Enkripsi Asimetris Pada Transfer Data Antar Perangkat IoT Menggunakan Protokol HTTP dan MQTT

Rahmad Yesa Surya - 13515088
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Insttitut Teknologi Bandung
Jalan Ganesha Nomor 10, Bandung
13515088@std.stei.itb.ac.id

Abstrak—Dengan semakin bertambahnya jumlah alat IoT, semakin bertambah pula jumlah data yang ditransfer antar alat-alat tersebut. Hal ini menimbulkan pertanyaan terhadap keamanan transfer data pada alat IoT: apakah data yang ditransfer aman dan rahasia? Protokol komunikasi yang umum digunakan pada IoT, MQTT, tidak menyediakan enkripsi data sehingga data ditransfer dalam bentuk plainteks. Makalah ini membahas rancangan dan implementasi enkripsi data menggunakan algoritma enkripsi yang ringan, ECC-EG, pada transfer data di IoT. Rancangan arsitektur komunikasi yang digunakan masih tetap sama, yakni mirip dengan MQTT. Namun, implementasi dilakukan dengan HTTP sehingga data dapat dienkripsi. Walaupun secara keseluruhan proses pengiriman data yang terenkripsi ini memerlukan waktu lebih banyak daripada yang tidak terenkripsi, alternatif ini dapat dilakukan untuk meningkatkan keamanan proses transfer data pada IoT.

Kata kunci—*MQTT, transfer data, IoT, enkripsi*

I. PENDAHULUAN

Teknologi saat ini telah melahirkan terobosan baru yang mampu menghubungkan benda-benda fisik ke dalam jaringan, yakni *Internet of Things* (IoT). IoT membuat benda-benda tersebut terhubung satu sama lain dan memfasilitasi adanya "komunikasi", misalnya transfer data dari satu ke yang lain. Istilah IoT sendiri ditemukan pertama kali pada 1999 oleh Kevin Ashton dari Procter & Gamble untuk menyebut *Internet for Things*[1]. Pada saat itu, yang dimaksud *Internet for Things* merujuk kepada penggunaan *Radio Frequency Identification* (RFID) agar komputer dapat mengatur semua benda-benda fisik[2]. Namun demikian, pada saat ini IoT telah berkembang lebih canggih daripada hanya sekedar penggunaan RFID untuk mengontrol benda-benda fisik dengan komputer.

Saat ini, IoT telah melahirkan banyak produk di pasaran, misalnya *smartwatch* dan alat-alat *smarthome*. Berdasarkan laporan Gartner, jumlah alat-alat IoT yang digunakan di seluruh dunia pada tahun 2020 mencapai 20 miliar unit[3]. Hal tersebut mengindikasikan IoT telah memiliki popularitas yang tinggi dan skala penggunaan yang luas. Namun demikian, terdapat hal utama yang menjadi kekhawatiran dalam penggunaan IoT, yakni masalah keamanan (secara spesifik, keamanan dalam transfer data).

Hampir seluruh alat IoT memiliki fungsionalitas untuk merekam data tertentu. Pada *smartwatch*, data yang direkam diantaranya adalah jumlah langkah yang telah ditempuh atau jumlah detak jantung pengguna pada suatu waktu. Pada alat *smarthome*, misalnya *smart thermostat*, data yang direkam adalah penggunaan energi dan kondisi suhu ruangan. Data-data tersebut pada umumnya akan ditransfer ke *server* terpusat sehingga dapat diolah menjadi informasi yang bermanfaat bagi pengguna. Sebagaimana pada umumnya transfer data pada jaringan, terdapat kemungkinan bahwa data dapat disadap pada saat transfer sedang berlangsung. Untuk menjaga kerahasiaan, data tersebut perlu dienkripsi sehingga tetap aman ketika ditransfer.

Hal pertama yang menjadi perhatian terhadap enkripsi pada IoT adalah algoritma enkripsi yang digunakan. Algoritma tersebut harus dirancang untuk tetap menghasilkan enkripsi yang kuat, namun disisi lain dapat dieksekusi pada perangkat IoT yang pada umumnya memiliki kemampuan komputasi rendah. Salah satu algoritma yang didesain ringan secara komputasi untuk memenuhi kebutuhan ini adalah *Elliptic Curve Cryptography El Gamal* (ECC-EG). Algoritma ini termasuk kepada algoritma enkripsi asimetris yang menggunakan kunci privat dan kunci publik.

