PENERAPAN *ADVANCED ENCRYPTION STANDARD* (AES) 128 PADA MODUL *INTERNET OF THINGS* (IOT)

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI REKAYASA KOMPUTER

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

ROYYANNUUR KURNIAWAN ENDRAYANTO

NIM. 155060300111045

UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2019

LEMBAR PENGESAHAN

PENERAPAN *ADVANCED ENCRYPTION STANDARD* (AES) 128 PADA MODUL *INTERNET OF THINGS* (IOT)

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI REKAYASA KOMPUTER

Ditujukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

ROYYANNUUR KURNIAWAN ENDRAYANTO
NIM. 155060300111045

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing

Dosen Pembimbing I	Dosen Pembimbing II
Adharul Muttaqin, S.T., M.T.	Raden Arief Setyawan, S.T., M.T.
NIP. 19741203 200012 1 001	NIP. 19540424 198601 1 001

Mengetahui,
Ketua Jurusan Teknik Elektro

Ir. Hadi Suyono, S.T., M.T., Ph.D., IPM.
NIP. 19730520 200801 1 013

JUDUL SKRIPSI:

PENERAPAN *ADVANCED ENCRYPTION STANDARD* (AES) 128 PADA MODUL *INTERNET OF THINGS* (IOT)

Nama Mahasiswa: Royyannuur Kurniawan Endrayanto

NIM: 155060300111045

Program Studi: Teknik Elektro

Konsentrasi: Rekayasa Komputer

Dosen Pembimbing 1: Adharul Muttaqin, S.T., M.T. .....

Dosen Pembimbing 2: Raden Arief Setyawan, S.T., M.T. .....

Tim Dosen Penguji :

Dosen Penguji 1 : Rahmadwati, S.T., M.T., Ph.D .....

Dosen Penguji 2 : Dr. Fakhriy Hario Partiansyah, S.T., M.T. .....

Dosen Penguji 3 : Waru Djuriatno, S.T., M.T. .....

Tanggal Ujian : 25 Oktober 2019

SK Penguji: : 2215/UN10.F07/SK/2019

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya dan berdasarkan hasil penelusuran berbagai karya ilmiah, gagasan dan masalah ilmiah yang diteliti dan diulas di dalam Naskah Skripsi ini adalah asli dari pemikiran saya. 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 dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah Skripsi ini dapat dibuktikan terdapat unsur-unsur jiplakan, saya bersedia Skripsi dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, pasal 25 ayat 2 dan pasal 70).

Malang, 25 Oktober 2019

Mahasiswa,

ROYYANNUUR KURNIAWAN ENDRAYANTO
NIM. 155060300111045

"Sains tidak mengenal negara, karena pengetahuan adalah hak umat manusia, dan merupakan obor yang menerangi dunia." - Louis Pasteur

Teriring ucapan terima kasih kepada:
Ayah dan Bunda tercinta

RINGKASAN

Royyannuur Kurniawan Enrayanto, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Oktober 2019, Penerapan *Advanced Encryption Standard* (AES) 128 pada Modul *Internet of Things* (IoT), Dosen Pembimbing: Adharul Muttaqin dan Raden Arief Setyawan.

Tidak semua modul IoT dilengkapi dengan hardware accelerator untuk enkripsi, sehingga enkripsi diimplementasikan dalam program. Akan tetapi penerapan program enkripsi diketahui dapat menimbulkan permasalahan lain terutama jika diterapkan pada modul IoT berbasis *embedded system* yang memiliki sumber daya terbatas. Dalam kajian ini dibahas algoritma enkripsi AES-128 yang diimplementasikan dalam bentuk program dan diterapkan 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. Hasil pengujian juga menunjukkan besar energi yang digunakan oleh program enkripsi sebesar 663 mJ.

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

SUMMARY

Royyannuur Kurniawan Endrayanto, Department of Electrical Engineering, Faculty of Engineering University of Brawijaya, October 2019, *Implementation of Advanced Encryption Standard (AES) 128 on Internet of Things (IoT) Module*, *Academic Supervisor*: Adharul Muttaqin *and* Raden Arief Setyawan.

Not all IoT modules are equipped with hardware accelerators for encryption, so encryption is implemented in the program. However, the application of the encryption program is known to cause 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. The test results also indicate the amount of energy used by the encryption program is 663 mJ.

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

KATA PENGANTAR

Dengan Menyebut Nama Allah Yang Maha Pengasih Lagi Maha Penyayang. Alhamdulillah, puji syukur penulis panjatkan kehadirat Allah `Azza Wa Jalla atas rahmat dan hidayah-Nya telah memberikan kekuatan, kemampuan, ilmu pengetahuan dan wawasan sehingga penulis dapat menyelesaikan skripsi yang berjudul "Penerapan *Advanced Encryption Standard* (AES) 128 pada Modul *Internet of Things* (IoT)" dengan baik. Tak lepas shalawat serta salam selalu tercurahkan kepada junjungan besar Nabi Muhammad SAW yang telah menjadi suri tauladan dan pembawa rahmat bagi seluruh alam.

Penulis menyadari bahwa selama masa studi dan penyusunan skripsi ini tidak lepas dari bantuan, bimbingan, dorongan dan semangat dari berbagai pihak. Pada kesempatan ini penulis menyampaikan rasa terima kasih yang sebesar – besarnya kepada:

1. Allah `Azza Wa Jalla, karena memberikan karunia berupa ilmu dan kekuatan agar saya dapat mengambil manfaat dari setiap pelajaran sebagai jalan menuju kebijaksanaan.
2. Ayah dan Bunda tercinta, Ayah Mimit Primyastanto dan Bunda Enny Kuswandhari yang selalu memberikan do'a, dukungan, semangat, kasih sayang, serta perjuangan yang tiada akhir untuk dapat memberikan pendidikan terbaik kepada anak nomor dua mereka. Serta kedua saudaraku tercinta, Mas Mitsnein Luthfie Endryprimyas dan Dek Mitha Qurrota A'yun Lavany yang juga selalu memberikan semangat, dukungan, serta motivasi untuk menyelesaikan studi dan menjadi orang bermanfaat. Semoga saya dapat membuat kalian bangga dan mewujudkan impian keluarga bersama.
3. Umi Kusiati, Tante Inneke Kumalasanti dan dek Angga Indrafata yang selalu memberikan do'a dan dukungan agar penulis dapat menjadi pribadi yang lebih baik.
4. Bapak Ir. Hadi Suyono, S.T., M.T., Ph.D., IPM. selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya dan *Advisor of* IEEE Brawijaya University *Student Branch.*
5. Ibu Ir. Nurussa'adah, M.T. selaku Sekretaris Jurusan Teknik Elektro Universitas Brawijaya yang telah banyak memberikan nasehat dan pengarahan dalam mengembangkan prestasi akademik dan organisasi di Jurusan Teknik Elektro.
6. Ibu Rahmadwati, S.T.,M.T., Ph.D selaku Ketua Program Studi S1 Jurusan Teknik Elektro Universitas Brawijaya.
7. Bapak Adharul Muttaqin, S.T., M.T., selaku KKJF Rekayasa Komputer sekaligus Dosen Pembimbing I yang telah memberikan waktu, nasehat, motivasi, saran dan masukan dalam penyusunan skripsi. Terima kasih juga saya ucapkan karena telah banyak membantu dan melengkapi kekurangan saya dalam konferensi nasional dan PKM.
8. Bapak Raden Arief Setiawan, S.T., M.T. selaku Dosen Pembimbing II atas bimbingan, arahan, saran, dan kritik yang telah diberikan selama proses pengerjaan skripsi.
9. Bapak Angger Abdul Razak, S.T., M.T., M.Eng., Ph.D., yang telah mengikutkan saya dalam riset Laboratorium Informatika dan Komputer.
10. Bapak Akhmad Zainuri, S.T., M.T., yang telah mengikutkan saya dalam riset IoT, serta memberikan banyak ilmu dan pengalaman dalam membuat suatu inovasi.
11. Bapak Waru Djuriatno, S.T., M.T., selaku Ketua Laboratorium Sistem Digital yang telah banyak berbagi ilmu dan masukan dalam bidang *engineering*.
12. Bapak Ahmad Dulhadi, S.T., selaku Pranata Laboratorium Sistem Digital.
13. Segenap dosen dan staff administrasi Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya.
14. Adek Dinda Ersyah yang selalu mendo'akan, membantu, menyemangati dan menemani. Juga karena menjadi motivasiku untuk bersama-sama menjadi yang terbaik versi kami.

