# Advanced Encryption Standard (AES) pada Modul Internet of Things (IoT)

# Advanced Encryption Standard (AES) on Internet of Things (IoT) Module

**Royyannuur Kurniawan Endrayanto [1], Adharul Muttaqin [2*], Raden Arief Setyawan [3]**
Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya
Jalan Mayjen Haryono 167 Malang 65145-Telp (0341) 567886
roy.en@student.ub.ac.id [1], adharul@ub.ac.id [2*], rarief@ub.ac.id [3]

***Abstrak*** **–** Pada modul IoT yang memerlukan enkripsi data tetapi tidak dilengkapi dengan *hardware accelerator* khusus untuk enkripsi, perlu menggantikan hardware tersebut dalam bentuk program. Akan tetapi penambahan program enkripsi diketahui dapat menimbulkan permasalahan pada modul IoT berbasis *embedded system* yang memiliki sumber daya terbatas. Dalam kajian ini dibahas algoritma enkripsi AES-128 yang diimplementasikan pada modul IoT *Particle Photon* yang belum memiliki *hardware accelarator*. Tujuan yang hendak dicapai adalah untuk mengetahui pengaruh dari penerapan AES-128 pada modul IoT. Hasil pengujian menunjukkan AES-128 yang diterapkan dapat berjalan baik dengan waktu enkripsi paling lama 398 mikrodetik dan *throughput* terkecil 301507,538 bit/detik. Hasil pengukuran beban terhadap penerapan enkripsi berupa penggunaan memori *flash* oleh program sebesar 16.024 *Byte* dengan penggunaan RAM sebesar 3.020 *Byte*.

***Kata Kunci****: Modul IoT, Sniffing, AES-128*

***Abstract –*** *IoT modules which need data encryption need to provide encrytpion procedure implemented in a software, and it is not equipped whith special encription hardware. However, the implementation of the encryption program can lead other problems especially if it is applied to IoT modules based on embedded systems which have limited resources. This study discusses the AES-128 encryption algorithm which is implemented in the form of a program and applied to the IoT Particle Photon module that does not yet have a hardware accelarator. The aim to be achieved is to determine the effect of the application of AES-128 on the IoT module. The test results show that AES-128 can run well with the longest encryption time of 398 microseconds and the smallest throughput of 301507,538 bits / second. The results of the load measurement of the application of encryption in the form of flash memory usage by the program amounted to 16,024 Bytes with the use of RAM of 3,020 Bytes.*

***Keywords****: IoT Module, Sniffing, AES-128.*

## 1. Pendahuluan

*Internet of Things* (IoT) merupakan konsep teknologi untuk mendukung era Revolusi Industri 4.0. Pengaplikasian IoT saat ini telah digunakan untuk berbagai bidang seperti layanan kesehatan, energi dan otomasi industri[1]. IoT memungkinkan koneksi antara sensor, kendaraan, rumah sakit dan industri melalui konektifitas internet. Konsep teknologi ini juga memungkinkan penerapan ke arah *Smart City*, *Smart Home*, *Smart Agriculture* dan *Smart Industry*[2].

Namun perkembangan teknologi pada IoT masih belum diimbangi dengan kesadaran akan permasalahan dari IoT itu sendiri. Sistem IoT masih menjadi salah satu *trend* terhadap serangan siber[3]. Serangan pada sistem IoT yang terjadi mayoritas dikarenakan karena penggunaan protokol yang tidak aman[4]. Serangan yang umum terjadi pada suatu sistem IoT adalah *Sniffing*[2]. Salah satu cara untuk mengatasi *Sniffing* adalah dengan menggunakan enkripsi. Data akan lebih terjaga terhadap serangan dengan adanya enkripsi[5]. Saat ini telah banyak modul IoT yang dilengkapi dengan *hardware accelerator* untuk enkripsi. *Hardware accelerator* merupakan dukungan perangkat keras untuk enkripsi yang dapat mengefisienkan penggunaan sumber daya pada perangkat[6]. Modul IoT yang dilengkapi dengan *hardware accelerator* dapat melakukan enkripsi dengan menggunakan API yang telah tersedia.

Namun tidak semua modul IoT memiliki *hardware accelerator* sehingga enkripsi diimplementasikan murni dalam bentuk program. Akan tetapi penerapan program enkripsi diketahui dapat menimbulkan permasalahan lain terutama pada modul IoT berbasis *embedded system* yang memiliki sumber daya terbatas. Tantangan penerapan program enkripsi pada *embedded system* yaitu karena adanya keterbatasan memori[7] dan kemampuan komputasi[8]. Sehingga diperlukan program enkripsi yang dapat diterapkan pada modul IoT berbasis *embedded system*.

*Advanced Enryption Standard* (AES) merupakan algoritma enkripsi yang sesuai dengan *embedded system*. Hal ini terbukti dengan implementasi dan modifikasi AES pada *embedded system* telah digunakan secara luas[9]. Selain itu algoritma enkripsi AES juga sangat baik digunakan pada sistem dengan sumber daya terbatas seperti pada *smart card* dan mikrokontroler 8-bit[10]. Dari uraian tersebut, pemilihan AES dapat menjadi solusi pengamanan yang sesuai terhadap modul IoT yang memiliki keterbatasan sumber daya.