Hal kedua yang menjadi perhatian adalah peningkatan keamanan. Pada dasarnya, untuk melakukan enkripsi dan dekripsi secara asimetris, masing-masing pihak harus memiliki kunci privat dan kunci publik. Untuk meningkatkan keamanan pada penggunaannya di IoT, kunci privat dan kunci publik tersebut sebaiknya diganti secara periodik.

Makalah ini membahas rancangan dan implementasi algoritma ECC-EG pada IoT, berdasarkan protokol standar MQTT, yang diimplementasikan dengan HTTP.

II. DASAR TEORI

A. *Elliptic Curve Cryptography* (ECC) Pada Medan Terbatas

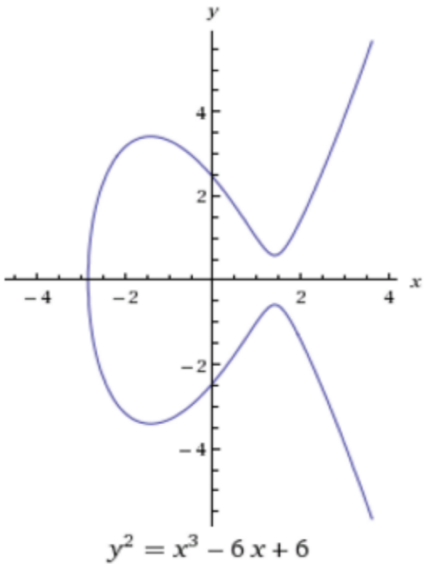
ECC adalah sistem kriptografi kunci publik yang didasarkan pada struktur logaritma diskrit kurva elips pada medan yang terbatas. ECC dikenal dengan ukuran kunci yang kecil, enkripsi yang cepat, dan implementasi yang lebih

efisien untuk tingkat keamanan yang sama jika dibandingkan dengan sistem kriptografi kunci publik lain, misalnya RSA[4].

ECC menggunakan kurva elips pada medan terbatas dengan bentuk persamaan:

$$y^2 = x^3 + ax + b \ (mod\ p)$$

Titik-titik yang memenuhi persamaan diatas akan membentuk sebuah kurva. Nilai *a* dan *b* pada persamaan akan menentukan bentuk dari kurva tersebut. Contoh persamaan dan kurva elips yang terbentuk dapat dilihat pada Gambar 2.1.



Gambar 2.1 Persamaan dan kurva elips yang dihasilkan (Sunuwar, Rosy dkk. 2015)

Operasi aritmatika yang digunakan pada kurva elips berbeda dengan operasi aritmatika pada umumnya. Untuk melakukan penjumlahan dua titik pada kurva, *P* dan *Q*, perlu digambar sebuah garis yang melalui dua titik tersebut. Garis ini akan berpotongan dengan kurva pada titik ketiga, -*R*. Titik ketiga ini akan direfleksikan pada sumbu X untuk pada akhirnya memperoleh titik *R*. Dengan demikian, *R* adalah hasil penjumlahan titik *P* dan *Q*. Jika *P* dan *Q* vertikal, sehingga *Q* = (-*P*), maka garis yang melalui dua titik tersebut tidak akan berpotongan dengan kurva, sehingga tidak ditemukan titik ketiga. Dengan demikian, hasil penjumlahannya adalah tidak terhingga[4].

Operasi lain adalah menjumlahkan titik *P* ke dirinya sendiri. Untuk melakukan itu, perlu digambar sebuah garis singgung (garis tangen) pada kurva yang menyinggung *P*. Jika *P* tidak berada pada sumbu X, maka garis singgung akan memotong kurva pada titik ketiga, -*R*. Sama seperti operasi sebelumnya, titik -*R* direfleksikan terhadap sumbu X untuk mendapatkan hasil akhir *R*. Operasi menjumlahkan titik *P* ke dirinya sendiri disebut dengan *point doubling*. Operasi ini pada umumnya dilakukan untuk perkalian titik pada kurva elips[4].