15. Pengurus inti Workshop Teknik Elektro 2018: Godam, Ghofur, Gristita, Ikrar, Mas Alaudin, Cita, Kukuh, Idam, Karil dan Bryan yang telah bersama-sama saling meringankan beban amanah selama satu periode kepengurusan.
16. Asisten Laboratorium Sistem Digital, khususnya Mas Rifki, Mas Hasan, Nugi, Luga, Toni, Samsul, Ester, Ina, Ipeh dan Nunil, serta para aslab lain yang tidak bisa saya sebutkan satu persatu.
17. Koordinator Paket E, Faris Aulia Ramadhan untuk berbagai bantuan selama masa studi.
18. Sesepuh Workshop Divisi IoT: Bagas, Indra, Wira, Anas, Fahri, Anabel, Lina. Pejabat kepengurusan tahun ini khususnya Bayu dan Candra, serta semua anggota IoT yang tidak bisa saya sebutkan satu persatu.
19. Sahabat seperjuangan KKN-P dan skripsi: Anjas Maulana, Lalu Arya Taruna Jaya dan M. Fikri Utomo yang mau menjadi tempat curhat, serta saling mendukung dan membantu sepanjang masa perjuangan.
20. Kawan-kawanku: Dewangga, Ichar, Rahman, Fitrah, Roby, Anas, Rama, Boby dan Rafi yang selalu menghibur, memotivasi dan memberi inspirasi.
21. Keluarga besar SERVO 2015 untuk dukungan, serta semangat yang diberikan.
22. Keluarga Kabinet KREASI RKIM 2018 untuk ilmu dan pengalaman dalam menebar manfaat.
23. Anggota tim lomba abadi ERKIM, Dino dan Candra untuk petualangan yang tak terlupakan.
24. Rekan riset dan jalan-jalan M. Mufti Fajar dan Ahmad Fathan Halim untuk pengalaman pertama lomba di luar negeri yang nanti bisa aku ceritakan kembali.
25. Semua pihak yang telah memberikan do'a, bantuan serta dukungan baik secara langsung maupun tidak langsung dalam penyusunan skripsi ini yang namanya tidak bisa saya sebutkan satu persatu.

Malang, Oktober 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

No Judul Halaman

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

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 (Wei, et al., 2018). 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* (Vashi, et al., 2017).

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 (Ukil, et al., 2014). Serangan pada sistem IoT yang terjadi mayoritas dikarenakan karena penggunaan protokol yang tidak aman (Carroll, et al., 2019). Serangan yang umum terjadi pada suatu sistem IoT adalah *sniffing* (Vashi, et al., 2017). Salah satu cara untuk mengatasi *sniffing* adalah dengan menggunakan enkripsi. Data akan lebih terjaga terhadap serangan dengan adanya enkripsi (Razzaq, et al., 2017). Saat ini telah banyak modul IoT yang dilengkapi dengan *hardware accelerator* untuk enkripsi *end-to-end* pada protokol tidak aman. *Hardware accelerator* merupakan dukungan perangkat keras untuk enkripsi yang dapat mengefisienkan penggunaan sumber daya pada perangkat (Leveugle, et al., 2018). 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, disipasi daya (Noura, et al., 2012) dan kemampuan komputasi (Tsai, et al., 2018). 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 (Raju, et al., 2017). Selain itu algoritma enkripsi AES juga sangat baik digunakan pada sistem dengan sumber daya terbatas seperti pada *smart*

card dan 8-bit mikrokontroler (Daemen & Rijmen, 2001). Dari uraian tersebut, pemilihan AES dapat menjadi solusi pengamanan yang sesuai terhadap modul IoT berbasis *embedded system* 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 skripsi ini diterapkan program algoritma enkripsi pada modul IoT *Particle Photon*. 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 performa dan beban dari penerapan program enkripsi AES-128 pada modul IoT. Parameter yang ingin diketahui yaitu performa enkripsi yang dihasilkan memori dan energi yang digunakan oleh program enkripsi AES-128 pada modul IoT.

1.2 Rumusan Masalah

Rumusan masalah dalam kajian ini adalah sebagai berikut.

a. Bagaimana penerapan program enkripsi AES-128 pada modul IoT?
b. Bagaimana performa program enkripsi AES-128 pada modul IoT?
c. Bagaimana beban (besar memori dan energi) yang digunakan oleh program enkripsi AES-128 pada modul IoT?
d. Bagaimana hasil pengiriman *cipherterxt* pada UDP dan TCP/IP?

1.3 Batasan Masalah

Hal-hal yang berkaitan dengan perancangan akan diberi batasan masalah agar pembahasan kajian lebih fokus pada poin-poin rumusan masalah. Adapun batasan masalah tersebut diuraikan sebagai berikut:

a. Modul IoT yang digunakan untuk penerapan program AES-128 adalah *Particle Photon*.
b. Penerapan hanya pada node sensor IoT.
c. Menggunakan algoritma enkripsi AES-128.
d. Parameter performa yang digunakan adalah kecepatan enkripsi dan dekripsi dalam satuan bit per *second* (bps) dan besar memori yang digunakan dalam satuan *Bytes*.
e. Hanya melakukan analisis dan pengukuran proses enkripsi pada modul IoT.
f. Hanya melakukan pengukuran tegangan *shunt* untuk menghitung energi yang digunakan oleh program enkripsi.

g. Koneksi yang digunakan bersumber dari *tethering* ponsel selular.

1.4 Tujuan

a. Mengetahui penerapan AES-128 pada modul IoT dengan implementasi program.
b. Mengetahui performa AES-128 yang diterapkan pada modul IoT.
c. Mengetahui beban sumber daya (memori dan energi) yang digunakan karena adanya penerapan AES-128.
d. Mengetahui keberhasilan pengiriman *ciphertext* lewat protokol UDP dan TCP/IP.

1.5 Manfaat

Manfaat maupun luaran yang diharapkan pada kajian ini diuraikan sebagai berikut:

a. Bagi pengembang IoT

   Penerapan enkripsi pada modul IoT dapat mencegah serangan seperti *sniffing* yang memungkinkan penyerang mengumpulkan data yang dilewatkan pada internet. Dengan diterapkannya enkripsi pada modul IoT diharapkan sistem IoT yang dibuat oleh pengembang dapat lebih aman.

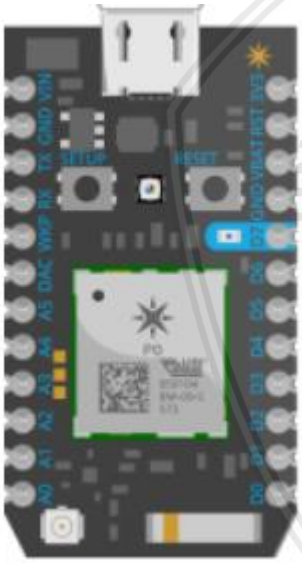
b. Bagi pengguna IoT

   Salah satu konsen terhadap data pengguna IoT adalah privasi. Adanya penerapan enkripsi untuk mengamankan data akan membuat transaksi data antara perangkat pengirim dan penerima menjadi lebih aman dan terpercaya. Sehingga data pengguna yang bersifat sensitif dapat lebih rahasia.

BAB II
TINJAUAN PUSTAKA

2.1 *Particle Photon*

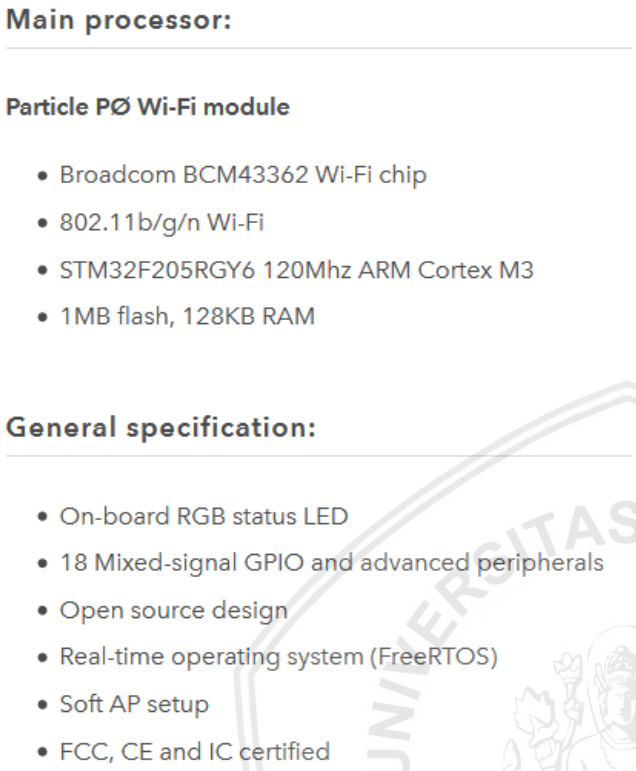
Particle Photon adalah salah satu modul IoT yang diproduksi oleh Particle Industries Inc., dengan modul *open-source* berukuran 37x20 mm yang terintegrasi dengan mikrokontroler ARM dan *chip* Wi-Fi (Hart, 2017)**.** *Particle Photon* merupakan modul IoT berbasis ARM Cortex M3 dengan kecepatan *clock* 120 MHz. *Particle Photon* memiliki 18 periferal GPIO yang terdiri dari 8 pin digital dan 6 pin analog. *Photon* merupakan perangkat IoT yang mutakhir dengan kemampuan untuk memungkinkan pengontrolan jarak jauh atau pengumpulan data dari beberapa sensor yang telah terhubung (Karlsson, 2017).