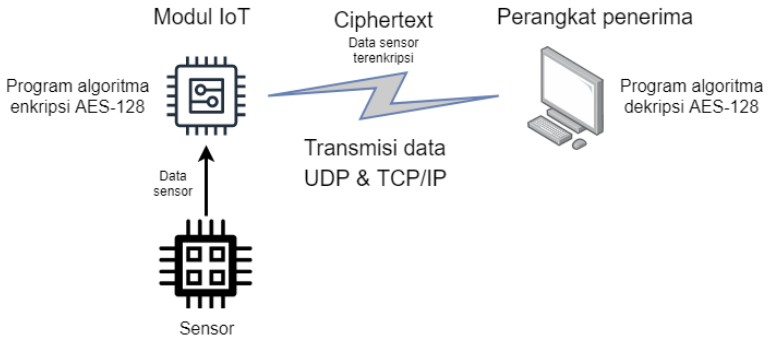
*Particle Photon* merupakan salah satu modul IoT berbasis mikrokontroler ARM Cortex M3 dengan jenis mikrokontroler STM32F205. Namun tidak seperti ESP32 dan Intel Joule yang memiliki *hardware accelerator*, *Particle Photon* tidak memiliki *hardware accelerator* untuk enkripsi. Sehingga pada kajian ini diterapkan program algoritma enkripsi pada modul IoT *Particle Photon* dan melakukan analisis terhadap beban yang ditimbulkan. Algoritma enkripsi yang digunakan adalah AES dengan panjang kunci 128 bit atau AES-128. Pemilihan AES-128 didasari pada panjang data IoT yang dienkripsi. Data IoT umumnya tidak memiliki ukuran data yang panjang, sehingga pemilihan AES-128 sudah cukup untuk mengamankan data IoT. Adapun tujuan yang ingin dicapai adalah untuk mengetahui beban dari penerapan program AES-128 pada modul IoT berbasis *embedded system*. Parameter yang ingin diketahui yaitu performa enkripsi yang dihasilkan dan memori yang digunakan oleh program algoritma AES-128 pada modul IoT.

## 2. Metode Penelitian

### 2.1 Gambaran Umum Sistem

Sistem yang dibuat terdiri dari dua bagian yaitu perangkat keras dan perangkat lunak yang saling terintegrasi. Modul IoT yang digunakan pada kajian ini adalah *Particle Photon* karena belum memiliki *hardware accelerator* untuk enkripsi. Pemilihan *Particle Photon* didasari pada jenis mikrokontroler yang digunakan yaitu ARM Cortex M3, di mana mikrokontroler ini pernah digunakan untuk kajian serupa yaitu penerapan algoritma enkripsi AES pada *embedded system* yang telah dilakukan pada kajian sebelumnya dengan judul *"Fast Implementation of AES on Cortex-M3 for Security Information Devices"* yang dilakukan oleh Wardhani, et al. [11] dengan menggunakan jenis *embedded system* LPC 1769. Dalam penelitian tersebut menggunakan mode CTR dengan *plaintext* yang telah ditentukan. Sedangkan pada kajian ini menggunakan mode ECB dengan *plaintext* yang berasal dari data sensor BME280. Pada bagian perangkat keras terdapat sensor BME280 yang dihubungkan dengan modul IoT *Particle Photon*. Pada bagian perangkat lunak terdapat program algoritma enkripsi AES-128 dan program untuk mengirim data yang ditanamkan pada modul IoT. Sedangkan pada perangkat penerima terdapat program untuk menerima data dan program algoritma dekripsi AES-128. Gambar 1 menunjukkan gambaran umum sistem.

ISSN (e): 2540-9123
ISSN (p): 2502-1982

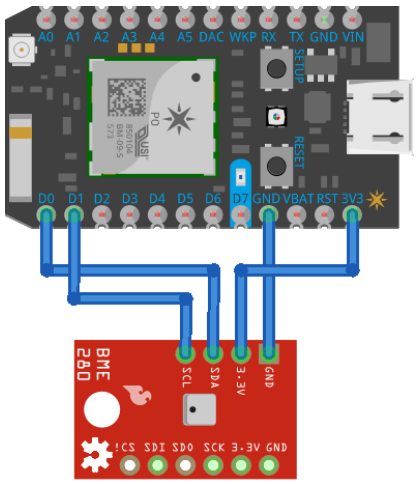



Gambar 1. Gambaran umum sistem.

Proses kerja sistem diawali dengan pengambilan nilai suhu dan kelembaban dari sensor BME280 oleh mikrokontroler yang digunakan sebagai data IoT sekaligus sebagai *plaintext*. Program algoritma enkripsi AES-128 yang ditanamkan pada modul IoT kemudian melakukan proses enkripsi yang menghasilkan suatu *ciphertext* atau data terenkripsi. *Ciphertext* kemudian dikirimkan lewat TCP/IP dengan *port* 5050 ke perangkat penerima.

## 2.2 Skematik Sensor

Skematik sensor pada Gambar 2 menunjukkan hubungan antara pin-pin sensor dengan pin pada modul IoT. Sensor BME280 dihubungkan dengan modul IoT *Particle Photon* menggunakan skema I2C (*Inter-Integrated Circuit*) atau menggunakan pin SDA dan SCL untuk mendapatkan nilai keluaran dari sensor secara langsung. Pin pada sensor dengan pin pada modul IoT saling dihubungkan dengan tujuan agar modul IoT dapat membaca nilai keluaran berupa tegangan.