ECC digunakan dalam kriptografi karena memiliki kelebihan yang berkaitan dengan permasalahan logaritma diskrit. Permasalahan logaritma diskrit: jika *p* adalah bilangan prima, sedangkan *g* dan *y* adalah bilangan bulat sembarang, maka carilah nilai *x* sedemikian sehingga

$$g^x \equiv y \ (mod\ p)$$

Pada kurva elips, permasalahan tersebut diadopsi menjadi: jika *P* dan *Q* adalah dua titik pada kurva elips *G,* maka carilah nilai *k* sehingga

$$P \circ k = Q$$

Permasalahan logaritma diskrit tersebut merupakan permasalahan yang secara komputasi sulit dipecahkan. Dengan demikian, penerapannya pada kriptografi dapat meningkatkan keamanan[4][6].

B. Algoritma ElGamal Dengan ECC (ECC-EG)

Algoritma ElGamal ditemukan oleh Taher El Gamal pada tahun 1985. Konsep algoritma ini adalah menyembunyikan pesan *m* menggunakan a^k dan b^k yang mana *a* dan *k* adalah bilangan acak. Dalam hal ini, $b = a^a$ yang mana a adalah bilangan rahasia yang hanya diketahui oleh penerima pesan. Berikut adalah contoh penggunaan algoritma ElGamal.

Misalkan terdapat tiga pihak, Alice, Bob, dan Charlie. Alice ingin mengirimkan pesan *m* kepada Bob melalui saluran yang dapat diintervensi oleh Charlie. Pertama, Bob akan memilih sebuah bilangan rahasia yang hanya diketahui olehnya, a. Kemudian, Bob memilih bilangan prima besar *p* dan bilangan acak *a*. Selanjutnya, Bob menghitung $b = a^a$ dan membuat (*a, b, p*) publik. Artinya, (*a, b, p*) dapat diketahui oleh siapapun. Untuk mengirimkan pesan, Alice memilih bilangan acak *k* terlebih dahulu. Selanjutnya, ia mengirimkan pesan (a^k, $b^k m$) kepada Bob. Jika Charlie mendapatkan pesan tersebut, Charlie perlu menyelesaikan permasalahan logaritma diskrit untuk mengetahui pesan asli. Namun demikian, untuk mengetahui pesan asli, Bob dapat menggunakan bilangan rahasianya, a[4].

$$(\alpha^k)^{-a} * (b^k m) = (\alpha^a)^{-k} * (b^k m) = (b^{-k}) * (b^k m) = m$$

Pada penerapan algoritma ElGamal dengan ECC, *a* dan *b* yang dipilih adalah titik-titik yang berada pada kurva elips. Operasi perkalian diganti dengan operasi penjumlahan ECC, sedangkan operasi perpangkatan diganti dengan operasi perkalian ECC. Dengan skenario pengiriman pesan yang sama seperti diatas, Bob akan memilih terlebih dahulu kurva elips C yang memenuhi persamaan kurva elips dan titik rahasia a. Selanjutnya, Bob akan memilih salah satu titik yang berada pada kurva elips secara acak, *a* (x_1, y_1). Kemudian, Bob menghitung titik *b* (x_2, y_2), yang mana $b = a \bullet a$, yaitu *a* dikalikan dengan dirinya sendiri sebanyak a kali. Bob akan membuat (*a, b, p*) publik. Ketika Alice ingin mengirimkan pesan, ia memilih bilangan acak *k* terlebih dahulu. Kemudian, Alice menghitung dua titik baru pada kurva elips, *r* (x3, y3) dan *t* (x4, y4).

$$r = (x_3, k \circ \alpha)$$

$$t = (x_4, m + k \circ b)$$

Pada perhitungan diatas, perkalian dan penjumlahan yang dimaksud dilakukan dengan operasi aritmatika ECC. *r* dan *t* adalah pesan yang dikirimkan oleh Alice ke Bob. Jika Charlie memperoleh pesan tersebut, Charlie harus menyelesaikan permasalahan logaritma diskrit untuk mengetahui pesan asli. Untuk mengetahui pesan asli, Bob dapat melakukan perhitungan sebagai berikut[4].