Gambar 2.1 Bentuk fisik modul IoT *Particle Photon*
Sumber: (Karlsson, 2017)

Particle Photon memiliki memori *flash* sebesar 1 MB dan RAM sebesar 128 KB. Namun memori yang bisa digunakan oleh pengguna sebesar 125.000 *Byte* memori *flash* dan 60.000 *Byte* RAM. 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. (2017). Penerapan enkripsi pada *Particle Photon* juga didasari karena modul IoT ini belum memiliki

hardware accelerator untuk enkripsi. Berikut merupakan spesifikasi umum dari *Particle Photon*:



Gambar 2.2 Spesifikasi umum Particle Photon
Sumber: (Hart, 2017)

2.2 Sensor Suhu dan Kelembaban BME280

BME280 merupakan sensor suhu, kelembapan dan tekanan udara yang diproduksi oleh Bosch GmbH. BME280 adalah sensor generasi terbaru dari Bosch, sensor ini memiliki spesifikasi yang sama dengan sensor keluaran sebelumnya, tetapi dapat menggunakan I2C atau SPI (Ada, 2017). BME280 adalah sensor lingkungan terintegrasi yang dikembangkan khusus untuk aplikasi seluler di mana ukuran dan konsumsi daya yang rendah merupakan tujuan dari desain utama. Unit ini menggabungkan linearitas dan akurasi tinggi untuk mengukur tekanan, kelembaban dan suhu dalam 8-pin logam-tutup berukuran 2,5 x 2,5 x 0,93 mm^3 paket LGA. BME280 juga dirancang untuk konsumsi arus rendah (3,6 µA @ 1Hz), stabilitas jangka panjang dan daya tahan EMC yang tinggi. Tabel 2.1 menunjukkan spesifikasi umum dari sensor BME280.

Gambar 2.3 Sensor BME280
Sumber: (Bosch, 2018)

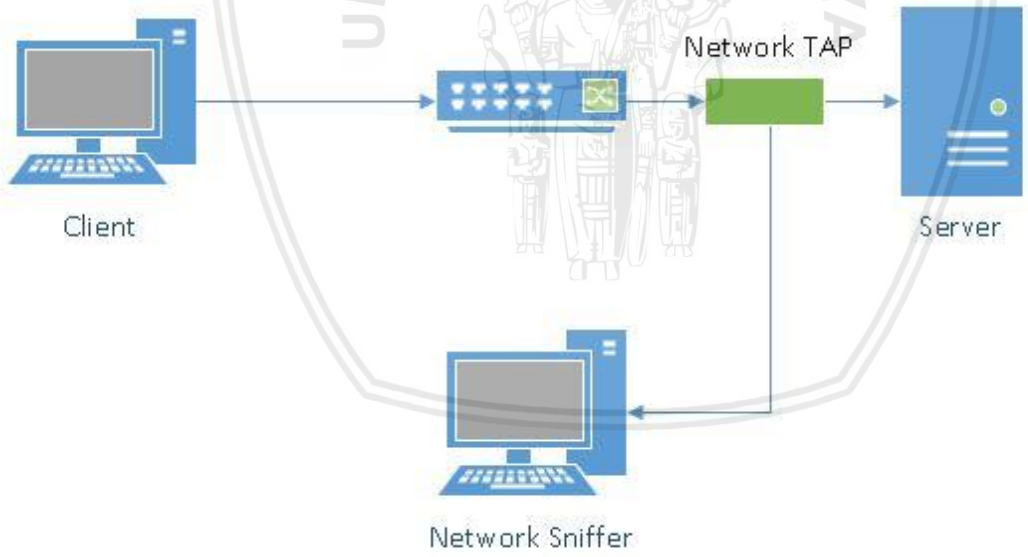
Tabel 2.1
Spesifikasi parameter elektrik sensor BME280

Parameter	Simbol	Kondisi	Min	Typ	Max	Satuan
Tegangan masukan	V_{DD}	Ripple maks 50 mV_{pp}	1.71	1.8	3.6	V
Arus sleep	V_{DDIO}			0.1	0.3	
Arus *standby* (periode tidak aktif pada mode normal)	I_{DDSL}			0.2	0.5	µA
Arus ketika pengukuran kelembaban	I_{DDSB}	Nilai maks pada 85 °C		340		µA
Arus ketika pengukuran suhu	I_{DDP}	Nilai maks pada 85 °C		350		µA
Akurasi waktu *standby*	$\Delta t_{standby}$			± 5	± 25	%

2.4 *Sniffing*

Sniffing adalah salah satu teknik serangan yang bertujuan untuk megumpulkan informasi di dalam suatu jaringan. *Packet sniffing* didefinisikan sebagai teknik yang digunakan untuk memonitor setiap paket yang melintasi jaringan (Rupam, et al., 2013).

Packet sniffing merupakan bagian dari perangkat lunak atau perangkat keras yang memonitor semua lalu lintas jaringan. Sedangkan *network sniffing* adalah serangan pada lapisan jaringan yang terdiri dari pengambilan paket data yang ditransmisikan oleh komputer lain dan membaca konten data untuk mencari informasi sensitif seperti kata sandi, token sesi dan informasi rahasia (Oluwabukola, et al., 2013). Untuk dapat membaca dan menganalisis setiap protokol yang melintasi jaringan, diperlukan suatu perangkat lunak yang disebut *Sniffer*. *Packet sniffer* atau *sniffer* adalah alat dasar untuk mengamati pertukaran paket jaringan di komputer. Seperti namanya, *sniffer* mengendus ("*sniffs*") paket yang dikirim/diterima dari/oleh komputer. *Sniffer* akan menyimpan dan/atau menampilkan konten dari berbagai protokol dalam suatu jaringan dari paket yang ditangkap (Zhang, 2016). Pada dasarnya, *sniffer* bekerja dengan menganalisis lalu lintas jaringan tanpa mengarahkan atau mengganggu aliran data. Saat data mengalir ke *sniffer*, ia dapat menangkap setiap paket dan mendekode data paket mentah untuk mengungkapkan berbagai hal dalam paket (Allen, 2017).



Gambar 2.4 Skema *sniffing* pada suatu jaringan
Sumber: (Allen, 2017)

2.5 *Advanced Encryption Standard* (AES)

Pada Januari 1997, Institut Nasional Standar dan Teknologi (NIST) mengumumkan awal inisiatif untuk mengembangkan standar enkripsi baru yaitu AES untuk menjadi Standar Pemrosesan Informasi Federal (FIPS). *Advanced Encryption Standard* (AES) merupakan salah satu algoritma enkripsi modern pengganti dari *Data Encryption Standard* (DES) dan

triple-DES. Tidak seperti proses seleksi untuk DES, *Secure Hash Algorithm* (SHA-1) dan *Digital Signature Algorithm* (DSA), NIST telah mengumumkan bahwa proses seleksi AES akan terbuka. Algoritma AES ditemukan pada September 1997 oleh Daemen dan Rijmen yang merupakan dua orang peneliti dari Belgia.

Implementasi AES memiliki karakteristik khusus, salah satunya adalah *versatility* yang artinya AES dapat diimplementasikan pada beberapa perangkat berbeda. Kemampuan AES yang menjadi syarat standar keamanan adalah dapat diimplementasikan pada 8-bit mikrokontroler dan *smart cards*, yang memiliki keterbatasan penyimpanan untuk program dan besar RAM yang terbatas pula (Daemen & Rijmen, 2001). Model Rijndael asli dan model AES memiliki perbedaan mendasar yaitu panjang blok dan panjang kunci yang dapat diterapkan. Model Rijndael memiliki panjang blok dan panjang kunci berupa variabel yang secara spesifik merupakan hasil perkalian dari 32 bit, dengan nilai minimum 128 bit dan maksimum 256 bit.

Sedangkan model AES memiliki panjang blok tetap yaitu sebesar 128 bit, dan hanya mendukung panjang kunci 128, 192 dan 256 bit (Daemen & Rijmen, 2001). Panjang kunci pada AES juga menentukan jumlah *round* atau operasi putaran untuk menghasilkan *ciphertext* atau pesan terenkripsi dengan beberapa *round* (Boneh & Shoup, 2015). AES memerlukan 10, 12, atau 14 kali putaran enkripsi yang disesuaikan dengan panjang kunci yang digunakan (Dobbertin, et al., 2005). Tabel 2.2 menunjukkan perbandingan AES dari panjang kunci yang didukung.

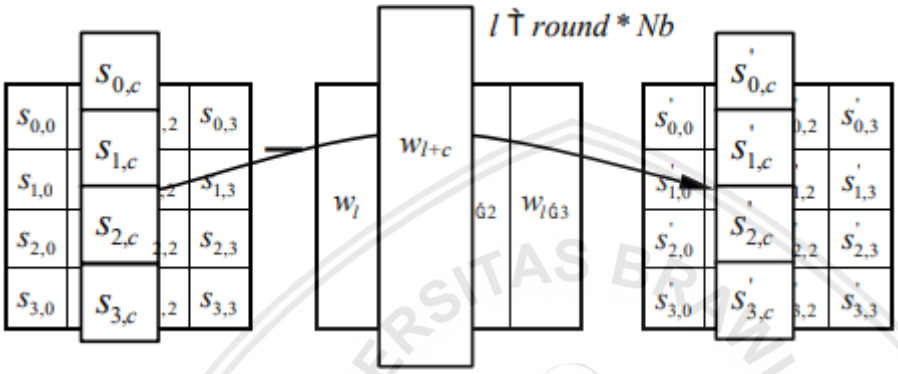
Tabel 2.2
Perbandingan AES berdasarkan panjang kunci
Sumber: (Boneh & Shoup, 2015)

Chipper name	*Key-size (bits)*	*Number of rounds*
AES-128	128	10
AES-192	192	12
AES-256	256	14

2.5.1 AES-128

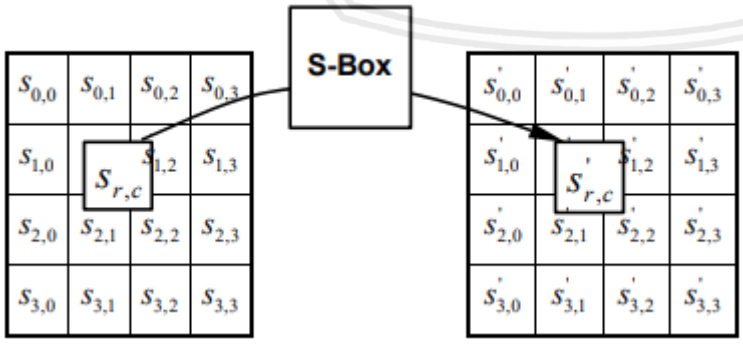
Algoritma enkripsi AES-128 memiliki *Byte array* 4×4 yang dinamakan *array state* dan memiliki operasi putaran atau *round* yang berulang sebanyak sepuluh kali. Pada semua *array state* terjadi operasi substitusi dan permutasi untuk memungkinkan adanya pengacakan dan difusi. Terdapat empat fungsi utama dalam AES-128 sebagai fungsi transformasi pada tiap *round*. Empat fungsi dalam setiap *round* AES-128 antara lain:

a. ***AddRoundKey***: Pada setiap putaran AES, 16 *Byte round key* berasal dari kunci master yang diinterpretasikan menjadi *array Byte* 4×4. Kemudian, array kunci secara sederhana 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 $k_{i,j}$. Sehingga tahap *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.



