. Informasi tersebut menunjukkan jenis pesan dan apa yang dilakukan oleh pesan tersebut.



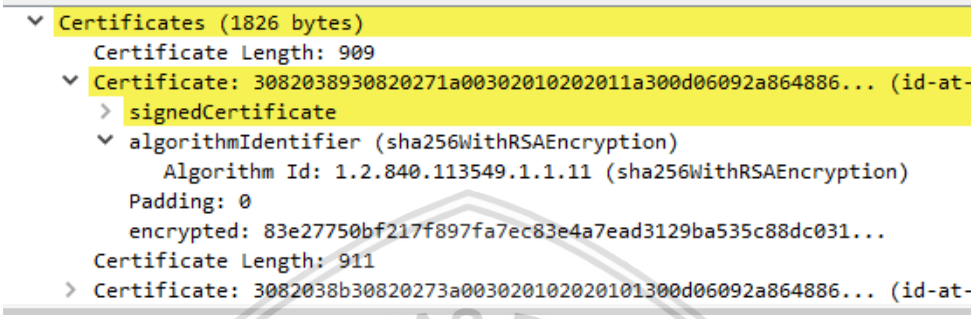
Gambar 5.9 Hasil pengambilan data skenario 3 dalam *wireshark*

Pada Gambar 5.9 di atas menunjukkan salah satu informasi yang terletak pada satu paket data. Informasi tersebut menunjukkan protokol apa yang digunakan oleh paket data dan pesan apa yang dibawa oleh paket data tersebut. Paket data tersebut membawa pesan untuk proses handshake dari data center.



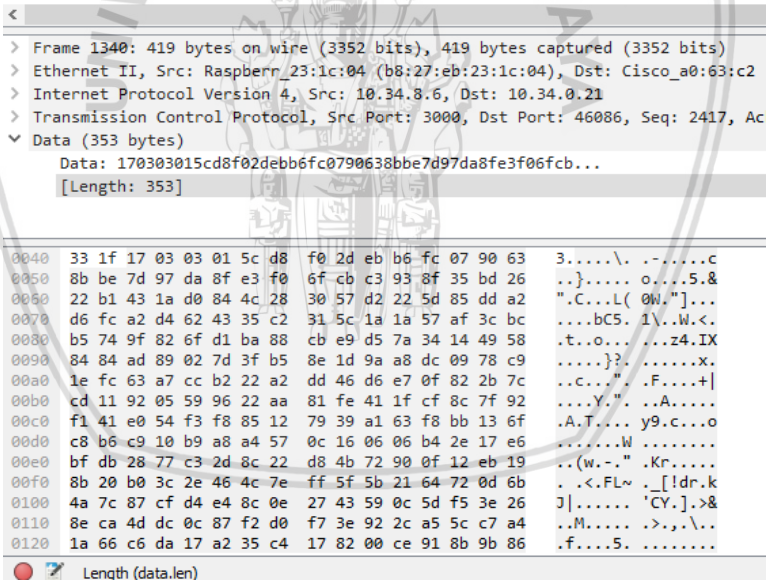
Gambar 5.10 Hasil pengambilan data skenario 3 dalam *wireshark*

Pada Gambar 5.10 di atas menunjukkan salah satu informasi protokol apa yang digunakan oleh paket data dan pesan apa yang dibawa oleh paket data tersebut. Paket data tersebut membawa pesan untuk proses *handshake* dari *middleware*. Dalam paket data tersebut, terdapat pesan *change cipher spec protocol* yang merupakan proses penentuan algoritma enkripsi yang akan digunakan pada pengiriman data selanjutnya setelah *handshake* berhasil.



Gambar 5.11 Algoritma enkripsi TLS/SSL

Pada Gambar 5.11 di atas menunjukkan algoritma apa yang digunakan untuk proses enkripsi pada TLS/SSL yang digunakan, yaitu SHA-256 dengan enkripsi RSA.

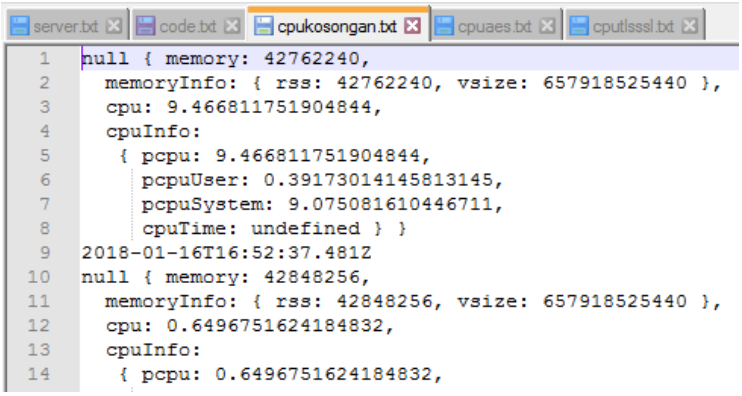


Gambar 5.12 Hasil pengambilan data skenario 3 dalam *wireshark*

Pada Gambar 5.12 di atas menunjukkan isi dari paket data yang dikirimkan. Paket data tersebut adalah data aplikasi yang dikirimkan dari *middleware* yang sudah dienkripsi menggunakan mekanisme keamanan TLS/SSL.

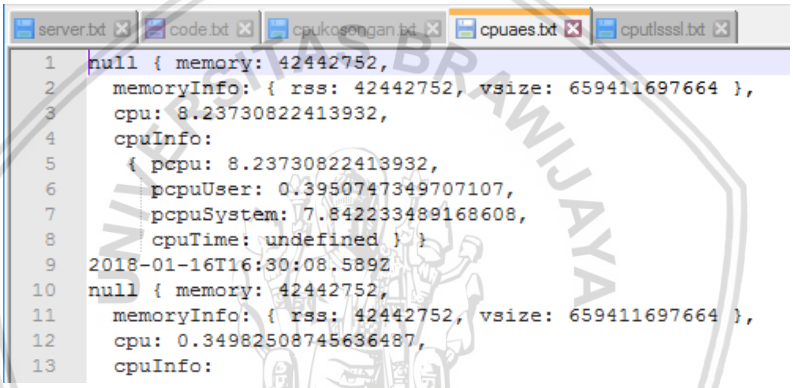
5.2.4 Penggunaan CPU dan Memory

Pengambilan data penggunaan CPU dan memory dilakukan 5 kali untuk setiap skenario. Berikut merupakan hasil pengambilan data tersebut:



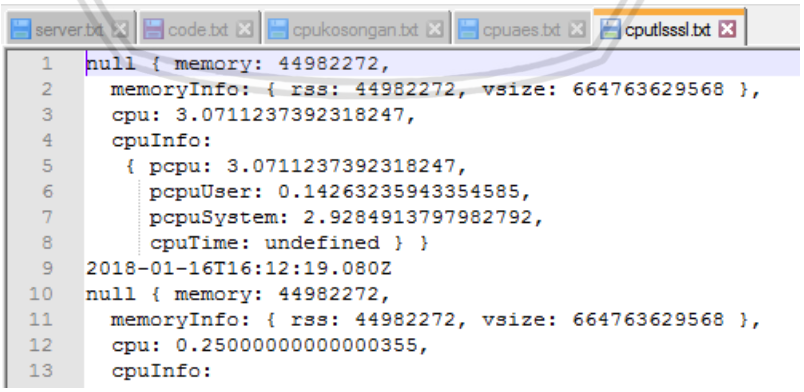
Gambar 5.13 Pengambilan data penggunaan CPU dan Memory 1

Gambar 5.13 di atas menunjukkan hasil pengambilan data penggunaan CPU dan *memory* pada skenario 1, yaitu pengiriman pesan tanpa mekanisme keamanan. Hasil pengambilan data diberi nama *cpukosongan.txt*.