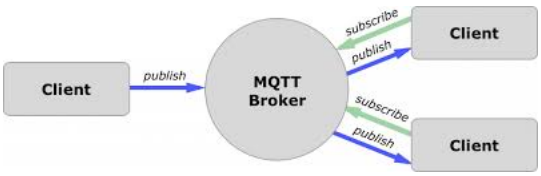
$$y4 - (a \circ y3) = (m + k \circ b) - a \circ (k \circ \alpha) =$$

$$m + a \circ k \circ \alpha - a \circ k \circ \alpha = m$$

C. *Message Queuing Telemetry Transport* (MQTT)

MQTT adalah *messaging protocol* berbasis *publish-subscribe* yang didesain untuk beroperasi pada konektivitas dengan kualitas yang rendah, misalkan pada lokasi yang terpencil dengan *network bandwidth* yang terbatas. Protokol ini dibuat pada tahun 1999 oleh Andy Stanford-Clark dari IBM dan Arlen Nipper dari Cirrus Link[5].

Cara kerja MQTT mirip dengan arsitektur *client-server* yang mana terdapat komunikasi tersentralisasi yang dikoordinasi oleh *server*. Pada MQTT, *client* terdiri dari pihak-pihak yang bisa menjadi *publisher* atau *subscriber*. Selanjutnya, terdapat *server* atau *broker* yang menjadi penghubung antar *publisher* dan *subscriber*. Secara sederhana, *publisher* pada MQTT mengirimkan pesan melalui *broker* terlebih dahulu. Selanjutnya, *broker* akan mengirimkan pesan tersebut kepada *subscriber*. Ilustrasi cara kerja MQTT dapat dilihat pada Gambar 2.2.



Gambar 2.2 Diagram cara kerja MQTT

Informasi pada MQTT diatur berdasarkan topik. Topik-topik tersebut didefinisikan terlebih dahulu pada *broker*. Ketika sebuah *publisher* ingin mengirimkan pesan, maka ia mengirimkan pesan tersebut kepada *broker* dengan topik tertentu. Ketika *broker* menerima pesan tersebut, *broker* akan mendistribusikan pesan tersebut ke seluruh *subscriber* yang telah men-*subscribe* topik tersebut. Dengan demikian, *publisher* tidak perlu mengetahui jumlah atau lokasi dari *subscriber*, sedangkan *subscriber* tidak perlu menyimpan informasi apapun tentang *publisher*. Hal ini membuat *publisher* dan *subscriber* sebagai dua pihak yang independen.

Jika *broker* menerima pesan pada sebuah topik yang tidak memiliki *subscriber*, maka *broker* akan secara otomatis menghapus topik tersebut, kecuali jika *publisher* mengindikasikan bahwa topik tersebut harus tetap disimpan. Dengan demikian, *subscriber* akan selalu mendapatkan pesan paling baru dari *publisher*. *Client*, baik *publisher* maupun *subscriber* berinteraksi dengan *broker*. Pada sebuah sistem mungkin terdapat beberapa *broker* yang dapat berkomunikasi satu sama lain, misalkan untuk menyinkronisasi topik-topik yang terdaftar diantara *broker*.

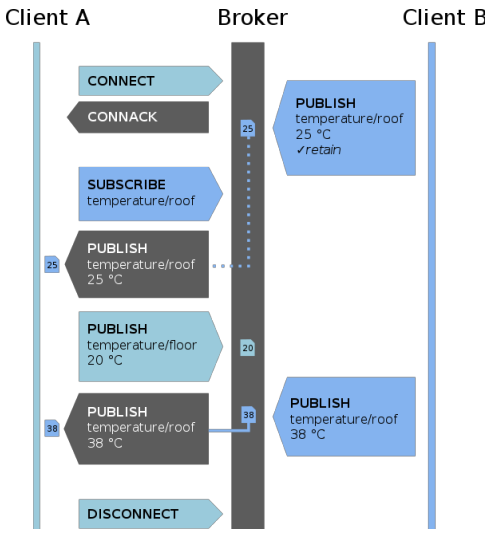
Sebuah pesan MQTT paling kecil berukuran 2 *bytes*, sedangkan paling besar berukuran 256 *megabytes*. Terdapat empatbelas jenis pesan pada MQTT dengan fungsi yang bervariasi. Tiga jenis pesan yang umum diketahui adalah:

1. CONNECT, berarti menunggu sampai koneksi TCP antara *client* dan *server* terbentuk.
2. DISCONNECT, berarti menunggu *client* menyelesaikan proses yang dijalankan, kemudian mengindikasikan bahwa koneksi TCP akan diputus.
3. PUBLISH, berarti mengirimkan data antara *client* dan *server*