Gambar 2.5 Ilustrasi fungsi AddRoundKey
Sumber: (NIST, 2001)

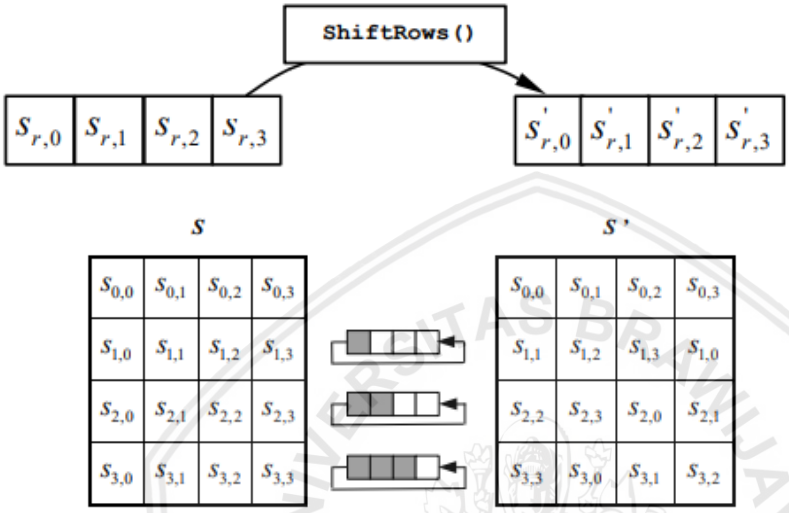
b. ***SubBytes***: Setiap *Byte* pada *array state* diubah dengan setiap *Byte* yang lain berdasarkan pada suatu tabel S yang telah ditetapkan. Tabel substitusi (tabel S) ini disebut sebagai *S-box*. Tahap *SubBytes* dikomputasikan sebagai $a_{i,j} = S(a_{i,j})$ untuk setiap $1 \leq i \leq 4$ dan $1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



Gambar 2.6 Ilustrasi fungsi SubBytes
Sumber: (NIST, 2001)

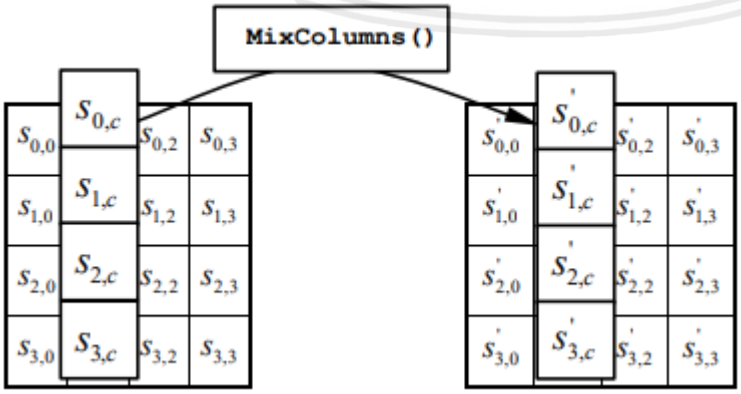
c. ***ShiftRows***: Proses tranformasi shiftrow merupakan suatu proses transposisi *Byte* yang dapat mengeser baris dari suatu state matriks berdasarkan suatu pola (ke kiri atau ke

kanan) dan nilai offset yang bervariasi. *Byte* dari setiap baris dari *array state* akan 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.



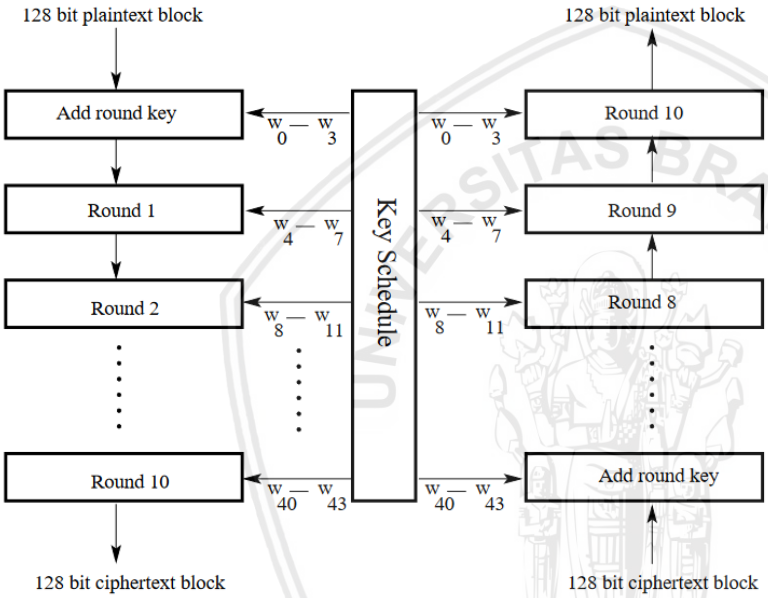
Gambar 2.7 Ilustrasi fungsi ShiftRows
Sumber: (NIST, 2001)

d. ***MixColumns***: Pada tahap ini, 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 2.8 Ilustrasi fungsi *MixColumns*
Sumber: (NIST, 2001)

AES-128 memiliki sebelas putaran kunci atau *round key* yang terdiri dari satu *initial round key* sebagai pembangkit *round key* setelahnya yaitu sembilan *main rounds* dan satu *final round*, dimana masing-masing *round key* akan 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 diikutkan untuk melakukan transformasi.



Gambar 2.9 Rounds (putaran) pada AES-128
Sumber: (Kak, 2019)

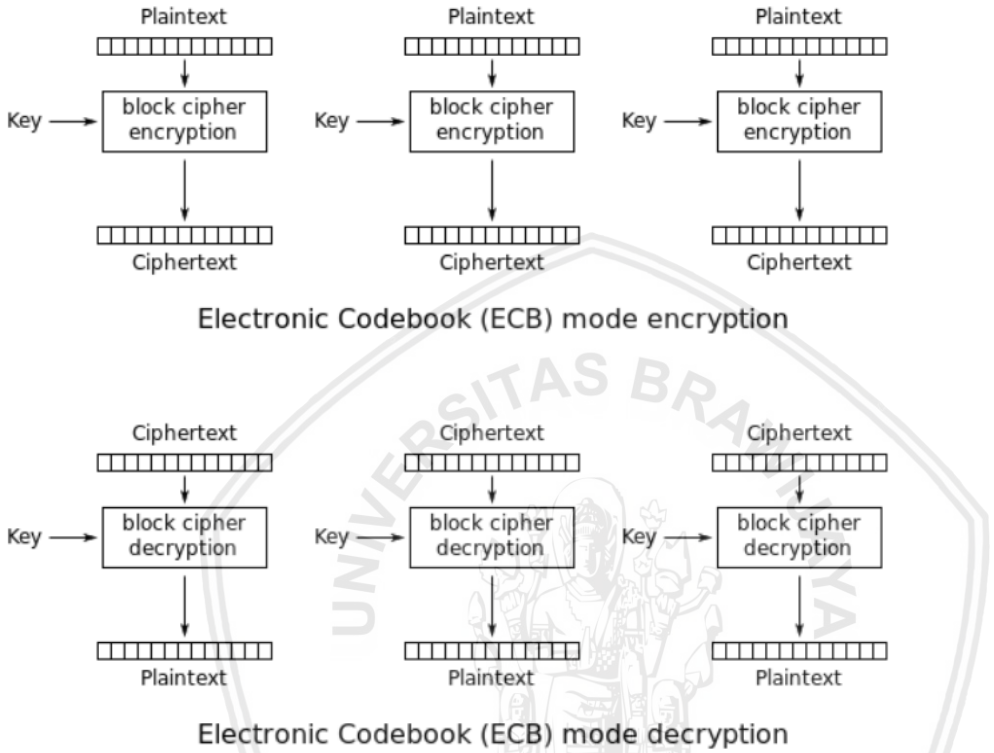
2.5.2 AES Mode ECB

AES *Electronic Code Block* (ECB) adalah implementasi murni dari algoritma enkripsi AES pada suatu bahasa pemrograman. ECB merupakan mode paling sederhana dibandingkan dengan berbagai mode lain yang dapat diterapkan pada algoritma enkripsi. Berbeda dengan mode lainnya, AES 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 (Blazhevski, et al., 2013). AES ECB bersifat deterministik, karena blok *plaintext* identik akan menghasilkan blok *ciphretext* yang identik pula. Berdasarkan penjelasan tersebut, mode ECB dapat dinotasikan pada Persamaan (2.1) dan Persamaan (2.2) (Dworkin, 2001).