Gambar 5.14 Pengambilan data penggunaan CPU dan Memory 2

Gambar 5.14 di atas menunjukkan hasil pengambilan data penggunaan CPU dan *memory* pada skenario 2, yaitu pengiriman pesan menggunakan mekanisme keamanan AES-256. Hasil pengambilan data diberi nama *cpuaes.txt*.



Gambar 5.15 Pengambilan data penggunaan CPU dan Memory 3

Gambar 5.15 di atas menunjukkan hasil pengambilan data penggunaan CPU dan *memory* pada skenario 3, yaitu pengiriman pesan menggunakan mekanisme keamanan TLS/SSL. Hasil pengambilan data diberi nama *cputlsssl.txt*.

BAB 6 PENGOLAHAN DATA DAN ANALISIS

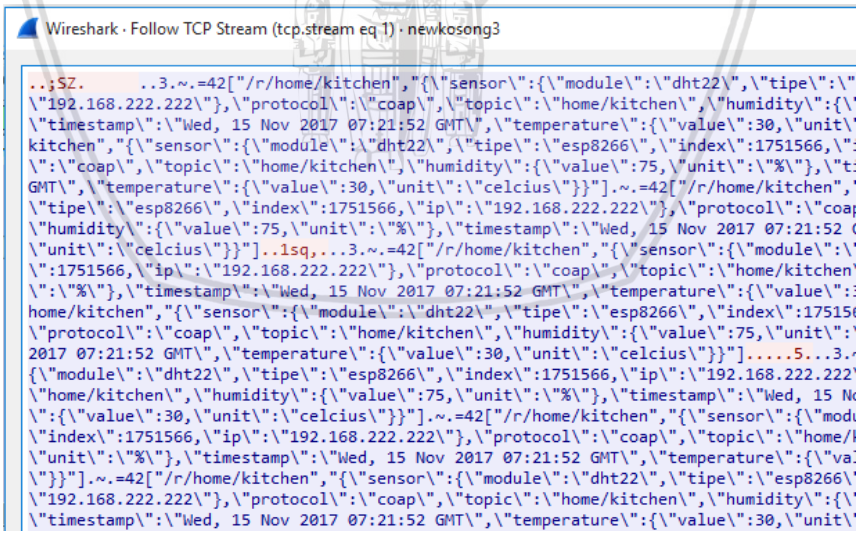
Pada bagian ini, pernulis melakukan pengolahan data dengan melakukan *filter* pada *wireshark* untuk mendapatkan data yang dibutuhkan untuk analisis. Hasil *filter* yang dibuka pada wireshark akan disimpan dalam format dokumen Microsoft excel, yang pada akhirnya digunakan untuk menghitung parameter uji. Parameter uji yang digunakan adalah *packet loss, delay, jitter,* dan penggunaan cpu dan memory. Penulis juga melakukan analisis hasil berdasarkan pengolahan data.

6.1 Pengolahan Data

Pengolahan data dilakukan untuk mengetahui nilai dari parameter uji yang telah ditentukan pada awal penelitian. Langkah pertama untuk melakukan pengolahan data adalah dengan melakukan *filter* pada *wireshark* untuk mengambil nilai yang dibutuhkan untuk mendapatkan nilai parameter uji. Kemudian, data tersebut dikonversi atau diubah menjadi *file* berformat *csv* yang merupakan format untuk *file* yang digunakan pada *Microsoft excel.* Setiap parameter uji memiliki cara penghitungan yang berbeda. Berikut hasil pengolahan data berdasarkan parameter uji dan skenario.

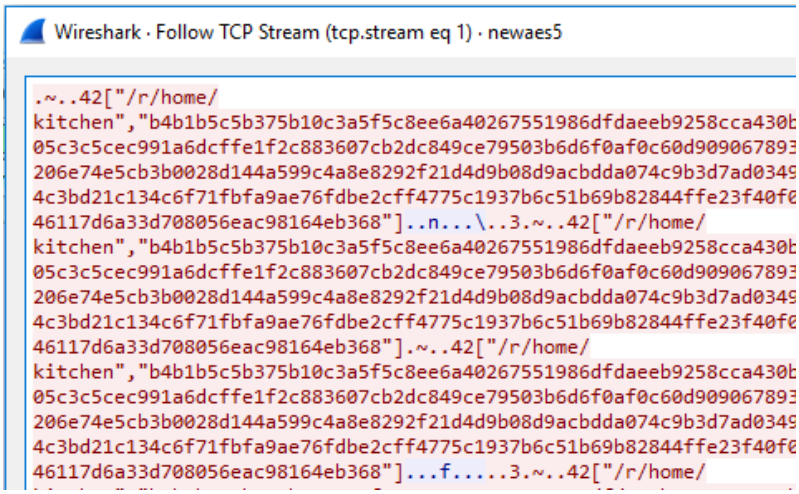
6.1.1 Kinerja Mekasnime Keamanan

Pengolahan data kinerja mekanisme keamanan dilakukan menggunakan fungsi *follow tcp stream* pada *wireshark*. Hasil pengolahan data kinerja mekanisme keamanan adalah sebagi berikut:



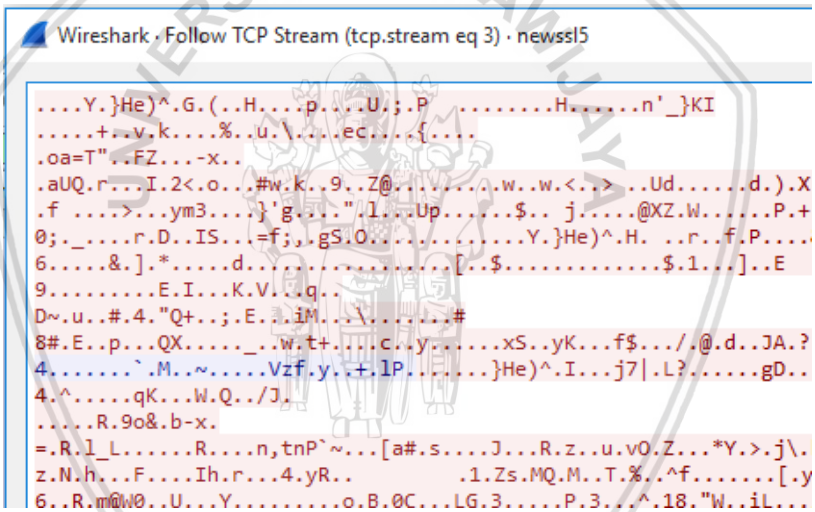
Gambar 6.1 Hasil TCP Stream tanpa mekanisme keamanan

Gambar 6.1 di atas menunjukkan hasil *tcp stream* dari pengiriman pesan tanpa mekanisme keamanan. Dapat dilihat bahwa isi dari data yang dikirimkan dapat dibaca dengan jelas.