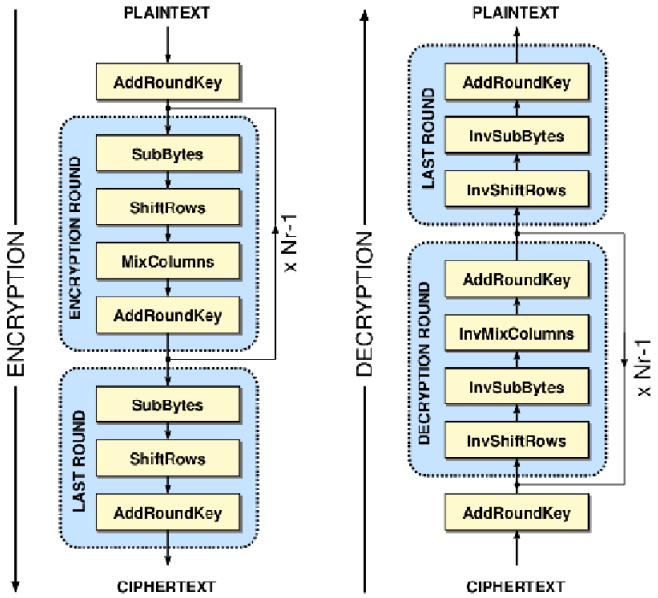



Gambar 2. Skematik sensor BME280 dengan Particle Photon.

## 2.3 Implementasi Algoritma Enkripsi dan Dekripsi AES-128

Proses dari algoritma enkripsi AES terdiri dari 4 fungsi transformasi yaitu; *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey*. Pada awal proses enkripsi, *plaintext* akan mengalami transformasi *Byte* pada fungsi *AddRoundKey*. Kemudian, *state* mengalami transformasi pada fungsi *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey* secara berulangkali (proses Nr). Proses Nr dalam algoritma enkripsi AES disebut juga sebagai *round function*. *Round function* mengalami transformasi yang berulang, namun pada round terakhir state tidak mengalami transformasi dengan fungsi *MixColumns*, proses perulangan ini dapat diformulasikan sebagai proses Nr-1. Proses enkripsi dan dekripsi AES ditunjukkan pada Gambar 3 [12].

ISSN (e): 2540-9123
ISSN (p): 2502-1982

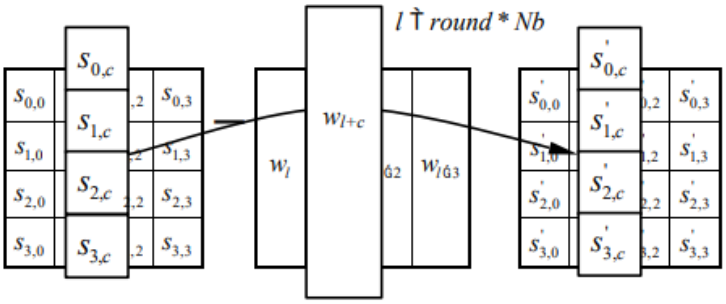



Gambar 3. Proses enkripsi dan dekripsi AES.

Pada Gambar 3 dapat diketahui bahwa terdapat empat fungsi utama dalam AES-128 sebagai fungsi transformasi pada tiap *round*. Empat fungsi dalam setiap *round* AES-128 adalah *AddRoundKey*, *SubBytes*, *ShiftRows*, dan *MixColoumn*.

### 2.3.1 *AddRoundKey*

Pada setiap putaran AES, 16 *Byte round key* berasal dari kunci master yang diinterpretasikan menjadi *array Byte* $4 \times 4$. Kemudian, array kunci di XOR-kan dengan array *state*. Dinotasikan sebagai $a_{i,j}$, sehingga menjadi baris $i$ dan kolom $j$ pada array *state*, sama halnya dengan array kunci yang dinotasikan sebagai $k_{i,j}$. *AddRoundKey* dapat dinyatakan sebagai transformasi dari $a_{i,j} = a_{i,j} \oplus k_{i,j}$ untuk setiap $1 \leq i \leq 4$ dan $1 \leq j \leq 4$ pada proses komputasi [13]. Proses tersebut diilustrasikan pada Gambar 4 [13].

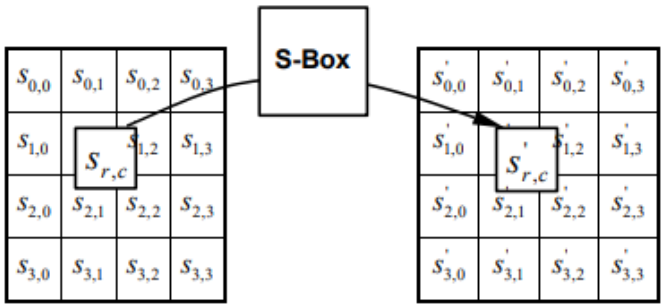



Gambar 4. Ilustrasi fungsi AddRoundKey.

### 2.3.2 *SubBytes*

Setiap *Byte* pada *array state* diubah dengan setiap *Byte* yang lain berdasarkan pada suatu tabel S yang telah ditetapkan. Tabel substitusi (tabel S) ini disebut sebagai *S-box*. Seperti ditunjukkan pada Gambar 5 [13], Tahap *SubBytes* dikomputasikan sebagai $a_{i,j} = S(a_{i,j})$ untuk setiap t1 $\leq i \leq 4$ dan $1 \leq j \leq 4$ [13]. Proses ini hanya menggunakan satu *S-box* untuk mensubstitusi semua *Byte* dari *array state*.

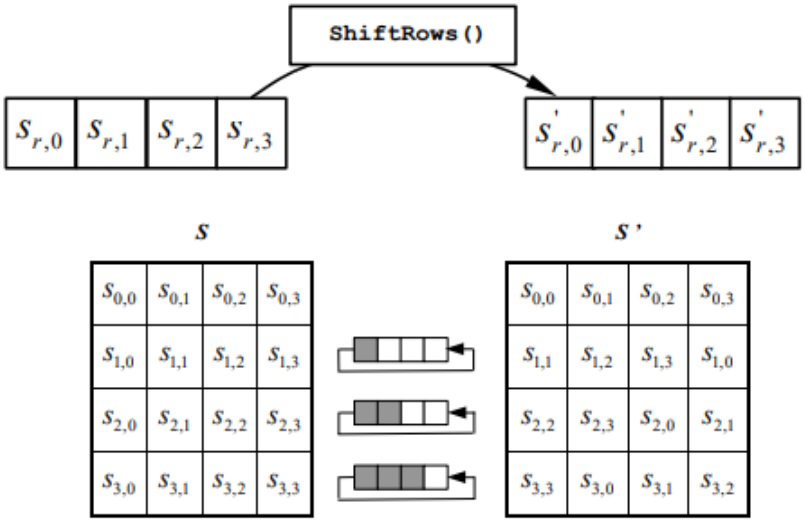
ISSN (e): 2540-9123
ISSN (p): 2502-1982




Gambar 5. Ilustrasi fungsi SubBytes.