ECB Encryption : $C_j=CHIP_k(P_j)$ for j=1… n .....(2.1)

Sedangkan untuk dekripsi dapat dinyatakan sebagai berikut.

ECB Encryption : $P_j=CHIP^{-1}{}_k(C_j)$ for j=1… n .....(2.2)

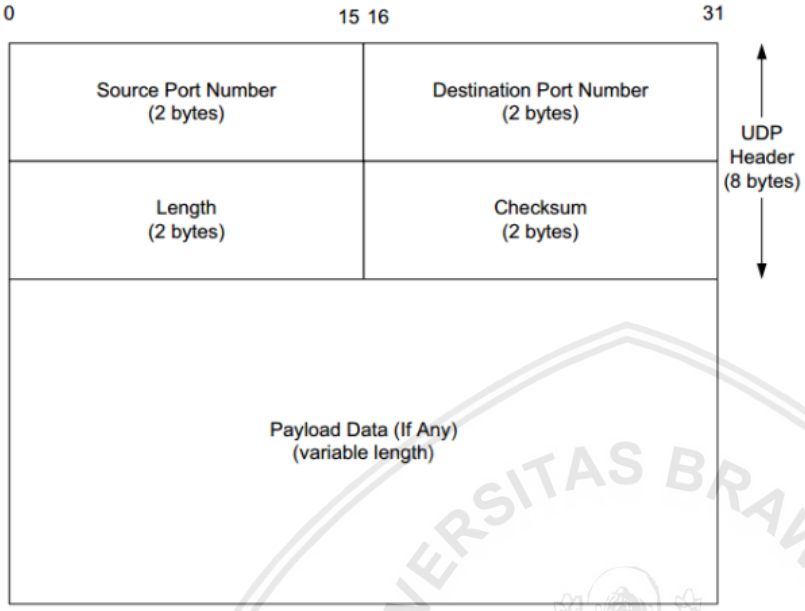


Gambar 2.10 Skema operasi AES mode ECB
Sumber: (Miller, 2018)

2.6 UDP

UDP merupakan lapisan transport yang digunakan pada kebanyakan perangkat IoT. UDP dipilih untuk komunikasi data pada perangkat IoT karena UDP memiliki performa lebih daripada TCP/IP. UDP lebih cepat dibandingkan TCP/IP karena tidak ada bentuk kontrol terhadap aliran data dan koreksi error (Rodriguez, 2014). Berbeda halnya dengan TCP, UDP tidak memberikan respon balik kepada pengirim bahwa paket telah sampai ke penerima. UDP tidak dapat melakukan *pipelining* seperti halnya TCP yang dapat memberikan respon balik antara pengirim dan penerima (Mesander, 2014). UDP lebih rentan terhadap serangan, sehingga memerlukan skema pengamanan seperti enkripsi dan autentikasi. Namun, UDP merupakan layer transport dengan kemampuan melakukan *multicasting* seperti yang dibutuhkan untuk jenis transaksi data pada IoT, dimana

kemampuan *multicasting* sudah tertanam pada software UDP namun tidak pada TCP/IP (Hadiyuwono & Pambudi, 2011).



Gambar 2.11 Struktur UDP
Sumber: (Shichao, 2015)

Dengan:

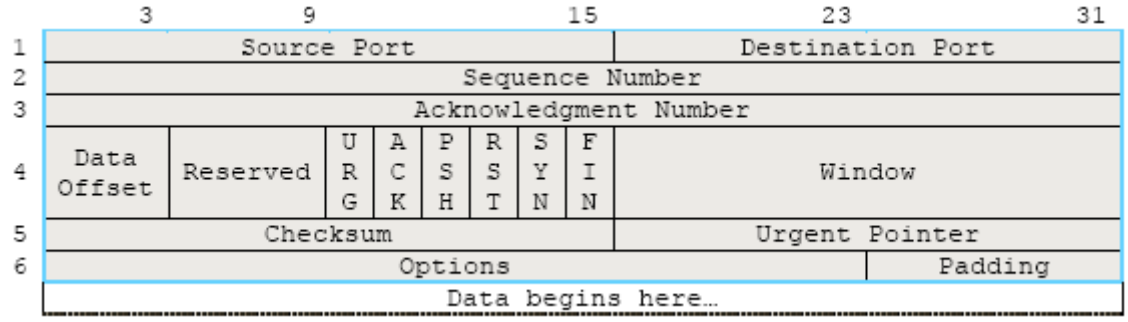
Source port = Untuk mengidentifikasi port pengirim dan diasumsikan sebagai port untuk balasan jika diperlukan.

Destination port = Untuk mengidentifikasi port penerima dan selalu diperlukan.

Length = Untuk menentukan panjang dalam Byte header UDP dan data UDP.

2.7 TCP/IP

TCP/IP berada pada lapisan transport sebagaimana protokol komunikasi lainnya seperti UDP (Sukaridhoto, 2014). Namun, berbeda dengan UDP, TCP/IP memiliki respon umpan balik kepada pengirim apakah data yang dikirim telah diterima oleh penerima. Protokol ini dapat digunakan sebagai *error detection*, dimana jika terdapat kesalahan pada paket data yang diterima, maka penerima tidak akan menerima data yang dikirim. Pengirim akan mengirim ulang paket data yang mengandung kesalahan tadi. Namun kemampuan *error detection* tersebut dapat menimbulkan delay yang cukup berarti (Lydia, et al., 2006). Untuk mengetahui performa dari TCP/IP dapat dilihat dari besar nilai *delay*. *Delay* merupakan jumlah jeda waktu pengiriman data antara perangkat dengan perangkat lain atau waktu transit pada komunikasi data (Kwon, 2012).



Gambar 2.12 Struktur TCP/IP
Sumber: (Tenouk, 2019)

Dengan:

Source port = Untuk mengidentifikasi sumber protokol pada lapisan aplikasi yang mengirim segmen TCP.

Destination port = Mengidentifikasi tujuan protokol pada lapisan aplikasi yang menerima segmen TCP.

Sequence number = Mengindikasikan nomor urut dari oktet pertama data dalam suatu segmen TCP yang akan dikirim.

Acknowledge Number = Mengindikasikan nomor urut dari oktet selanjutnya dalam urutan Byte yang diharapkan untuk diterima oleh pengirim dari penerima pada pengiriman selanjutnya.

Data offset = Mengindikasikan letak data dalam segmen TCP dimulai. Field ini juga dapat berarti ukuran dari header TCP.

Reserved = Dialokasikan untuk digunakan pada waktu berikutnya. Pengirim segmen TCP akan mengeset bit-bit ini ke dalam nilai 0.

Flags = Menandakan flag-flag TCP yang terdiri atas: URG (Urgent), ACK (Acknowledgment), PSH (Push), RST (Reset), SYN (Synchronize), dan FIN (Finish).

Window = Menandakan jumlah Byte yang tersedia yang dimiliki oleh buffer host penerima segmen yang bersangkutan.

Checksum = Untuk melakukan pengecekan integritas segmen TCP (*header* dan *payload* TCP).

Urgent Pointer = Menandakan lokasi data yang dianggap "*urgent*" pada segmen.

Options = Berfungsi sebagai penampung beberapa opsi tambahan TCP.

2.8 *Avalanche Effect*

Avalanche Effect merupakan suatu metode yang digunakan untuk menguji kekuatan suatu algoritma terutama algoritma enkripsi *cipher block* dan kriptografi fungsi *hash* berdasarkan tingkat keacakan *ciphertext* yang dihasilkan. Penilaian dari suatu algoritma enkripsi yang aman dapat diketahui dengan nilai pengujian *Avalanche Effect* yang tinggi (Echeverri, 2017). Desain algoritma yang baik mengacu pada hasil pengujian *Avalanche Effect* dimana jika terjadi perubahan sedikit (misal satu bit) dari *input* (*plaintext*) akan menghasilkan *output* (*ciphertext*) yang berubah secara signifikan (Chakraborty, et al., 2011). Begitupula sebaliknya, jika hasil pengujian *Avalanche Effect* tidak menunjukkan tingkat keacakan yang tinggi, maka desain algoritma tersebut dapat dikatakan tidak baik. Desain algoritma yang tidak baik akan menghasilkan *ciphertext* yang tidak terlalu acak jika terjadi perubahan *input* meskipun hanya sebesar satu bit (Witoolkollachit, 2015).