Gambar 6.2 Hasil TCP Stream menggunakan AES-256

Gambar 6.2 di atas menunjukkan hasil *tcp stream* dari pengiriman pesan menggunakan AES-256. Dapat dilihat bahwa isi dari data yang dikirimkan tidak dapat dibaca dengan jelas.



Gambar 6.3 Hasil TCP Stream menggunakan TLS/SSL

Gambar 6.3 di atas menunjukkan hasil *tcp stream* dari pengiriman pesan menggunakan TLS/SSL. Dapat dilihat bahwa isi dari data yang dikirimkan tidak dapat dibaca dengan jelas.

6.1.2 Penggunaan CPU dan Memory

Hasil pengolahan data penggunaan CPU dan *memory* adalah sebagai berikut:

Tabel 6.1 Rata-rata Penggunaan Memory

Pengiriman ke-	Memory Usage (%)		
	Tanpa mekanisme	AES	TLS/SSL
1	4,465673913	4,425025145	4,652384633

2	2,401813869	4,317123261	4,358979994
3	2,419773261	4,398566334	4,438283659
4	2,469037879	4,489440271	4,443959641
5	2,489282217	4,510805252	4,516568557

Tabel 6.1 di atas menunjukkan penggunaan memory berdasarkan mekanisme keamanan. Terlihat bahwa jika dibandingkan penggunaan *memory* tanpa mekanisme keamanan memiliki nilai lebih sedikit dibandingkan kedua mekanisme keamanan. Setiap mekanisme keamanan menghasilkan kenaikan penggunaan *memory* paling tidak 2% dibandingkan tanpa mekanisme keamanan. Sedangkan untuk penggunaan CPU adalah sebagai berikut:

Tabel 6.2 Rata-rata Penggunaan CPU

Pengiriman ke-	CPU Usage (%)		
	Tanpa mekanisme	AES	TLS/SSL
1	0,667542063	0,592358369	0,445243373
2	0,190433069	0,612176684	0,725541418
3	0,160652898	0,34152133	0,43443128
4	0,150503636	0,237024268	0,359218086
5	0,147518592	0,237017806	0,367569326

Tabel 6.2 di atas menunjukkan penggunaan CPU berdasarkan mekanisme keamanan. Terlihat bahwa secara umum, nilai penggunaan CPU tanpa mekanisme keamanan lebih kecil jika dibandingkan dengan kedua mekanisme keamanan. Akan tetapi, terdapat satu titik dimana penggunaan CPU tanpa mekanisme keamanan lebih besar, hal ini dikarenakan beberapa faktor, seperti waktu pencatatan penggunaan CPU, aktifitas yang dilakukan pada titik tersebut, serta kondisi dari CPU itu sendiri. Hal tersebut dapat mengakibatkan kenaikan penggunaan CPU yang cukup besar. Berikut merupakan tabel rata-rata dari penggunaan CPU dan memory berdasarkan mekanisme keamanan:

Tabel 6.3 Rata-rata Total Penggunaan CPU dan Memory

	Mekanisme Keamanan		
	Tanpa mekanisme	AES-256	TLS/SSL
CPU (%)	0,263330052	0,404019691	0,466400697
Memory (%)	2,849116228	4,428192053	4,482035297

Tabel 6.3 di atas menunjukkan penggunaan CPU dan memory berdasarkan mekanisme keamanan. Terlihat bahwa rata-rata penggunaan CPU dan memory tanpa mekanisme keamanan lebih kecil dibandingkan dengan kedua mekanisme keamanan. Penggunaan CPU dan *memory* pada kedua mekanisme keamanan memiliki nilai hamper dua kali lipat dibandingkan dengan tanpa mekanisme

keamanan. Akan tetapi, jika kedua mekanisme keamanan tersebut dibandingkan, perbedaan rata-rata penggunaan CPU dan memory-nya tidak berbeda jauh.

6.1.3 Packet Loss

Packet loss adalah rasio antara paket yang hilang dengan total paket yang dikirimkan pada satuan waktu. Proses pengiriman data oleh *middleware* dilakukan secara terjadwal yaitu setiap 10 detik sekali sehingga sepuluh menit seharusnya terdapat 60 data yang seharusnya terkirim oleh *middleware*. Berikut hasil pengolahan data *packet loss* berdasarkan skenario. Nilai *jitter* didapatkan menggunakan rumus (3.1)

1. Skenario 1

Skenario 1 adalah pengiriman data tanpa menggunakan mekanisme keamanan. Pengiriman data dilakukan sebanyak lima kali. Berikut hasil pengolahan data *packet loss* skenario 1.

Tabel 6.4 Packet loss skenario 1

Pengambilan data ke-	Jumlah paket dikirim	Jumlah paket diterima	Packet loss (%)
1	60	60	0
2	60	59	1,666666667
3	60	59	1,666666667
4	60	59	1,666666667
5	60	59	1,666666667
Rata-Rata (%)			**1,333333333**

Tabel 6.4 di atas di atas menampilkan nilai *packet loss* yang didapatkan pada pengolahan data skenario 1. Rata-rata *packet loss* pada skenario 1 adalah 1.333333333%. Satu paket data yang gagal diterima tersebut kemungkinan terjadi dikarenakan proses *capture packet* dijalankan pada waktu yang tidak bersamaan dengan interval pengiriman data, jadi program gagal untuk menangkap pengiriman data pertama atau terakhir.

2. Skenario 2

Skenario 2 adalah pengiriman data menggunakan mekanisme keamanan AES-256. Pengiriman data dilakukan sebanyak lima kali. Berikut hasil pengolahan data *packet loss* skenario 2.

Tabel 6.5 Packet loss skenario 2

Pengambilan data ke-	Jumlah paket dikirim	Jumlah paket diterima	Packet loss (%)
1	60	60	0
2	60	59	1,666666667
3	60	59	1,666666667
4	60	60	0
5	60	59	1,666666667
Rata-rata (%)			**1**

Tabel 6.5 di atas di atas menampilkan nilai *packet loss* yang didapatkan pada pengolahan data skenario 2. Rata-rata *packet loss* pada skenario 2 adalah 1 %, lebih kecil jika dibandingkan dengan rata-rata *packet loss* skenario 1. Satu paket data yang gagal diterima tersebut kemungkinan terjadi dikarenakan proses *capture packet* dijalankan pada waktu yang tidak bersamaan dengan interval pengiriman data, jadi program gagal untuk menangkap pengiriman data pertama atau terakhir.