Pada MQTT, koneksi antara *client* dan *server* memiliki ukuran *Quality of Service* (QoS) yang diklasifikasikan menjadi tiga tingkatan:

1. *At Most Once*: pesan dikirimkan sekali saja dan *client* atau *broker* tidak akan melakukan penjaminan bahwa pesan tersebut sampai pada tujuan.
2. *At Least Once*: pesan dikirimkan berulang kali hingga *client* atau *broker* (yang mengirimkan pesan) menerima *acknowledgement* bahwa pesan telah diterima.
3. *Exactly Once*: pengirim dan penerima pesan melakukan *two-level handshake* untuk memastikan bahwa hanya satu *copy* pesan saja yang sampai.

Pada dasarnya, MQTT beroperasi diatas *layer* TCP, sehingga komunikasi yang terbentuk antara *client* dan *broker* adalah *connection-oriented*. Namun demikian, ada beberapa varian MQTT, misalnya MQTT-SN yang tidak menggunakan TCP, melainkan UDP atau Bluetooth. Data yang dikirimkan oleh MQTT pada dasarnya tidak dienkripsi sama sekali, sehingga data berbentuk plainteks saja. Gambar 2.3 mengilustrasikan komunikasi antara *client* melalui *broker* pada MQTT.



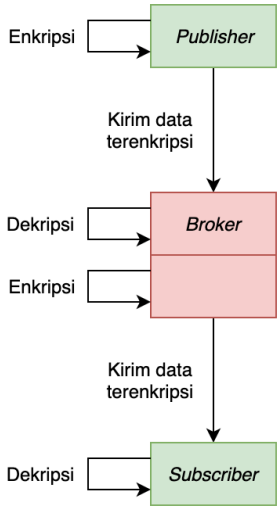
Gambar 2.3 Komunikasi pada MQTT
(Sumber: http://upload.wikimedia.org)

III. ANALISIS DAN RANCANGAN

Pada bagian ini, dirancang sebuah model komunikasi protokol MQTT dimana data yang dikirim adalah data yang dienkripsi dengan algoritma ECC-EG. Jika diperhatikan lebih lanjut, komunikasi pada protokol MQTT adalah komunikasi yang terjadi antara dua pihak saja, namun terjadi lebih dari satu kali. Misalkan terdapat *client* A sebagai *publisher*, *client* B sebagai *subscriber*, dan *broker* C. Ketika *client* A akan mengirimkan data ke *client* B, komunikasi yang terjadi adalah dari *client* A, kemudian ke *broker* C, baru kemudian ke *client* B. Pada contoh tersebut, terdapat komunikasi antara dua pihak yang dilakukan sebanyak dua kali, yakni A ke C dan C ke A. Dengan demikian, jika diasumsikan terdapat sebuah *publisher,* sebuah *broker,* dan N buah *subscriber*, maka jumlah komunikasi antara dua pihak yang terjadi adalah T, yang mana:

$$T = 1 + N$$

Pada makalah ini, enkripsi dirancang untuk dilaksanakan setiap komunikasi antara dua pihak terjadi. Dengan asumsi seperti diatas, maka akan terjadi $1 + N$ enkripsi dan dekripsi. Secara umum, ilustrasi proses enkripsi dan dekripsi yang dijelaskan dapat dilihat pada Gambar 3.1.

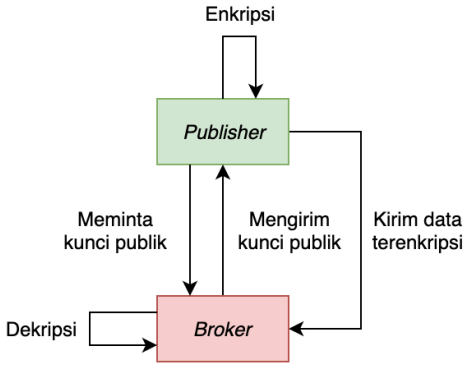


Gambar 3.1. Ilustrasi proses enkripsi dan dekripsi pada pengiriman pesan dari *publisher* ke *subscriber*

A. Pengiriman Data Antara *Publisher* dan *Broker*

Pada pengiriman data antara *publisher* dan *broker*, proses enkripsi dan dekripsi dilakukan masing-masing sebanyak satu kali. Enkripsi dilakukan oleh *publisher* ketika akan mengirimkan data kepada *broker*, sedangkan dekripsi dilakukan oleh *broker* ketika menerima data dari *publisher*. Untuk melakukan proses ini, *publisher* harus memiliki kunci publik yang berpasangan dengan kunci privat milik *broker*. Hal ini merupakan syarat wajib agar proses ini terjadi.