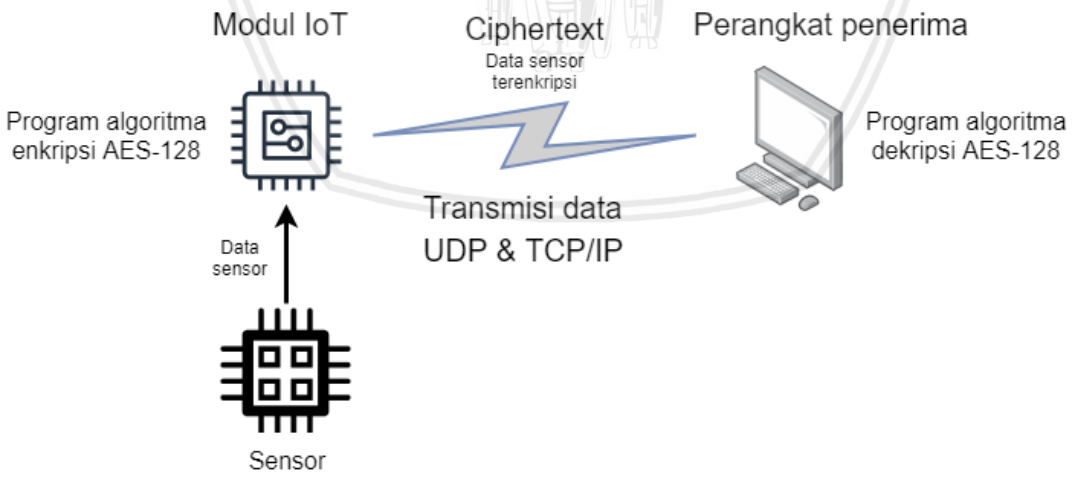
Suatu algoritma enkripsi dikatakan baik apabila memiliki nilai *Avalanche Effect* sesuai dengan parameter yang ditetapkan yaitu mendekati atau lebih dari 50% (Echeverri, 2017). Kekuatan dari suatu algoritma enkripsi merupakan salah satu hal yang penting karena tujuan diterapkannya enkripsi adalah untuk mengamankan suatu informasi agar tidak mudah diserang sehingga informasi asli dapat diketahui orang lain. Salah satu tujuan dari pengujian *Avalanche Effect* adalah untuk mengukur seberapa besar perubahan informasi meski sekecil apapun perubahannya, sehingga jika terjadi serangan maka akan sulit untuk penyerang melakukan analisis agar mendapatkan informasi aslinya (Kumar & Tiwari, 2012).

BAB III
METODE KAJIAN

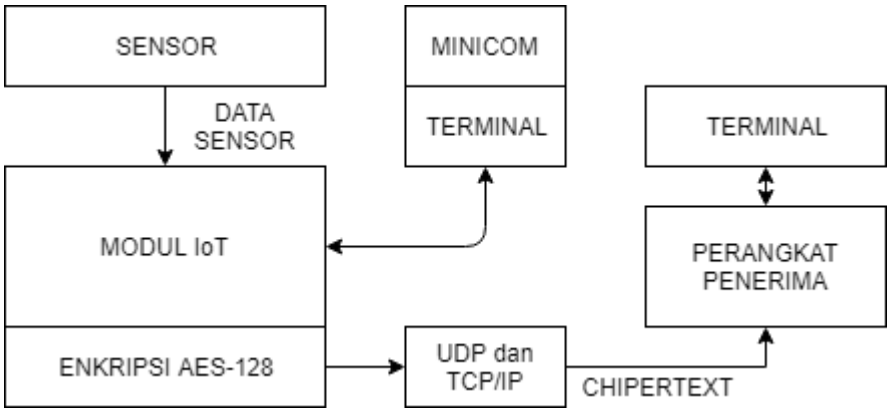
3.1 Gambaran Umum Sistem

Sistem yang dibuat terdiri dari dua bagian yaitu perangkat keras dan perangkat lunak yang saling terintegrasi. Pada bagian perangkat keras terdapat sensor BME280 yang dihubungkan dengan modul IoT *Particle Photon*. Pada bagian perangkat lunak terdapat program enkripsi AES-128 dan program untuk mengirim data yang ditanamkan pada modul IoT. Sedangkan pada perangkat penerima (*Personal Computer*) terdapat program untuk menerima data dan program dekripsi AES-128. Untuk transmisi data dilakukan dengan menggunakan koneksi jaringan lokal yang berasal dari *tethering* ponsel selular. Adapun gambaran umum sistem ditunjukkan pada Gambar 3.1.

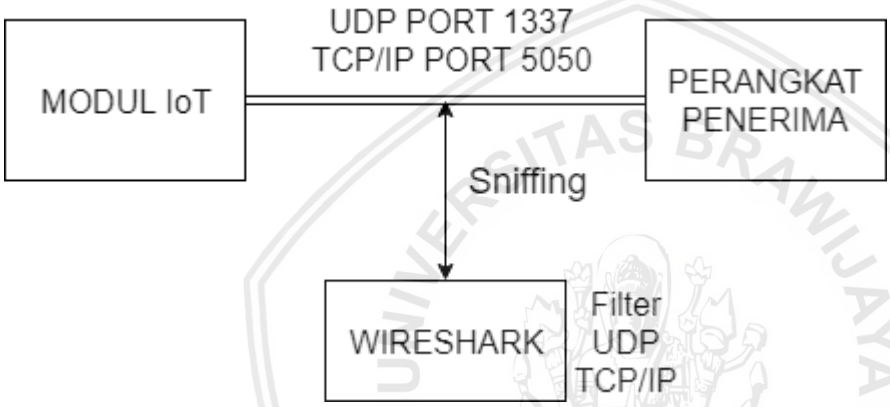
Proses kerja sistem diawali dengan pengambilan nilai suhu dan kelembaban dari sensor BME280. Program enkripsi AES-128 yang ditanamkan pada modul IoT kemudian melakukan proses enkripsi sehingga akan menghasilkan *ciphertext* atau data terenkripsi. *Ciphertext* kemudian dikirimkan melalui UDP dengan *port* 1337 dan TCP/IP dengan *port* 5050 ke perangkat penerima. Pada perangkat penerima terdapat program untuk menerima data dari modul IoT. Skema proses dari sistem ditunjukkan pada Gambar 3.2.



Gambar 3.1 Gambaran umum sistem



Gambar 3.2 Diagram proses kerja sistem

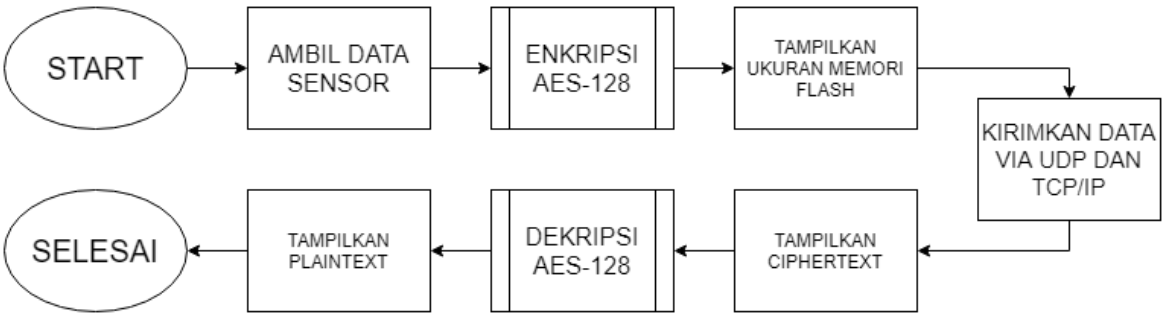


Gambar 3.3 Proses *sniffing* menggunakan *Wireshark*

Ciphertext dikirimkan ke perangkat penerima melalui protokol UDP port 1337 dan TCP/IP port 5050. Untuk melihat apakah informasi yang dikirim merupakan *ciphertext* dapat diketahui dengan melakukan *sniffing* pada jaringan menggunakan *Software Wireshark*. Paket yang ditangkap kemudian disaring untuk mendapatkan paket yang hanya dilewatkan pada UDP port 1337 dan TCP/IP port 5050. Selain itu agar *ciphertext* yang dikirimkan dapat diterima, maka program untuk menerima data (program penerima) UDP dan TCP/IP yang ditulis dengan bahasa pemrograman Python 3 dijalankan pada perangkat penerima.

3.3 Perancangan Sistem Enkripsi/Dekripsi AES-128

Gambar 3.4 menunjukkan blok diagram proses enkripsi pada modul IoT *Particle Photon* hingga proses dekripsi pada perangkat penerima.



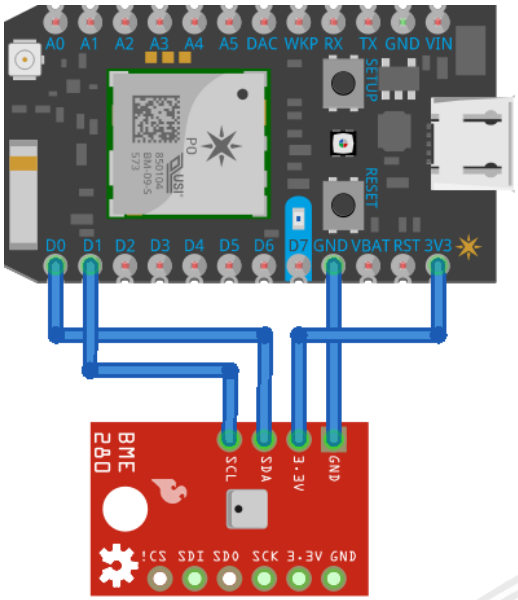
Gambar 3.4 Blok diagram proses enkripsi

Blok diagram di atas menunjukkan proses yang terjadi adalah sebagai berikut:

a. Perangkat mengambil data dari sensor.
b. Kemudian terdapat data sensor yang masuk sebagai *plaintext.*
c. Setelah itu masuk ke dalam proses enkripsi AES-128.
d. Menampilkan hasil *upload* program enkripsi AES-128 berupa ukuran memori *flash*.
e. *Ciphertext* dikirim ke perangkat penerima melalui UDP dan TCP/IP.
f. Pada perangkat penerima akan ditampilkan *ciphertext* yang telah terkirim.
g. Kemudian *ciphertext* akan masuk ke dalam proses dekripsi.
h. Hasil dekripsi ditampilkan pada perangkat penerima.