3. Skenario 3

Skenario 3 adalah pengiriman data menggunakan mekanisme keamanan TLS/SSL. Pengiriman data dilakukan sebanyak lima kali. Berikut hasil pengolahan data *packet loss* skenario 2.

Tabel 6.6 Packet loss skenario 3

Pengambilan data ke-	Jumlah paket dikirim	Jumlah paket diterima	Packet loss (%)
1	60	60	0
2	60	59	1,666666667
3	60	59	1,666666667
4	60	59	1,666666667
5	60	59	1,666666667
Rata-rata (%)			**1,333333333**

Tabel 6.6 di atas di atas menampilkan nilai *packet loss* yang didapatkan pada pengolahan data skenario 3. Rata-rata *packet loss* pada skenario 3 adalah 1.333333333%, lebih besar jika dibandingkan dengan rata-rata *packet loss* skenario 2 dan sama dengan skenario 1. Satu paket data yang gagal diterima tersebut kemungkinan terjadi dikarenakan proses *capture packet* dijalankan pada

waktu yang tidak bersamaan dengan interval pengiriman data, jadi program gagal untuk menangkap pengiriman data pertama atau terakhir.

6.1.4 Delay

Pengolahan data untuk *delay* dilakukan dengan mengambil nilai *delta time* dari hasil *capture packet* yang telah dibuka di *wireshark*. Berikut merupakan hasil pengolahan data *delay* berdasarkan skenario.

1. Skenario 1

Skenario 1 adalah pengiriman data tanpa menggunakan mekanisme keamanan. Data yang telah dibuka di dalam aplikasi *wireshark* disimpan dalam format dokumen *Microsoft Excel.* Menggunakan *Microsoft Excel,* penulis dapat mencari nilai *delay* dan rata-rata *delay*. Berikut adalah dari pengolahan data skenario 1 serta hasil penghitungan rata-rata *delay*-nya:

Tabel 6.7 Delay skenario 1

No. / Pengiriman data ke-	Delay (detik)				
	1	2	3	4	5
1	0,086862	0,09028	0,082935	0,066304	0,083093
2	0,298569	0,798449	0,048616	1,12017	0,005745
3	0,000673	0,000788	0,001672	0,00056	0,000728
4	0,316624	0,575243	0,000627	0,549018	1,538901
5	0,001838	0,001718	0,160618	0,001647	0,000821
6	0,000551	0,000614	0,001798	0,000571	0,111516
7	0,189664	0,189237	0,000707	0,147141	0,001702
8	0,001666	0,001539	0,229261	0,001397	0,000695
9	0,086862	0,09028	0,082935	0,066304	0,083093
10	0,298569	0,798449	0,048616	1,12017	0,005745

Tabel 6.7 di atas menampilkan 10 nilai pertama *delay* skenario 1. Berikut merupakan nilai rata-rata *delay* skenario satu. Terlihat bahwa nilai *delay*-nya bermacam-macam, nilai terkecil dari *delay* di atas adalah 0,000551 detik, sedangkan nilai terbesarnya adalah 1,538901 detik.

Tabel 6.8 Rata-rata delay skenario 1

Pengiriman data ke-	Rata-rata delay (detik)
1	0,153242513

2	0,136893707
3	0,148512657
4	0,164616042
5	0,180352948
Rata-rata delay keseluruhan (detik)	**0,156723573**

Tabel 6.8 di atas menampilkan rata-rata *delay* pada setiap pengiriman data skenario 1. Terlihat bahwa nilai rata-rata *delay* yang dihasilkan tidak teratur, hal ini dikarenakan beberapa factor, seperti waktu pengamatan, trafik jaringan dan lain – lain. Rata-rata *delay* yang dihasilkan pada skenario 1 adalah 0,156723573 detik.

2. Skenario 2

Skenario 2 adalah pengiriman data menggunakan mekanisme keamanan AES-256. Data yang telah dibuka di dalam aplikasi *wireshark* disimpan dalam format dokumen *Microsoft Excel.* Menggunakan *Microsoft Excel,* penulis dapat mencari nilai *delay* dan rata-rata *delay.* Berikut adalah dari pengolahan data skenario 2 serta hasil penghitungan rata-rata *delay*-nya:

Tabel 6.9 Delay skenario 2

No. / Pengiriman data ke-	Delay (detik)				
	1	2	3	4	5
1	0,097788	0,097154	0,07771	0,609409	1,301281
2	0,563175	1,015236	1,21627	0,000679	0,000688
3	0,000688	0,000777	0,000752	0,840507	0,152926
4	0,114263	0,136831	0,653666	0,001881	0,001677
5	0,000216	0,00159	0,001923	0,000504	0,000692
6	0,002712	0,000524	0,000568	0,445683	0,148893
7	0,000568	0,362615	0,144575	0,001719	0,001462
8	0,35074	0,001471	0,001933	0,000586	0,000461
9	0,000202	0,000543	0,000612	0,578699	0,646078
10	0,00203	0,99879	0,093664	0,000771	0,000735

Tabel 6.9 di atas menampilkan 10 nilai pertama *delay* skenario 2. Berikut merupakan nilai rata-rata *delay* skenario 2. Terlihat bahwa nilai *delay*-nya

bermacam-macam, nilai terkecil dari *delay* di atas adalah 0,000202 detik, sedangkan nilai terbesarnya adalah 1,301281 detik.

Tabel 6.10 Rata-rata delay skenario 2

Pengiriman data ke-	Rata-rata delay (detik)
1	0,156576404
2	0,165701668
3	0,172463989
4	0,186801833
5	0,160295313
Rata-rata delay keseluruhan (detik)	**0,168367841**

Tabel 6.10 di atas menampilkan rata-rata *delay* pada setiap pengiriman data skenario 2. Terlihat bahwa nilai rata-rata *delay* yang dihasilkan tidak teratur, hal ini dikarenakan beberapa factor, seperti waktu pengamatan, trafik jaringan dan lain – lain. Nilai rata-rata *delay* skenario 2 adalah 0,168367841 detik, lebih besar jika dibandingkan dengan skenario 1.