Broker tidak menyimpan informasi tentang *publisher* yang berada pada sistem, sehingga *client* manapun dapat menjadi *publisher*. Yang *broker* perlu ketahui adalah bahwa pesan yang ia terima adalah pesan yang terenkripsi dengan kunci publiknya. Dengan demikian, pada awalnya, *broker* perlu membangkitkan terlebih dahulu kombinasi kunci privat dan kunci publik. Kemudian, ketika sebuah *publisher* ingin mengirimkan data, ia terlebih dahulu meminta kunci publik *broker*. Ilustrasi proses ini dapat dilihat pada Gambar 3.2.

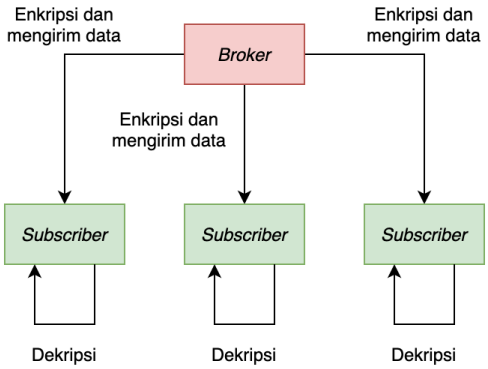


Gambar 3.2. *Publisher* meminta kunci publik kepada *broker* sebelum mengirimkan data

B. Pengiriman Data Antara *Broker* dan *Subscriber*

Pada pengiriman data antara *broker* dan *subscriber*, proses enkripsi dan dekripsi masing-masing dilakukan sebanyak jumlah *subscriber*. Enkripsi dilakukan oleh *broker* ketika akan mengirimkan data kepada *subscriber*, sedangkan dekripsi dilakukan oleh *subscriber* ketika menerima data dari *broker*. Untuk melakukan proses ini, *broker* harus memiliki kunci publik yang berpasangan dengan kunci privat milik *subscriber*.

Berbeda dengan *publisher,* informasi tentang *subscriber* disimpan oleh *broker*. Hal ini sudah secara *default* dilakukan oleh MQTT karena untuk mengirimkan data ke *subscriber*, *broker* perlu mengetahui *subscriber* mana saja yang men-*subscribe* topik tertentu. Informasi tentang *subscriber* dan topik ini disimpan bersama dengan kunci publik setiap *subscriber*. Proses ini dilakukan ketika *subscriber* men-*subscribe* topik tertentu.



Gambar 3.2 *Broker* mengenkripsi data sebaganyak jumlah *subscriber* dan mengirimkannya ke *subscriber* yang bersangkutan

Ketika *broker* menerima data dari *publisher* dan telah mendekripsinya, *broker* kemudian menemukan terlebih dahulu daftar *subscriber* yang terdaftar pada topik yang diinginkan. Untuk setiap *subscriber, broker* akan melakukan enkripsi dengan kunci publik *subscriber* tersebut dan mengirimkannya ke *subscriber*. Ilustrasi proses ini dapat dilihat pada Gambar 3.2.

IV. IMPLEMENTASI

Pada bagian ini, implementasi dilakukan sesuai dengan analisis dan rancangan yang telah dibuat pada Bab III. Namun demikian, terdapat hal yang perlu diperhatikan, yakni protokol MQTT tidak mengizinkan sebuah pesan di-*intercept* dan dimodifikasi ditengah transmisinya dari *publisher* ke *subscriber*. Hal ini tentu tidak mendukung rancangan yang dibuat pada Bab III. Alih-alih menggunakan MQTT untuk mengimplementasikan rancangan, protokol HTTP digunakan sehingga dapat diperoleh implementasi sesuai rancangan. Dalam hal ini, rancangan dasar dan terminologi-terminologi yang digunakan tetap menggunakan milik MQTT, namun pengimplementasiannya menggunakan HTTP.