3.3.1 Skematik Sensor

Skematik sensor adalah diagram untuk menunjukkan hubungan antara beberapa pin yang ada pada 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. Nilai keluaran dari sensor kemudian dikonversi menjadi nilai dalam satuan suhu dalam derajat celcius (℃) dan kelembaban dalam persen (%) dengan menggunakan program. Gambar 3.5 menunjukkan skematik sensor.



Gambar 3.5 Skematik sensor BME280 dengan *Particle Photon*

Pin 3.3V sensor BME280 dihubungkan ke pin 3V3 *Particle Photon*. Pin GND sensor BME280 dihubungkan dengan pin GND *Particle Photon*. Pin SDA sensor BME280 dihubungkan pada pin D0 *Particle Photon* dan pin SCL sensor BME280 dihubungkan dengan pin D1 *Particle Photon*. Agar lebih mudah dipahami, skematik sensor BME280 dan modul IoT *Particle Photon* juga dapat dilihat pada Tabel 3.1.

Tabel 3.1
Wiring pin sensor BME280 dengan *pin Particle Photon*

Pin BME280	Pin Particle Photon
3,3V	3V3
GND	GND
SDA	D0
SCL	D1

3.3.2 Karakterisasi Sensor

a. Tujuan

Karakterisasi sensor dilakukan untuk mengetahui keakuratan pembacaan sensor BME280 terhadap suhu dan kelembaban sebenarnya dengan membandingkan nilai keluaran dari sensor BME280 dengan nilai pada termometer dan higrometer.

b. Peralatan yang digunakan

- Sensor BME280.
- Modul IoT *Particle Photon*.

- Termometer digital.
- Higrometer digital.
- *Minicom* yang telah terinstall pada terminal atau *serial monitor* IDE.

c. Langkah karakterisasi

- Menghubungkan sensor BME280 dengan modul IoT *Particle Photon* (Gambar 3.5.)
- Menyalakan termometer digital dan higrometer digital.
- Membuka *Minicom* pada terminal atau *serial monitor*.
- Menghubungkan modul IoT *Particle Photon* dengan *Personal Computer* (PC) menggunakan kabel USB.
- Mengunggah program untuk membaca sensor BME280 dan mengeluarkan *output* pada *serial monitor* IDE.
- Mengamati dan mencatat apakah data hasil pembacaan suhu dan kelembaban oleh sensor BME280 sesuai dengan termometer dan higrometer digital.

d. Hasil analisis

Berikut merupakan hasil karakterisasi sensor BME280 yang ditunjukkan pada Tabel 3.2.

Tabel 3.2
Karakterisasi sensor BME280

No.	Suhu dibaca termometer (℃)	Suhu dibaca BME280 (℃)	Kelembaban dibaca higrometer (%RH)	Kelembaban dibaca BME280 (%RH)
1.	26	26,3	54	54,09
2.	26	26,2	54	54,09
3.	26	26,2	54	54,08
4.	27	27,5	50	50,20
5.	27	27,6	50	50,34
6.	27	27,8	48	48,15
7.	28	28,3	48	48,40
8.	28	28,1	48	48,56
9.	30	30,2	40	40,62
10.	30	30,2	40	40,74
	Total		Total	
	275	278,4	486	489,27

Perhitungan kesalahan pembacaan sensor:

$$\%\ error = \frac{|\sum \text{Hasilpengukuran} - \sum \text{Re ferensi}|}{\sum \text{Referensi}} \times 100\%$$

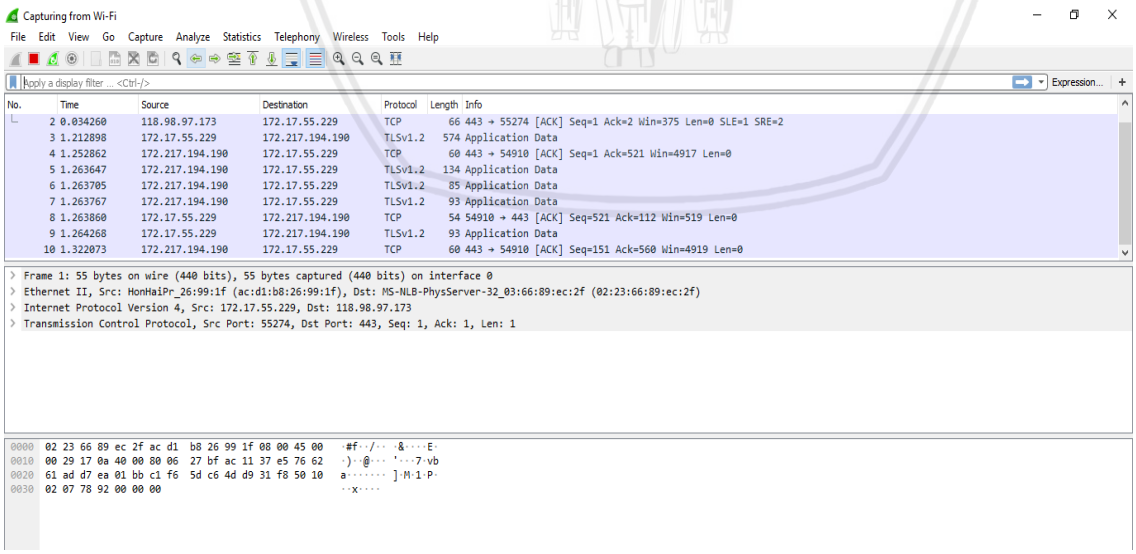
$$\%\ error \text{ suhu} = \frac{|278.4\text{-}275|}{275} \times 100\% = 1{,}23\%$$

$$\%\ error \text{ kelembaban} = \frac{|489.27\text{-}486|}{486} \times 100\% = 0{,}672\%$$

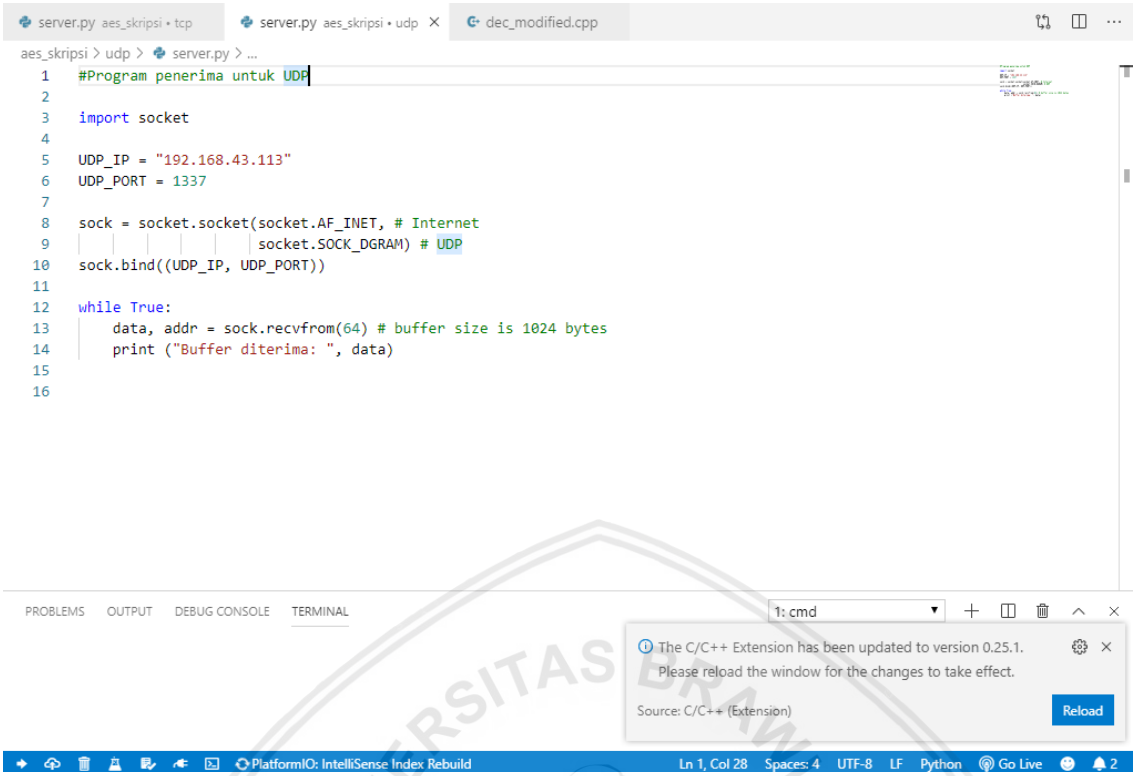
Berdasarkan hasil karakterisasi pada Tabel 3.2 terkait persentase *error* pembacaan suhu dan kelembaban sensor BME280 terhadap pembacaan suhu dan kelembaban dari termometer dan higrometer, didapatkan hasil bahwa nilai *error* pembacaan untuk suhu sebesar 1,23%, sedangkan nilai *error* pembacaan untuk kelembaban sebesar 0,672%.