3. Skenario 3

Skenario 3 adalah pengiriman data menggunakan mekanisme keamanan TLS/SSL. Data yang telah dibuka di dalam aplikasi *wireshark* disimpan dalam format dokumen *Microsoft Excel.* Menggunakan *Microsoft Excel,* penulis dapat mencari nilai *delay* dan rata-rata *delay.* Berikut adalah dari pengolahan data skenario 3 serta hasil penghitungan rata-rata *delay*-nya:

Tabel 6.11 Delay skenario 3

No. / Pengiriman data ke-	Delay (detik)				
	1	2	3	4	5
1	0,668621	0,030192	0,220215	0,393127	1,130941
2	0,000308	0,000184	0,000245	0,000232	0,000283
3	0,00048	0,000537	0,000681	0,000454	0,00059
4	0,000219	0,000003	0,000004	0,000185	0,000004
5	0,000105	0,000178	0,00024	0,000073	0,000251
6	0,005519	0,00507	0,005797	0,00422	0,006018
7	0,000625	0,000527	0,000681	0,000478	0,001065
8	0,000187	0,000133	0,000165	0,000132	0,00017
9	0,668621	0,030192	0,220215	0,393127	1,130941
10	0,000308	0,000184	0,000245	0,000232	0,000283

Tabel 6.11 di atas menampilkan 10 nilai pertama *delay* skenario 3. Berikut merupakan nilai rata-rata *delay* skenario 3. Terlihat bahwa nilai *delay*-nya bermacam-macam, nilai terkecil dari *delay* di atas adalah 0,000133 detik, sedangkan nilai terbesarnya adalah 1,130941 detik.

Tabel 6.12 Rata-rata delay skenario 3

Pengiriman data ke-	Rata-rata delay (detik)
1	0,118611342
2	0,080793532
3	0,071402141
4	0,09284886
5	0,081386241
Rata-rata delay keseluruhan (detik)	**0,089008423**

Tabel 6.12 di atas menampilkan rata-rata *delay* pada setiap pengiriman data skenario 3. Terlihat bahwa nilai rata-rata *delay* yang dihasilkan tidak teratur, hal ini dikarenakan beberapa factor, seperti waktu pengamatan, trafik jaringan dan

lain – lain. Nilai rata-rata *delay* skenario 3 lebih besar jika dibandingkan dengan skenario 1, tetapi lebih kecil jika dibandingkan dengan skenario 2.

6.1.5 Jitter

Jitter disebabkan oleh panjang queue dalam satu pengolahan data dan reassemble data di akhir pengiriman. Nilai *jitter* didapatkan menggunakan rumus (3.2) dan rumus (3.3). Berikut hasil pengolahan data *jitter* berdasarkan skenario.

1. Skenario 1

Skenario 1 adalah pengiriman data tanpa menggunakan mekanisme keamanan. Berikut hasil pengolahan data variasi dan *jitter* nilai skenario 1.

Tabel 6.13 Variasi delay skenario 1

No. / Pengiriman data ke-	Variasi delay (detik)				
	1	2	3	4	5
1	-0,156646	-0,629778	0,030091	-0,8616	1,533156
2	-0,108689	0,662224	0,874429	-0,176346	-1,51918
3	0,580389	-0,087453	-0,742358	0,317327	0,832204
4	-0,131518	0,12672	-0,150515	-0,308725	-0,55294
5	0,086349	-0,497928	-0,006124	-0,062886	1,001235
6	-0,423222	-0,200486	0,103173	0,234846	-0,50253
7	-0,076367	-0,083939	-0,125435	0,722745	0,604907
8	0,433456	0,698042	-0,108501	-0,333863	-1,179484
9	-0,022097	-0,605755	0,178615	0,289697	0,078456
10	-0,022963	-0,017051	-0,091247	-0,031168	1,034517

Tabel 6.13 di atas menampilkan 10 nilai pertama variasi *delay* skenario 1. Untuk melihat keseluruhan datanya dapat dilihat pada lampiran. Berikut merupakan nilai *jitter* dan rata-rata *jitter* skenario 1. Untuk melihat keseluruhan hasil pengolahan data nilai *jitter* skenario 1, dapat dibuka pada Lampiran A.2.

Tabel 6.14 Rata-rata jitter skenario 1

Pengiriman data ke-	Jitter
1	0,001769156
2	0,007975226

3	0,002230408
4	0,011811921
5	0,009024724
Rata-rata jitter	**0,006562287**

Tabel 6.14 di atas menampilkan nilai *jitter* dan rata-rata *jitter* pada setiap pengiriman data skenario 1. Sama dengan rata-rata *delay,* rata-rata *jitter* memiliki nilai yang tidak teratur, karena nilai *jitter* sendiri diperoleh berdasarkan nilai *delay*. Berdasarkan Tabel 6.14 di atas, rata-rata *jitter* pengolahan data skenario 1 adalah 0,006562287 detik.

2. Skenario 2

Skenario 2 adalah pengiriman data menggunakan mekanisme keamanan AES-256. Berikut hasil pengolahan data variasi dan jitter nilai skenario 2.

Tabel 6.15 Variasi delay skenario 2

No. / Pengiriman data ke-	Variasi delay (detik)				
	1	2	3	4	5
1	-0,249483	-0,016446	-1,122606	-0,03071	-0,655203
2	0,164184	-0,455873	0,061442	-0,02225	-0,34086
3	-0,132853	-0,475139	0,399978	-0,260358	0,052353
4	-0,08469	0,867137	-0,365528	0,171912	-0,341156
5	0,9857	-0,443197	-0,12375	-0,028117	0,303508
6	0,094879	0,394844	0,034083	-0,41118	1,016531
7	-1,132003	-0,74824	0,037087	0,369412	-0,796319
8	-0,207924	-0,066255	0,906994	-0,3548	-0,178194
9	1,280682	0,749914	-0,021564	0,311302	0,170817
10	-1,122981	-0,759299	-0,938899	-0,023191	-0,047603

Tabel 6.15 di atas menampilkan 10 nilai pertama variasi *delay* skenario 2. Untuk melihat keseluruhan datanya dapat dilihat pada lampiran. Berikut merupakan nilai *jitter* dan rata-rata *jitter* skenario 2. Untuk melihat keseluruhan hasil pengolahan data nilai *jitter* skenario 2, dapat dibuka pada Lampiran A.3.

Tabel 6.16 Rata-rata jitter skenario 2

Pengiriman data ke-	Jitter

1	0,003256756
2	0,009690845
3	0,014197192
4	0,00209117
5	0,015408818
Rata-rata jitter	**0,008928956**

Tabel 6.16 di atas menampilkan nilai *jitter* dan rata-rata *jitter* pada setiap pengiriman data skenario 2. Sama dengan rata-rata *delay,* rata-rata *jitter* memiliki nilai yang tidak teratur, karena nilai *jitter* sendiri diperoleh berdasarkan nilai *delay*. Berdasarkan Tabel 6.16 di atas, rata-rata *jitter* pengolahan data skenario 2 adalah 0,008928956 detik.

3. Skenario 3

Skenario 3 adalah pengiriman data tanpa menggunakan mekanisme keamanan TLS/SSL. Berikut hasil pengolahan data variasi dan jitter nilai skenario 3.