### 2.3.3 *ShiftRows*

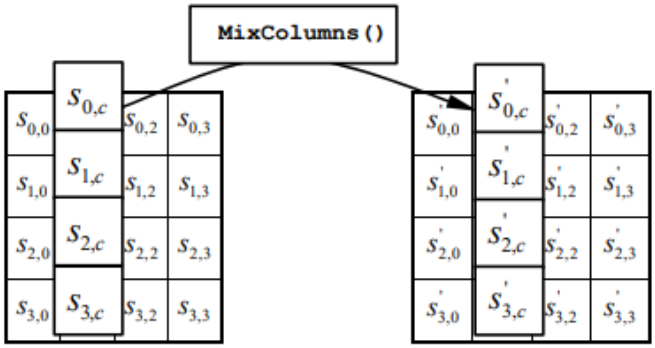
Proses *ShiftRows* ditunjukkan pada Gambar 6 [13]. Proses tranformasi ini mengeser baris dari suatu *state* matriks berdasarkan suatu pola (ke kiri atau ke kanan) dan nilai *offset*. *Byte* dari setiap baris dari *array state* digeser secara siklis ke kiri. Baris pertama dari *array* tidak digeser, baris ke-dua digeser satu tempat ke kiri, baris ke-tiga digeser dua tempat ke kiri, dan baris ke-empat digeser tiga tempat ke kiri. Sehingga *ShiftRows* dapat dinotasikan sebagai *Byte* $a_{2,1}$ menjadi $a_{2,4}$, *Byte* $a_{2,2}$ menjadi $a_{2,1}$ dan seterusnya [13].




Gambar 6. Ilustrasi fungsi ShiftRows.

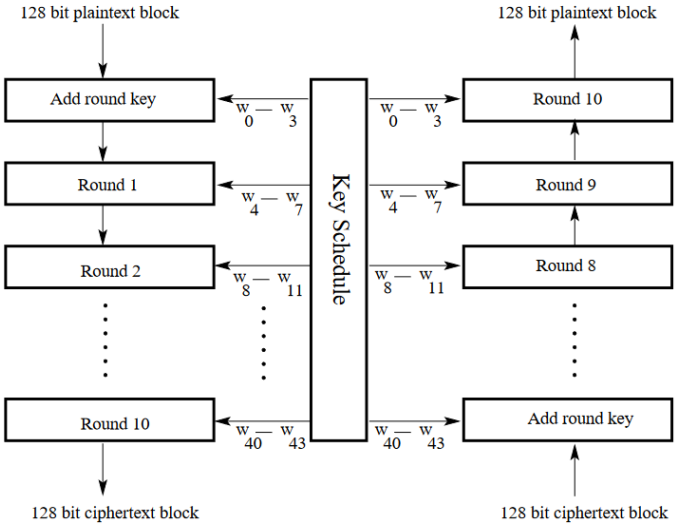
### 2.3.3 *MixColumns*

Pada tahap iyang diilustrasikan pada Gambar 7 [13], setiap kolom dicampur dari setiap transformasi linear yang telah terjadi pada *SubBytes* dan *ShiftRows*. Langkah ini bertujuan untuk menggantikan setiap *Byte* pada tiap kolom dengan fungsi dari semua *Byte* dalam kolom yang sama.




Gambar 7. Ilustrasi fungsi MixColumns[13].

Algoritma enkripsi AES-128, yang ditunjukkan pada Gambar 8 [13], memiliki sebelas putaran kunci atau *round key* yang terdiri dari satu *initial round key* sebagai pembangkit *round key* setelahnya, sembilan *main rounds* dan satu *final round*, dimana masing-masing *round key* membentuk suatu *key schedule*. Setelah *key schedule* terbentuk, maka akan terjadi proses putaran atau *round* atau proses Nr yang menyebabkan adanya transformasi. Terdapat sepuluh *round*, dengan kata lain akan terjadi sepuluh kali transformasi yang disebabkan oleh proses *SubBytes*, *ShiftRows* dan *MixColumns* yang kemudian di XOR-kan dengan *scheduled round key*. Namun hanya pada *round* ke-10 fungsi *MixColumns* tidak ikut melakukan proses transformasi [13].




Gambar 8. Rounds (putaran) pada AES-128.

Proses dekripsi AES pada dasarnya mirip dengan struktur proses enkirpsi, namun menggunakan fungsi invers yaitu; *InvSubBytes*, *InvShiftRows* dan *InvMixColumns* [14]. Algoritma AES-128 yang digunakan disini adalah *Pure* AES *software implementation*, dimana algoritma yang digunakan adalah algoritma *Rijndael* seperti blok diagram enkripsi AES yang telah ditunjukkan pada Gambar 3. Algoritma enkripsi ini menggunakan mode *Electronic Code Block* (ECB).