Batasan-batasan pada implementasi adalah sebagai berikut:

1. Perangkat yang digunakan adalah Raspberry Pi 3
2. Algoritma ECC-EG diimplementasikan menggunakan bahasa pemrograman Python
3. HTTP *server* yang digunakan baik pada *client* maupun *broker* adalah Flask
4. Persamaan kurva elips yang digunakan adalah

$$y^2 = x^3 + 7$$

5. *Generator point* yang digunakan pada ECC-EG adalah G (Gx, Gy).

```
Gx =
55066263022277343669578718895168534326250603453777594175500187360389116729240
Gy =
32670510020758816978083085130507043184471273380659243275938904335757337482424
```

A. Pembangkitan Kunci Pada *Broker* dan *Client*

Pada *broker* dan *client*, kunci privat dibangkitkan dengan menggunakan *library* random pada Python menggunakan *seed* 256. Kemudian kunci publik pasangannya dibangkitkan dengan menggunakan kunci privat tersebut.

```
priv_key = random.getrandbits(256)
pub_key = ecc_multiply(G, priv_key)
```

B. Pertukaran Kunci Pada *Broker* dan *Client*

Pada dasarnya, ketika bergabung ke jaringan, setiap *client* pasti akan menghubungi *broker* terlebih dahulu. Jika ia adalah *publisher*, maka ia perlu meminta kunci publik dari *broker*. Jika ia adalah *subscriber,* maka ia perlu mengirimkan kunci publiknya ke *broker* dan mendaftar pada topik yang diiingikan.

Jika *client* adalah *publisher*, maka ia akan mengirimkan GET *request* ke *broker* pada REST API **/request_key**. Ketika *broker* mendapatkan *request* tersebut, *broker* akan mengirimkan kunci publiknya pada *response* dalam format JSON.

```
broker = “192.168.0.5”
url = broker + “/request_key”
req = requests.get(url)
res = req.text
```

Jika *client* adalah *subscriber*, maka ia akan mengirimkan POST *request* ke *broker* pada REST API **/send_key**. Parameter-parameter yang harus dipenuhi pada POST *request* tersebut adalah kunci publik dan daftar topik yang ingin di-*subscribe*.

```
broker = “192.168.0.5”
url = broker + “/send_key”
payload = {
    “public_key”: ……,
    “topics”: [……, ……, ……]
}
req = requests.post(url, json.dumps(payload))
res = req.text
```

Saat menerima *request* tersebut, *broker* akan mencatat *address* daripada *subscriber* bersama dengan topik yang di-*subscribe* dan kunci publiknya.

C. Pengiriman Data Dari *Publisher* ke *Subscriber*

Ketika *publisher* ingin mengirimkan data ke *broker*, *publisher* mengenkripsi data terlebih dahulu menggunakan kunci publik *broker*, kemudian mengirimkan POST *request* ke *broker* pada REST API **/send_data**. Parameter-parameter yang harus dipenuhi pada POST *request* tersebut adalah data terenkripsi dan topik.

```
broker = “192.168.0.5”
url = broker + “/send_data”
payload = {
    “data”: ……,
    “topic”: ……
}
req = requests.post(url, json.dumps(payload))
res = req.text
```

Setelah *broker* menerima *request* tersebut, *broker* akan mendekripsi data tersebut menggunakan kunci privat miliknya. Kemudian, *broker* akan mencari daftar *subscriber* yang terdaftar pada topik yang dispesifikasikan. Untuk setiap *subscriber* yang ada, *broker* akan mengenkripsi data dengan kunci publik *subscriber* yang bersangkutan dan mengirimkannya melalui POST *request* pada REST API **/forward_data**.