Tabel 6.17 Variasi delay skenario 3

No. / Pengiriman data ke-	Variasi delay (detik)				
	1	2	3	4	5
1	-0,190333	0,483027	-0,581385	-0,332407	-0,000461
2	0,140834	0,288152	0,618964	-0,217656	-0,021623
3	-0,233677	-0,026491	-0,643153	0,045928	-0,021999
4	0,040454	-0,597732	-0,004972	0,596731	-0,022316
5	0,134295	0,55821	0,037652	0,903608	-0,022598
6	-0,021177	-0,024398	0,991488	-1,330568	-0,021464
7	-0,158439	-0,204883	-1,06687	0,372304	-0,026988
8	0,115589	0,165618	0,156586	0,87885	-0,103118
9	-0,021647	-0,4866	0,834061	-0,016995	0,068229
10	-0,020679	0,0358	-1,021399	-0,914786	-0,022047

Tabel 6.17 di atas menampilkan 10 nilai pertama variasi *delay* skenario 3. Untuk melihat keseluruhan datanya dapat dilihat pada lampiran. Berikut merupakan nilai *jitter* dan rata-rata *jitter* skenario 3. Untuk melihat keseluruhan hasil pengolahan data nilai *jitter* skenario 3, dapat dibuka pada Lampiran A.4.

Tabel 6.18 Rata-rata delay skenario 3

Pengiriman data ke-	Jitter
1	0,001660939
2	0,004390695
3	0,006967346
4	0,001294216
5	0,001731188
Rata-rata jitter	**0,003208877**

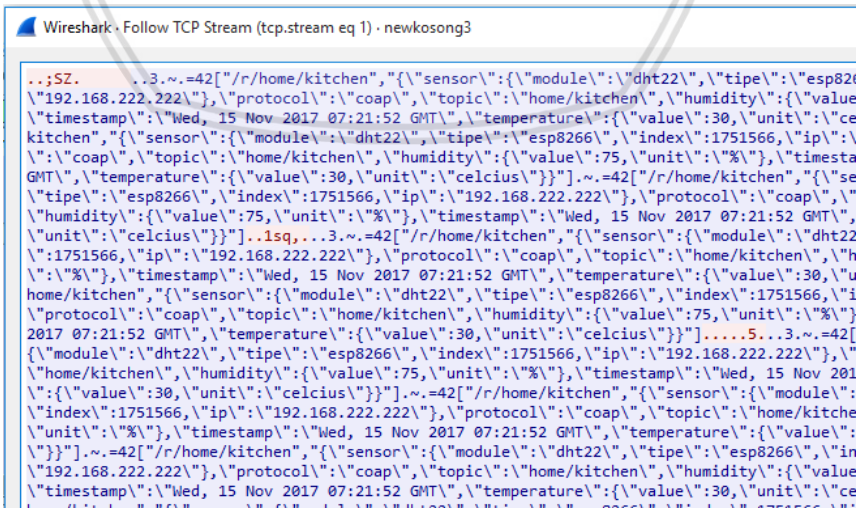
Tabel 6.18 di atas menampilkan nilai *jitter* dan rata-rata *jitter* pada setiap pengiriman data skenario 3. Sama dengan rata-rata *delay,* rata-rata *jitter* memiliki nilai yang tidak teratur, karena nilai *jitter* sendiri diperoleh berdasarkan nilai *delay*. Berdasarkan Tabel 6.18 di atas, rata-rata *jitter* pengolahan data skenario 3 adalah 0,003208877 detik.

6.2 Analisis

6.2.1 Kinerja mekanisme keamanan

1. Skenario 1

Skenario 1 merupakan pengiriman data tanpa menggunakan mekanisme keamanan.



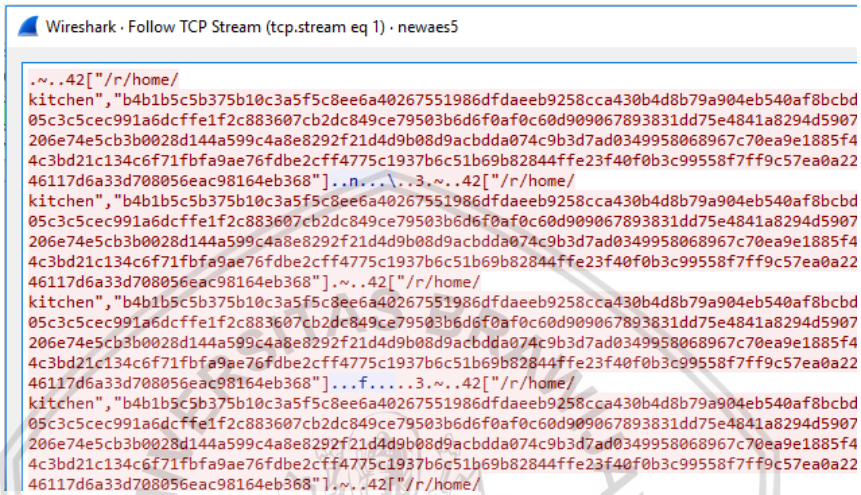
Gambar 6.4 Isi data pesan pada skenario 1

Berdasarkan Gambar 6.4, pada pengambilan data skenario 1, hasil *capture packet* terlihat sangat mudah dibaca dan data tersebut dapat dilihat isinya. Hal ini

menunjukkan bahwa mudah sekali untuk menyadap data yang dikirimkan dari *middleware* ke data center. Kerahasiaan data antara *middleware* dan data center tidak terjamin.

2. *Skenario 2*

Berdasarkan Gambar 6.5 pada pengambilan data skenario 2, hasil *capture packet* terlihat hanya sebagai kumpulan huruf-huruf yang acak. Ini dikarenakan data yang dikirim telah dienkripsi menggunakan algoritma AES-256.

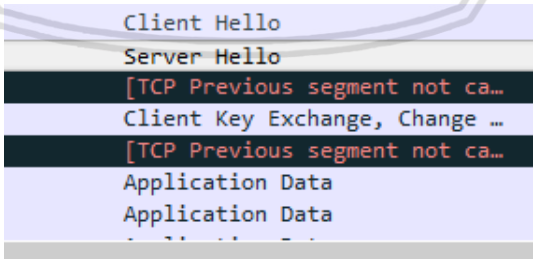


Gambar 6.5 Isi data pesan pada skenario 2

Berdasarkan Gambar 6.5, mekanisme keamanan telah berhasil mengenkripsi data yang dikirimkan, sehingga pihak lain tidak bisa mengetahui isi sebenarnya dari data tersebut. Sehingga data yang dikirimkan bisa dianggap aman.