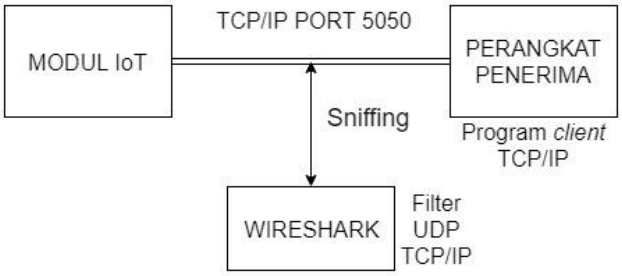
Program enkripsi/dekripsi ditulis dengan bahasa pemrograman C. Program ini ditulis menggunakan *DEV-C++* untuk mempermudah proses *debugging*, sedangkan untuk proses upload program ke mikrokontroler menggunakan *Particle-Dev IDE* yang berbasis *ATOM text editor* atau bisa juga menggunakan *Particle Build (Particle Web IDE)* yang berbasis aplikasi web dan *cloud*.

### 2.4 Perancangan Sistem Pengiriman Data

Data yang dikirim adalah *ciphertext* atau data sensor yang telah terenkripsi. Data dikirim melalui TCP/IP setelah proses enkripsi selesai. *Particle Photon* ditanami program untuk mengirim data dengan menggunakan pustaka bawaan yang telah tersedia pada *Particle Build IDE* agar dapat melakukan komunikasi dengan perangkat penerima. Perangkat penerima (PC) diprogram menggunakan pemrograman *socket* yang ditulis dalam bahasa pemrograman Python 3 dengan *Visual Studio Code*.

*Ciphertext* dikirimkan ke perangkat penerima melalui TCP/IP port 5050. Uji coba untuk melihat apakah informasi yang ditransmisikan merupakan *ciphertext* adalah dengan melakukan *sniffing* pada transmisi dengan menggunakan program *Wireshark*. Paket yang ditangkap kemudian disaring untuk mendapatkan paket yang hanya dilewatkan pada dan TCP/IP port 5050. Program untuk menerima data (program *client*) TCP/IP pada perangkat penerima juga harus

dijalankan agar *ciphertext* yang dikirimkan dapat diterima. Gambar 9 menunjukkan proses pengiriman dan *sniffing* data.




Gambar 9. Proses pengiriman dan *sniffing* data.

## 3. Hasil dan Pembahasan

### 3.1 Kesesuaian dan Kekuatan Hasil Enkripsi/Dekripsi

Pengujian menggunakan *plaintext* dari data sensor dengan format data *t x, h y*; dimana, *x = suhu, y = kelembaban* dan kunci enkripsi "kunci0123abcdef" atau *6B 75 6E 63 69 30 31 32 33 61 62 63 64 65 66 00* (dalam heksadesimal). Data *plaintext* dan kunci terlebih dulu diubah ke bentuk heksadesimal menggunakan fungsi *hex* yang terdapat di dalam program. Fungsi *Hex* merupakan suatu fungsi dari seluruh bagian program yang digunakan untuk melakukan konversi *string* dari *plaintext* menjadi bentuk heksadesimal ataupun sebaliknya. Kunci enkripsi menjadi bentuk bilangan heksadesimal yang disimpan dalam *array*. Fungsi *Hex* dibuat dengan tujuan agar program langsung mendapatkan masukan dalam bentuk heksadesimal. Tujuan dari adanya konversi ini adalah untuk menyesuaikan *input* dalam bentuk heksadesimal yang memang diperlukan oleh program enkripsi AES-128. Hasil konversi ini nantinya akan menghasilkan *array* dengan panjang maksimal 128 bit *array* atau 16 *Byte* karakter yang sesuai dengan panjang blok *plaintext* pada AES-128.

Fungsi *Hex* memiliki penambahan *padding* berupa bilangan heksadesimal 0x00 atau bisa disebut dengan *Zero Padding*. Penambahan *padding* akan terjadi jika jumlah *Byte input* < 16 *Byte* karakter karena blok *plaintext* dan kunci AES-128 memerlukan karakter masukan (termasuk spasi) sepanjang 16 *Byte* karakter. *Padding* disini menggunakan bilangan heksadesimal, maka 2 digit bilangan heksadesimal setara dengan nilai 1 *Byte array*. Tabel 1 menunjukkan hasil pengubahan *plaintext* ke dalam bentuk heksadesimal oleh fungsi *Hex*.

Tabel 1. Konversi *plaintext* ke heksadesimal.

| *Plaintext Asli* | Hasil konversi heksadesimal |
|---|---|
| t 27.2, h 57.69 | 742032372e322c20682035372e363900 |
| t 27.4, h 57.68 | 742032372e342c20682035372e363800 |
| t 27.4, h 57.65 | 742032372e342c20682035372e363500 |
| t 27.4, h 57.66 | 742032372e342c20682035372e363600 |
| t 27.4, h 57.66 | 742032372e342c20682035372e363600 |
| t 27.4, h 57.64 | 742032372e342c20682035372e363400 |
| t 27.4, h 57.52 | 742032372e342c20682035372e353200 |
| t 27.4, h 57.63 | 742032372e342c20682035372e363300 |
| t 27.4, h 57.62 | 742032372e342c20682035372e363200 |
| t 27.4, h 57.61 | 742032372e342c20682035372e363100 |

Pengujian ini bertujuan untuk mengetahui apakah informasi yang dikirimkan ke perangkat penerima melalui protokol TCP/IP merupakan *ciphertext*. Selain itu jug untuk mengetahui apakah program enkripsi telah berhasil menghasilkan *ciphertext* yang dapat dikembailkan ke bentuk *plaintext* sebagaimana mestinya. Hasil tersebut menjadi acuan apakah implementasi algoritma AES-128 dalam program sesuai dan berjalan sebagaimana mestinya atau tidak. Selain itu juga untuk mengetahui tingkat keacakan *ciphertext* yang dihasilkan dengan melakukan perhitungan nilai *Avalanche Effect*.