Setelah *subscriber* menerima *request* tersebut, *subscriber* akan mendekripsi data tersebut menggunakan kunci privat miliknya.

```
for subscriber in topic:
  data = encrypt(data, subscriber.pub_key)
  url = subscriber.address + "/forward_data"
  payload = {
      "data": data,
  }
  req = requests.post(url, json.dumps(payload))
  res = req.text
```

V. HASIL UJICOBA

Pada bagian ini, ujicoba dilakukan terhadap implementasi yang telah dilakukan sesuai pada Bab IV. Ujicoba dilakukan dengan membandingkan waktu yang ditempuh untuk seluruh tahap pengiriman pesan dari *publisher* ke *subscriber* antara saat menggunakan sistem yang terenkripsi dan sistem yang tidak terenkripsi. Ujicoba dilakukan untuk mengetahui waktu yang diperlukan pada setiap tahapan pengiriman pesan sebagai berikut:

1. Pembangkitan kunci pada *broker* dan *client*
2. Permintaan kunci oleh *publisher* kepada *broker*
3. *Subscription* oleh *subscriber* kepada *broker*
4. Enkripsi oleh *publisher*
5. Pengiriman data dari *publisher* kepada *broker*
6. Dekripsi oleh *broker*
7. Pengiriman data dari *broker* kepada *subscriber* (pada sistem terenkripsi, hal ini meliputi enkripsi yang dilakukan untuk setiap *subscriber*)
8. Dekripsi oleh *subscriber*

Dari tahapan-tahapan pengiriman pesan yang akan diuji, sebenarnya terdapat beberapa tahapan yang memiliki ketergantungan terhadap variabel luar. Tahap 2, 3, 5, dan 7 memiliki ketergantungan terhadap kualitas jaringan yang digunakan untuk pengetesan. Tahap 4, 6, dan 8 memiliki ketergantungan terhadap panjang data yang dienkripsi atau didekripsi. Kemudian, tahap 7 juga memiliki ketergantungan terhadap jumlah *subscriber* yang ada. Pada pengujian ini, variabel-variabel tersebut nilainya ditentukan sebagai berikut:

1. Jaringan yang digunakan memiliki kecepatan *download* 28.08 Mbps, kecepatan *upload* 18.08 Mbps, dan *ping* 3 ms.
2. Data yang dikirim (dienkripsi dan didekripsi) adalah *String* dengan panjang 30 karakter, yakni "Temperatur 100 Derajat Celcius".
3. Jumlah *publisher* dan *subscriber* masing-masing satu.

Tahap	Sistem Tidak Terenkripsi	Sistem Terenkripsi
1	0 ms	12 ms
2	0 ms	87 ms
3	84 ms	60 ms
4	0 ms	0,1 ms
5	85 ms	73 ms
6	0 ms	0,06 ms
7	110 ms	159 ms
8	0 ms	0,04 ms
Total	279 ms	391,2 ms

VI. KESIMPULAN

Berdasarkan implementasi dan eksperimen yang dilakukan, untuk ukuran sistem yang kecil, dengan jumlah *publisher* dan *subscriber* yang sedikit, perbedaan waktu yang diperlukan dalam pengiriman pesan secara keseluruhan masih bisa ditoleransi karena hanya berjarak kurang lebih 100ms. Namun demikian, untuk penggunaan sistem yang berukuran lebih besar, dengan jumlah *publisher* dan *subscriber* yang lebih banyak, serta ukuran data yang semakin besar, penggunaan metode ini perlu dievaluasi kembali karena peningkatan waktu yang diperlukan akan naik secara signifikan.

Makalah ini tidak menyimpulkan korelasi waktu antara sistem tidak terenkripsi dengan sistem terenkripsi (semisal linear atau eksponensial) karena jumlah skenario pengujian yang sedikit. Namun demikian, makalah ini menunjukkan salah satu alternatif cara yang dapat digunakan untuk mengamankan proses pengiriman data pada perangkat IoT.

REFERENSI

[1] Ashton, K. "*That 'Internet of Things' Thing*", 22 Juni 2009. Diakses pada 9 Mei 2019.

[2] Magrassi, P.; Berg, T. "*A World of Smart Objects*", 12 Agustus 2002.

[3] Knoll, L. "*Developing The Connected World of 2018 and Beyond*", 16 Maret 2018. Diakses pada 9 Mei 2019.

[4] Sunuwar, R.; Samal, S. K. "*Elgamal Encryption using Elliptic Curve Cryptography",* 9 Desember 2015.

[5] "*10th Birthday Party*". MQTT.org. Juli 2009. Diakses pada 9 Mei 2019.

[6] Munir, R. "Bahan Kuliah IF4020 Kriptografi: Algoritma Elgamal". 2018. Diakses pada 9 Mei 2019.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Mei 2019

Rahmad Yesa Surya