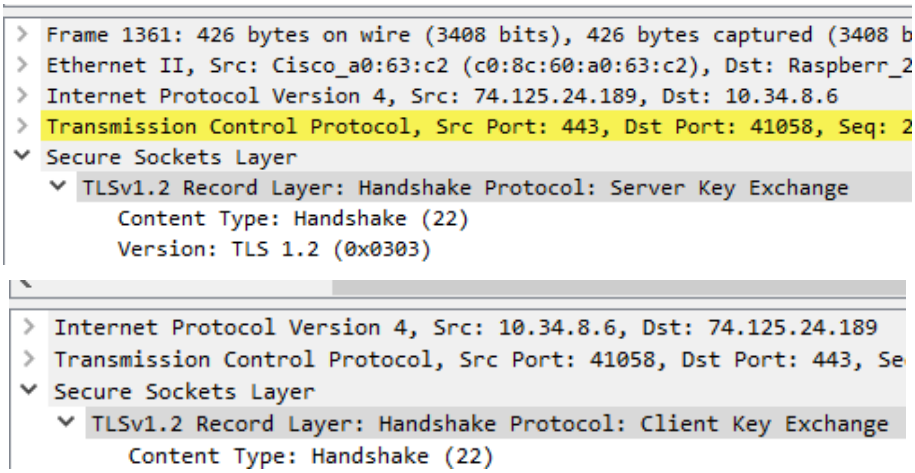
3. *Skenario 3*

Pada pengambilan data skenario 3, terlihat bahwa client mengirimkan packet *client hello* dan pada paket selanjutnya server membalas dengan packet *server hello.* Hal ini merupakan paket awal yang dikirimkan saat melakukan *handshake*.



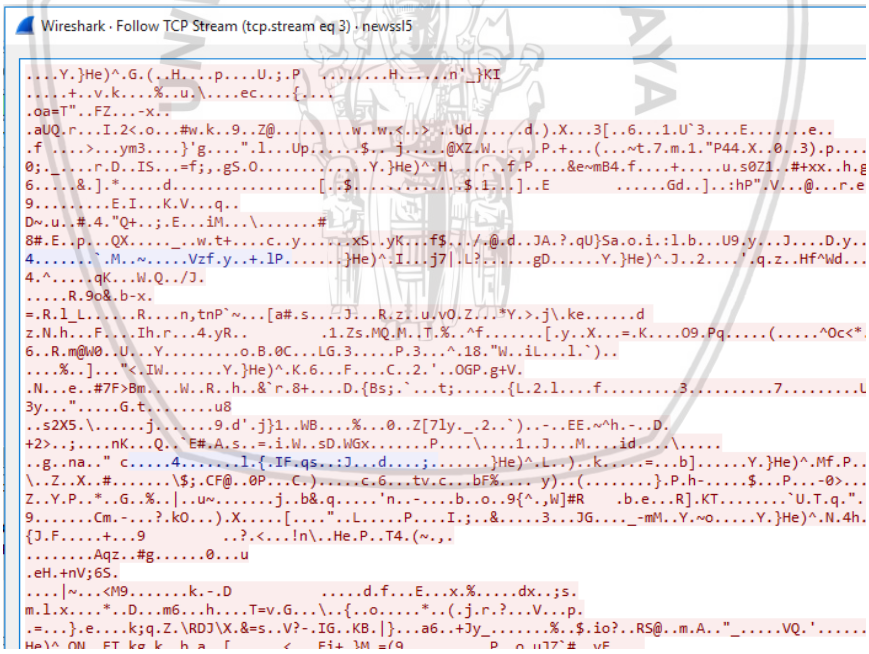
Gambar 6.6 Info proses *handshake* skenario 3

Gambar 6.6 di atas menunjukkan bahwa proses *handshake* terjadi saat pengambilan data. Diawali dengan *client* mengirim pesan *hello,* kemudian dibalas oleh server. Proses selanjutnya adalah *key exchange*.



Gambar 6.7 Isi paket data dalam pesan *key exchange* pada skenario 3

Gambar 6.7 menunjukkan bahwa terjadi pertukaran kunci atau *key exchange* antara *client* dan *server*. Proses ini merupakan mekanisme *authentication* yang terdapat pada TLS/SSL. Pada bagian ini, proses lanjutan dari *handshake* dimulai. Proses tersebut juga membawa kode pertukaran metode enkripsi yang diberi label *change cipher spec protocol* yang digunakan agar metode enkripsi yang digunakan sama. Ketika proses *handshake* berhasil, maka pengiriman pesan dapat dilakukan.



Gambar 6.8 Isi paket data pada skenario 3

Pada gambar 6.8, menunjukkan bahwa pengiriman data sudah bisa dilakukan dan data yang di-*capture* tidak bisa terbaca dengan jelas. Hal ini menunjukkan bahwa mekanisme keamanan TLS/SSL yang diimplementasikan telah berjalan dengan benar dan dapat mengamankan data yang dikirimkan oleh *middleware*.

6.2.2 Pengaruh mekanisme keamanan terhadap kinerja komunikasi

Pengimplementasian mekanisme keamanan akan mempengaruhi kinerja dari sistem yang digunakan. Pada bagian pengolahan data, telah diperlihatkan bagaimana pengaruh dari setiap mekanisme keamanan terhadap *packet loss, delay*, dan *jitter*. Berdasarkan pengolahan data di atas, perbandingan pengaruh mekanisme keamanan tersebut dapat dirangkum pada tabel berikut:

Tabel 6.19 Perbandingan antar mekanisme

Mekanisme keamanan	CPU Usage (%)	Memory Usage (%)	Rata-rata packet loss (%)	Rata-rata delay (detik)	Rata-rata Jitter (detik)
Tanpa mekanisme keamanan	0,263330052	2,849116228	1,333333333	0,156723573	0,006562287
AES-256	0,404019691	4,428192053	1	0,168367841	0,008928956
TLS/SSL	0,466400697	4,482035297	1,333333333	0,089008423	0,003208877

Pada Tabel 6.19, dapat dilihat perbandingan antara setiap mekanisme keamanan dalam hal kinerja komunikasi. Pada parameter *packet loss,* mekanisme keamanan AES-256 menghasilkan nilai yang lebih sedikit jika dibandingkan dengan TLS/SSL, yaitu 1%. Akan tetapi pada parameter *delay* dan *jitter,* mekanisme keamanan TLS/SSL menghasilkan nilai yang lebih baik jika dibandingkan dengan AES-256. Penggunaan CPU dan memory tidak memiliki perbedaan yang besar diantara AES-256 dan TLS/SSL meskipun keduanya menyebabkan kenaikan penggunaan CPU dan Memory dibandingkan tanpa mekanisme keamanan.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil pengambilan data dan analisa yang dilakukan, dapat ditarik kesimpulan seperti berikut:

1. Berdasarkan hasil pengambilan data, pengiriman data tanpa menggunakan mekanisme keamanan menghasilkan data yang sangat mudah dibaca oleh seseorang yang menyadap jaringan. Pengiriman data menggunakan mekanisme keamanan AES-256 dan TLS/SSL menghasilkan data yang tidak bisa dibaca karena data yang dikirim telah dienkripsi. Akan tetapi, terdapat tambahan proses *key exchange* pada TLS/SSL sebagai mekanisme *authentication* yang membuat mekanisme ini lebih aman dibandingkan dengan AES-256.
2. Berdasarkan hasil pengolahan data, pengaruh dari penggunaan AES-256 pada kinerja middleware adalah penurunan jumlah *packet loss* dari 1,333333333% menjadi 1%, peningkatan rata-rata *delay* dari 0,156723573 detik menjadi 0,168367841 detik, dan peningkatan rata-rata *jitter* dari 0,006562287 detik menjadi 0,008928956 detik. Sedangkan pengaruh TLS/SSL terhadap kinerja *middleware* adalah tidak ada peningkatan jumlah *packet loss* yaitu 1,333333333% dan penurunan rata-rata *jitter* menjadi 0,003208877 detik, lebih kecil dibandingkan dengan AES-256. Pada parameter *delay*, peningkatan rata-rata *delay*-nya lebih sedikit dibandingkan AES-256, yaitu 0,089008423 detik.
3. Kedua mekanisme keamanan dapat mengamankan paket yang dikirim menggunakan metode enkripsi. Kedua mekanisme juga tidak memiliki perbedaan yang besar pada pengaruh penggunaan CPU dan memory. Tetapi, TLS/SSL memberikan mekanisme keamanan yang lebih kuat dengan proses *key exchange* sebagai metode *authentication*. Ditambah dengan kinerja TLS/SSL pada data di atas, rata-rata *delay* dan jitter lebih baik. Jadi, TLS/SSL lebih direkomendasikan untuk digunakan sebagai mekanisme keamanan pada komunikasi *end-to-end* IoT *middleware* dengan *subscriber* berbasis protokol HTTP, dibandingkan dengan AES-256.

7.2 Saran

Terdapat beberapa saran dari penulis untuk penelitian selanjutnya, yaitu penelitian selanjutnya dapat menggunakan mekanisme keamanan yang berbeda, arsitektur *middleware* atau lingkungan uji yang berbeda, parameter uji yang berbeda, serta dapat menganalisis mekanisme keamanan pada jaringan yang memiliki lebih dari 1 koneksi.

DAFTAR PUSTAKA

Abdur R.M, Milojevic-jervic M, Palade A, Clake S., 2016. *Middleware for Internet of Things: A Survey*.

Ahmad M, Taj S, Mustafa T, Asri Md., 2012. *Performance Analysis of Wireless Network with the impact of Security Mechanisms*. Pakistan.

Anne H. H. Ngu, M. G., 2016. *IoT Middleware: A Survey on Issues and Enabling Technologies*. IEEE Internet of Things Journal, Vol. X, No. X.

Anwari H., 2017. *Pengembangan IoT Middleware Berbasis Eventbase Dengan Potokol Komunikasi CoAP, MQTT dan Websocket.* Malang. Universitas Brawijaya.

Atzori L, Iera A, Morabito G., 2010. *The Internet of Things: A Survey*.

Bhajaj R.D, Gokhale Dr U.M., 2016. *AES ALGORITHM FOR ENCRYPTION*. Nagpur. G.H. Raisoni Institute of Engineering and Technology for Women.

Behren R, Ahmed A., 2017. *Internet of Things: An End-to-End Security Layer*. Paris. Innovation in Clouds, Internet and Networks (ICIN).

Eko S.C, Simanjuntak P., 2017. *Potential Threat Analysis Hypertext Transfer Protocol and Secure Hypertext Transfer Protocol of Public WiFi Users (Batam Case)*. International Journal of Scientific & Engineering Research. Vol.8. ISSN 2229-5518.

Fali O.A, Irawan B., 2014. *Implementasi FTP Server dengan Metode Transfer Layer Security untuk Keamanan Transfer Data Menggunakan CentOS 5.8*. Ogan Ilir. Universitas Sriwijaya.

Fahrur R.M., 2017. *Analisis Performansi dan Skalabilitas pada Event-Based IoT Middleware*. Malang. Universiter Brawijaya.

Fauzi A.A., 2016. *Implementasi dan Analisis Keamanan Pesan Text Dengan Algoritma AES-256 bit Pada Aplikasi Chatting Berbasis Android*. Semarang. Universitas Dian Nuswantoro.

Islam N, Chandra B.C, Hasan J, Islam S.A., 2016. *Quality of Service Analysis of Ethernet Network Based on Packet Size*. Bangladesh.

Maqsood, Faiqa & Ahmed, Muhammad & Mumtaz, Muhammad & Shah, Munam. (2017). Cryptography: A Comparative Analysis for Modern Techniques. International Journal of Advanced Computer Science and Applications. 8. . 10.14569/IJACSA.2017.080659.

Michael H.B. 2009. End-to-End Security. *The Internet Protocol Journal*, [online] Tersedia di: <https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-45/123-security.html> [diakses pada 20 November 2017]

Nabeel M., 2017. *The Many Faces of End-to-End Encryption and Their Security Analysis*. Qatar. Qatar Computing Research Institute.

Nawir M, Amir A, Yaakob N, Bi Lynn O., 2016. *Internet of Things (IoT): Taxonomy of Security Attack*. Phuket: 3rd International Conference on Electronic Design (ICED).

Park J, Hwang H, Yun J, Moon I., 2014. *Study of HTML5 WebSocket for a Multimedia Communication*. Korea University of Technology and Education.

Rajra M.B.B, J Deepa ME.A., 2015. *A Survey on Network Security Attacks and Prevention Mechanism*. Ponjesly College of Engineering Nagercoil.

Ramdhansya A.F, Ariyanto E, Nuha H.H., 2014. *IMPLEMENTASI ADVANCED ENCRYPTION STANDARD (AES) PADA SISTEM KUNCI ELEKTRONIK KENDARAAN BERBASIS SISTEM OPERASI ANDROID DAN MIKROKONTROLER ARDUINO*. Bandung. Universitas Telkom.

Setiawan D, Triyono J, Iswahyudi C., 2017. *ANALISIS PERBANDINGAN QUALITY OF SERVICE (QOS) FIRMWARE DEFAULT DAN FIRMWARE OPENWRT PADA ACCESS POINT TP-LINK MR3020*. Yogyakarta. IST AKPRIND

Singh R, Kumar S., 2016. *An Overview of World Wide Web Protocol (Hypertext Transfer Protocol and Hypertext Transfer Protocol Secure)*. International Journal of Advanced Research in Computer Science and Software Engineering. Vol. 6. ISSN: 2277 128X.

Skvorc D, Horvat M, Srbljic S., 2014. *Performance Evaluation of Websocket Protocol for Implementation of Full-Duplex Web Streams*. Opajita. University of Zagreb.

Suartana I.M, Wahanani H.E, Darminta A.M., 2017. *Analisa Pengaruh Routing Protokol Dan Mekanisme Keamanan Pada Kualitas Layanan Multimedia pada Jaringan IP*. Surabaya. Universitar Negeri Surabaya.

Tehupeiory N, Chandra D.W., 2016. *Analisis Perbandingan Mekanisme Secure Socket Layer (SSL) dan Transfer Layer Security (TLS) Pada Koneksi File Transfer Protokol (FTP) Server Ubuntu*. Salatiga. Universitas Kristen Satya Wacana.

S. Turner, "Transport Layer Security," in *IEEE Internet Computing*, vol. 18, no. 6, pp. 60-63, Nov.-Dec. 2014. doi: 10.1109/MIC.2014.126