ISSN (e): 2540-9123
ISSN (p): 2502-1982

Tabel 2. Hasil pengujian enkripsi dan dekripsi.

| *Plaintext* (sebelum dienkripsi) | *Ciphertext* yang terbaca pada *Wireshark* | *Plaintext* (setelah didekripsi) |
|---|---|---|
| t 27.2, h 57.69 | 21 8e ec a4 6f 9e 9d 65 70 05 56 cc 2c a4 67 3c | t 27.2, h 57.69 |
| t 27.4, h 57.68 | 2d ec b6 f4 b4 fc 1b 3c d0 03 9d d5 90 88 a0 ad | t 27.4, h 57.68 |
| t 27.4, h 57.65 | d0 6f 11 ff a2 85 dd 33 9d 61 25 73 28 4f ad 2f | t 27.4, h 57.65 |
| t 27.4, h 57.66 | 2f 60 85 eb 54 8c 3f 6a 2f 7f 23 f1 30 65 b2 9d | t 27.4, h 57.66 |
| t 27.4, h 57.66 | 2f 60 85 eb 54 8c 3f 6a 2f 7f 23 f1 30 65 b2 9d | t 27.4, h 57.66 |
| t 27.4, h 57.64 | 75 a7 eb fd 14 62 86 a4 94 95 97 b0 2a a5 9c a4 | t 27.4, h 57.64 |
| t 27.4, h 57.52 | fd 00 62 4f b7 3b b6 0a de 7c b6 2e 4d d1 91 a4 | t 27.4, h 57.52 |
| t 27.4, h 57.63 | 90 c4 3d b3 5f c8 28 ac 6d 84 66 07 93 d4 34 e1 | t 27.4, h 57.63 |
| t 27.4, h 57.62 | 0c 6e 3e 25 ba 90 88 58 be ad 99 fa 88 07 cf 88 | t 27.4, h 57.62 |
| t 27.4, h 57.61 | 48 14 ba 4f 9b b4 72 3f ad 94 ee ed 56 c8 2e 9f | t 27.4, h 57.61 |

Tabel 2 menunjukkan bahwa program enkripsi dapat menghasilkan *ciphertext* yang juga dapat dikembalikan menjadi *plaintext* oleh program dekripsi, sehingga dapat dikatakan bahwa program ekripsi dan dekripsi dapat berjalan dengan baik. Plaintext yang sama menghasilkan menghasilkan ciphertext yang sama pula, sebagaimana yang ditunjukkan tabel 2 baris 4 dan baris 5. Hal ini terjadi karena penggunakan mode enkripsi ECB yang menghasilakan ciphertext sesuai dengan kunci yang telah didefinisikan.. Mode ECB paling sederhana jika dibandingkan dengan berbagai mode enkripsi lain. Berbeda dengan mode lainnya, mode ECB tidak menggunakan tambahan operasi untuk menghasilkan ciphertext pada setiap putarannya seperti halnya pada *Cipher Block Chaining* (CBC) yang menggunakan tambahan operasi XOR dalam notasi *Initialization Vector* (IV) pada setiap putarannya [14]. AES ECB bersifat deterministik jika digunakan untuk mengenkripsi satu blok *plaintext* dengan kunci yang sama. Mode ECB dapat dinotasikan sebagai berikut [15].

$$ECB\ Encryption : C_j = CHIP_k(P_j)\ for\ j = 1 \ldots n \quad (1)$$

Sedangkan untuk dekripsi dapat dinyatakan sebagai berikut.

$$ECB\ Encryption : P_j = CHIP^{-1}{}_k(C_j)\ for\ j = 1 \ldots n \quad (2)$$

Namun penggunaan mode ECB untuk enkripsi data IoT masih cukup aman. Adapun untuk mengetahui tingkat keamanan dari *ciphertext* yang dihasilkan dapat diketahui dengan melakukan perhitungan nilai *Avalanche Effect*. Perhitungan nilai *Avalanche Effect* dilakukan untuk mengetahui tingkat keamanan dari algoritma enkripsi AES-128 [16].

$$Avalanche\ Effect\ (\%) = \frac{Jumlah\ bit\ yang\ terbalik\ (bit)}{Jumlah\ bit\ ciphertext\ (bit)} \times 100 \quad (3)$$

*Tabel 3*. Hasil pengujian *Avalanche Effect*.

| | *Input* | *Ciphertext* | Nilai *Avalanche Effect* |
|---|---|---|---|
| *Plaintext 1* | 48 65 6C 6C 6F 20 77 6F<br>72 6C 64 **21** 00 00 00 00 | A9 3A 4D 66 B0 19 8F 3F<br>A8 BB 62 13 B0 ED 9E E5 | 53.10% |
| *Modified plaintext 1* | 48 65 6C 6C 6F 20 77 6F<br>72 6C 64 **23** 00 00 00 00 | A9 CB 3F 7C C3 27 14 EC<br>C4 41 F9 F2 A7 26 2C 57 | |
| *Plaintext 2* | 48 61 69 20 74 65 6d 61<br>6e 20 74 65 6d 61 **6e** 21 | F4 0C 6C 95 9A DE 1F 94<br>A8 15 5E 41 FA C3 BF D7 | 53,90% |
| *Modified plaintext 2* | 48 61 69 20 74 65 6d 61<br>6e 20 74 65 6d 61 **7e** 21 | 58 38 EC 04 85 25 F3 63 3F<br>59 27 A4 FB 0E C8 CA | |
| *Plaintext 3* | 30 31 32 33 34 35 36 37<br>38 39 61 62 63 64 65 **66** | 48 F1 D8 B7 A7 31 30 DC 66<br>05 98 68 76 63 8B B5 | 55.50% |

ISSN (e): 2540-9123
ISSN (p): 2502-1982

| | *Input* | *Ciphertext* | Nilai *Avalanche Effect* |
|---|---|---|---|
| *Modified plaintext 3* | 30 31 32 33 34 35 36 37<br>38 39 61 62 63 64 65 **67** | D8 CC 93 4B AA C7 CA 0D<br>5C DD 70 92 8F 75 9A 7A | |
| *Plaintext 4* | 74 20 32 35 2e 31 2c 20<br>68 20 35 36 2e 32 **35** 00 | C5 3F D8 B2 F2 CB 10 92 17<br>E9 A2 FF A0 64 A2 FA | 50,80% |
| *Modified plaintext 4* | 74 20 32 35 2e 31 2c 20<br>68 20 35 36 2e 32 **37** 00 | DE 34 C4 D2 03 1C 28 C7<br>0B 56 38 87 C3 97 CA 77 | |
| *Plaintext 5* | 74 20 32 35 2e 31 2c 20<br>68 20 35 38 2e **32** 35 00 | F0 A0 36 BC CE 00 41 04 7A<br>87 D4 DC 9F 56 FA BC | 54,70% |
| *Modified plaintext 5* | 74 20 32 35 2e 31 2c 20<br>68 20 35 38 2e **33** 35 00 | E0 58 58 03 B9 38 9E 86 25<br>F6 C3 24 14 0F 73 92 | |
| *Plaintext 6* | 48 69 20 77 65 6c 63 6f<br>6d 65 20 67 75 79 73 **21** | 8D 96 5A 16 EB 93 66 76 A8<br>AE F6 8E 95 22 A1 4E | 50% |
| *Modified plaintext 6* | 48 69 20 77 65 6c 63 6f<br>6d 65 20 67 75 79 73 **23** | 45 B9 9D 6F E6 A4 5B 35<br>BC B8 26 B3 DE CB B2 E5 | |

* Nilai tercetak tebal dan bergaris bawah menunjukkan adanya data yang berbeda

Tabel 3 menunjukkan enam kali hasil pengujian *Avalanche Effect* dengan cara mengganti satu bit dari *input*. Setiap bit yang diganti ditunjukkan oleh bilangan heksadesimal yang dicetak *bold* dan diberi garis bawah. Berdasarkan perhitungan *Avalanche Effect* pada Tabel 3 didapatkan hasil nilai *Avalanche Effect* lebih dari nilai tingkat keacakan yang telah ditetapkan yaitu 50% [16]. Jika nilai *Avalanche Effect* dari suatu algoritma enkripsi lebih besar dari 50% maka algoritma enkripsi tersebut memiliki kekuatan *ciphertext* yang baik [16]. Sehingga meski memiliki kekurangan pada *ciphertext* yang dihasilkan, namun dapat AES-128 masih aman dan memiliki tingkat keacakan yang tinggi.

## 3.2 Performa Enkripsi

Pengujian ini bertujuan untuk mengetahui bagaimana performa dari enkripsi AES-128 pada modul IoT *Particle Photon*. Performa dari suatu algoritma enkripsi dapat ditentukan dari nilai *throughput* yang didasarkan pada waktu enkripsi dan ukuran *plaintext*. Waktu eksekusi program enkripsi dapat dicari menggunakan fungsi waktu yang telah tersedia pada pemrograman *Particle Photon*. Fungsi waktu yang digunakan adalah *micros()* dimana fungsi ini memungkinkan mengambil waktu eksekusi program pada saat fungsi tersebut dipanggil. Satuan yang digunakan pada fungsi *micros()* adalah mikrodetik. Penentuan waktu enkripsi dapat dinyatakan dengan persamaan sebagai berikut [17].

$$Excecution\ Time = end\ time - start\ time \tag{4}$$

Berdasarkan perhitungan *encryption time* pada persamaan (4) maka performa enkripsi (*throughput*) dapat dicari dengan menggunakan persamaan (5) sebagai berikut [18].

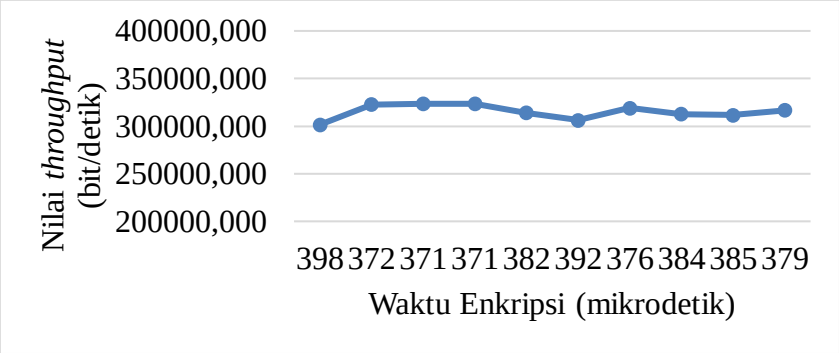
$$Throughput = \frac{plaintext\ size\ (bit)}{Excecution\ Time\ (s)} \tag{5}$$

Tabel 4. Waktu dan *throughput* enkripsi.

| Waktu enkripsi (*mikrodetik*) | Ukuran *plaintext* (bit) | *Throughput* (bit/detik) |
|---|---|---|
| 398 | 120 | 301507,538 |
| 372 | 120 | 322580,645 |
| 371 | 120 | 323450,135 |
| 371 | 120 | 323450,135 |
| 382 | 120 | 314136,126 |
| 392 | 120 | 306122,449 |
| 376 | 120 | 319148,936 |
| 384 | 120 | 312500,000 |
| 385 | 120 | 311688,312 |
| 379 | 120 | 316622,691 |

ISSN (e): 2540-9123
ISSN (p): 2502-1982

Tabel 4 menunjukkan dari sepuluh kali pengukuran throughput didapatkan nilai pada rentang 301507,538 bit/detik hingga 323450,135 bit/detik dengan rata-rata *throughput* sebesar 315120,697 bit/detik. Dapat diketahui pula bahwa semakin lama waktu enkripsi yang diperlukan maka nilai *throughput* akan semakin turun yang berarti waktu enkripsi berbanding terbalik dengan nilai throughput. Untuk lebih mudah dipahami disajikan pula grafik yang ditunjukkan pada Gambar 10.




Gambar 10. Grafik *throughput* enkripsi.

### 3.3 Memori Enkripsi

Pengujian ini bertujuan untuk mengetahui besar memori dan RAM yang digunakan oleh program enkripsi pada modul IoT *Particle Photon*.

Tabel 5. Hasil pengukuran penggunaan memori.

| Perbandingan Ukuran Memori | Memori (*Byte*) | RAM (*Byte*) |
|---|---|---|
| Tersedia | 125.000 | 60.000 |
| Digunakan | 16.024 | 3.020 |

Tabel 4 menunjukkan bahwa program algoritma enkripsi dapat diterapkan pada modul IoT dengan penggunaan memori flash oleh keseluruhan program sebesar 16.024 *Byte* yang berarti hanya menggunakan 12,819% dari total memori flash yang tersedia sebesar 125.000 *Byte*. Program algoritma enkripsi juga hanya menggunakan RAM sebesar 3020 *Byte* yang berarti masih menyisakan 5,033% dari RAM yang tersedia sebesar 60.000 *Byte*.

## 4. Kesimpulan

Implementasi enkripsi AES-128 pada modul IoT dapat berjalan dengan baik karena mampu memberikan hasil enkripsi *ciphertext* yang dapat dikembalikan menjadi pesan asli atau *plaintext*. Selain itu dari tiga *ciphertext* yang dihasilkan memiliki nilai *Avalanche Effect* di atas 50%. Waktu rata-rata yang dihasilkan relatif cepat untuk melakukan proses enkripsi. Penggunaan memori oleh program enkripsi relatif kecil untuk modul IoT yaitu memerlukan sekitar 16kB memori flash dan sekitar 3 kB RAM.

## Referensi

[1] Wei, Z., et al., "The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved," IEEE Paper Manuscript, 2018, pp. 1-11.

[2] Vashi, S. et al, "Internet of Things (IoT) - A Vision, Architectural Elements, and Security Issues," IEEE, 2017.

[3] Ukil, A., Sen, J., Koilakonda, S, "Embedded Security for Internet of Things,". IoT Security Laboratory, Kolkata. 2014.

[4] Carroll, E. et al., "*McAfee Labs 2019 Threats Predictions Report,"* [Online] Available: https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/mcafee-labs-

2019-threats-predictions/
[Diakses 10 Agustus 2019].

[5] Razzaq, M. A., Qureshi, M. A., Gill, S. H. & Ullah, S., "Security Issues in the Internet of Things (IoT): A Comprehensive Study" *(IJACSA) International Journal of Advanced Computer Science and Applications,* 8(6), pp. 383-388, 2017.

[6] Leveugle, R., Mkhinini, A. & Maistri, P., 2018. Hardware Support for Security in the Internet of Things: From Lightweight Countermeasures to Accelerated Homomorphic Encryption. *MDPI Journal Information*, 9(114), pp. 1-15.

[7] Noura, B., et al., "A compact 32-Bit AES design for embedded system", IEEE, 2012.

[8] Tsai, K., et al, "AES-128 Based Secure Low Power Communication for LoRaWAN IoT Environments," Special Section on Security and Trusted Computing for Indutrial Internet of Things, pp. 45325-45333.

[9] Raju, B., et al., "Implementation of an Efficient Dynamic AES", IEEE, 2017, pp. 39-43.

[10] Daemen, J. dan Rijmen, V. "The Design of Rijndael AES - The Advanced Encryption Standard", Springer, Berlin, 2001.

[11] Wardhani, R. W., Ogi, D., Syahral, M. & Catur, D. S., 2017. Fast Implementation of AES in Cortex-M3 for Security Information Devices, Bogor: IEEE.

[12] Arrag, S., Hamdoun, A., Tragha, A. & Khamlich, S., "Design and Implementation A different Architectures of mixcolumn in FPGA", *International Journal of VLSI Design & Communication Systems*, Vol. 3, 2012.

[13] Kak, A., "Lecture 8: AES: The Advanced Encryption Standard - Lecture Notes on "Computer and Network Security,*"* [Online], Available: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf [diakses 17 Mei 2019].

[14] Blazhevski, D., "Modes of Operation of The AES Algorithm. Faculty of Computer Science and Engineering", Skopje, Macedonia, pp. 212-216, 2013.

[15] Dworkin, M., "Recommendation for Block Cipher Modes of Operation", National Institute of Standards and Technology (NIST), 2001.

[16] Echeverri, C., "Visualization of the Avalanche" Faculty for Business Informatics & Business Mathematics, University of Mannheim, Mannheim, 2017.

[17] Moreno, C. & Fischmeister, S., 2017. Accurate Measurement of Small Execution Times – Getting Around Measurement Errors, Waterloo: University of Waterloo.

[18] Elminaam, D. S. A., Kader, H. M. A. & Hadhoud, M. M., 2010. "Evaluating The Performance of Symmetric Encryption Algorithms", *International Journal of Network Security*, 10(3), pp. 213-219.