3.3.3 Pengiriman Data

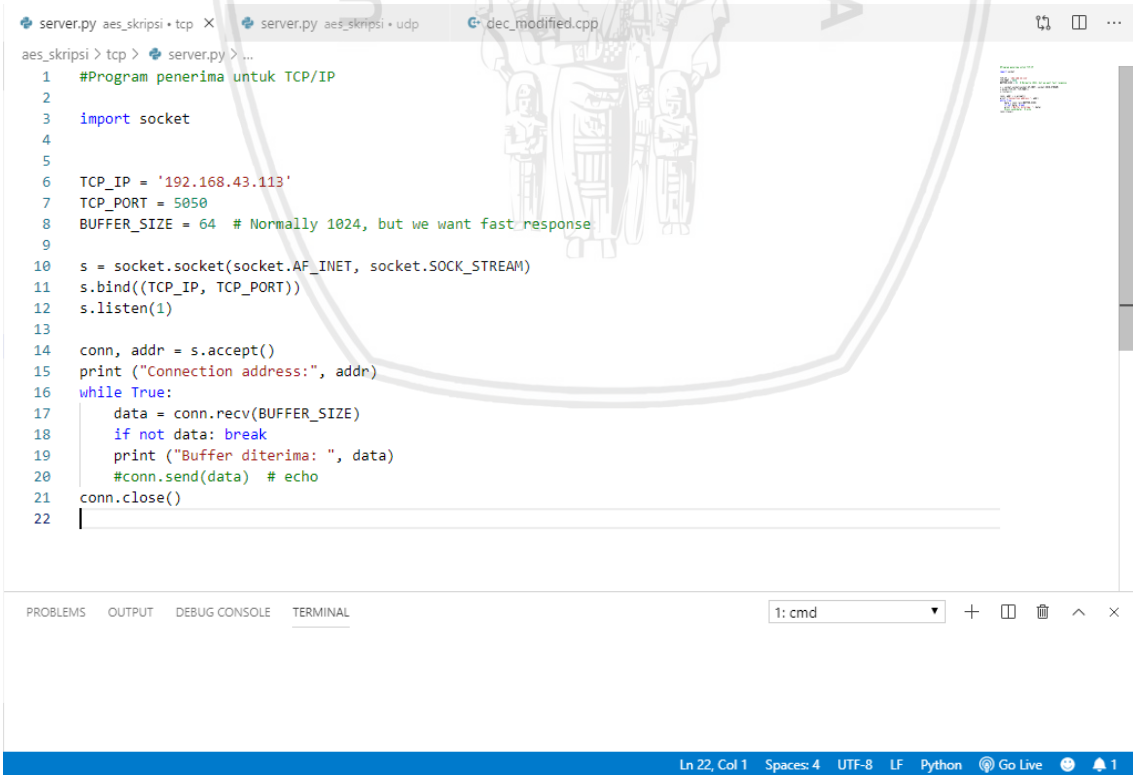
Data dikirim melalui UDP dan TCP/IP setelah proses enkripsi selesai. Data yang dikirim adalah *ciphertext* atau data sensor yang telah terenkripsi. Pada modul IoT *Particle Photon* ditanami program untuk mengirim data dengan menggunakan pustaka bawaan yang telah tersedia pada *Particle Build IDE* agar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



Gambar 3.6 Analisa paket menggunakan *Wireshark*



Gambar 3.7 Pemrograman *socket* untuk komunikasi menggunakan UDP



Gambar 3.8 Pemrograman *socket* untuk komunikasi menggunakan TCP/IP

Langkah-langkah yang dilakukan untuk melakukan pengiriman data adalah sebagai berikut:

1. Mendefiniskan alamat IP tujuan dan port tujuan pengiriman data UDP dan TCP/IP pada program pengirim untuk modul IoT *Particle Photon*.
2. Mendefinisikan data hasil keluaran sensor sebagai data yang akan dikirim oleh modul IoT.
3. Format data yang dikirimkan adalah sebagai berikut:
   t x, h y; dimana, *x = suhu*, *y = kelembaban*.
4. Menyiapkan program penerima data pada *Visual Studio Code*, dan me-*run* program tersebut.
5. *Upload* program pengirim pada modul IoT.
6. Melihat hasil pengiriman data pada *terminal Visual Studio Code*.
7. Mencatat data yang dikirim oleh modul IoT dan data yang diterima oleh komputer, serta apakah data yang dikirim berhasil diterima oleh program penerima atau tidak.

Pengiriman data melalui UDP menggunakan alamat IP 192.168.43.113 dan port 1337, hasil pengiriman melalui UDP dapat dilihat pada Tabel 3.5.

Tabel 3.3
Hasil pengiriman data sensor melalui UDP

Data dikirim oleh modul IoT	Data diterima oleh komputer	Terkirim/tidak
t 27.1, h 61.36	t 27.1, h 61.36	Terkirim
t 27.1, h 61.35	t 27.1, h 61.35	Terkirim
t 27.1, h 61.37	t 27.1, h 61.37	Terkirim
t 27.1, h 61.40	t 27.1, h 61.40	Terkirim
t 27.1, h 61.37	t 27.1, h 61.37	Terkirim
t 27.1, h 61.33	t 27.1, h 61.33	Terkirim
t 27.1, h 61.35	t 27.1, h 61.35	Terkirim
t 27.1, h 61.38	t 27.1, h 61.38	Terkirim
t 27.1, h 61.37	t 27.1, h 61.37	Terkirim
t 27.1, h 61.41	t 27.1, h 61.41	Terkirim

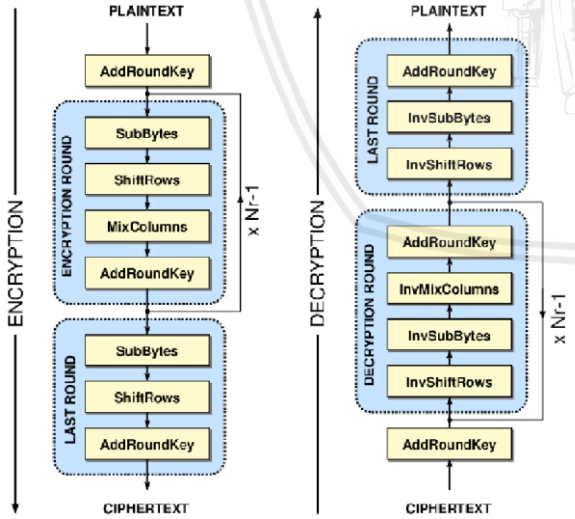
Tabel 3.3 menunjukkan bahwa data yang dikirim oleh modul IoT dan data yang diterima oleh komputer menghasilkan hasil yang sama, sehingga semua data terkirim dan pengiriman data melalui UDP dapat dikatakan berjalan dengan baik. Pengiriman data sensor melalui TCP/IP menggunakan alamat IP 192.168.43.113 dan port 5050, hasil pengiriman melalui TCP/IP dapat dilihat pada Tabel 3.6. di bawah ini.

Tabel 3.4
Hasil pengiriman data sensor melalui TCP/IP

Data dikirim oleh modul IoT	Data diterima oleh komputer	Terkirim/tidak
t 27.1, h 61.12	t 27.1, h 61.12	Terkirim
t 27.1, h 61.09	t 27.1, h 61.09	Terkirim
t 27.1, h 61.13	t 27.1, h 61.13	Terkirim
t 27.1, h 61.13	t 27.1, h 61.13	Terkirim
t 27.1, h 61.12	t 27.1, h 61.12	Terkirim
t 27.1, h 61.10	t 27.1, h 61.10	Terkirim
t 27.1, h 61.10	t 27.1, h 61.10	Terkirim
t 27.1, h 61.11	t 27.1, h 61.11	Terkirim
t 27.1, h 61.11	t 27.1, h 61.11	Terkirim
t 27.1, h 61.11	t 27.1, h 61.11	Terkirim

Tabel 3.4 menunjukkan hasil pengiriman data sensor melalui TCP/IP. Data yang dikirim oleh modul IoT dengan data yang diterima oleh komputer keduanya menghasilkan keluaran yang sama dan semua data berhasil terkirim, sehingga pengiriman data melalui TCP/IP dapat dikatakan berjalan dengan baik.

3.4 Penerapan Enkripsi AES-128

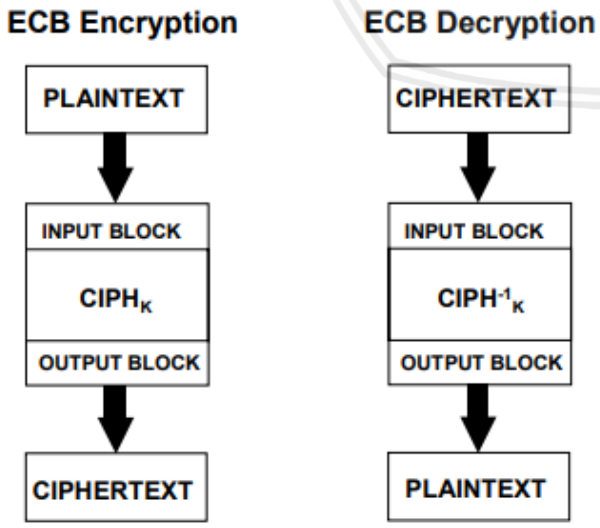


Gambar 3.9 Diagram proses enkripsi dan dekripsi AES
Sumber: (Arrag, et al., 2012)

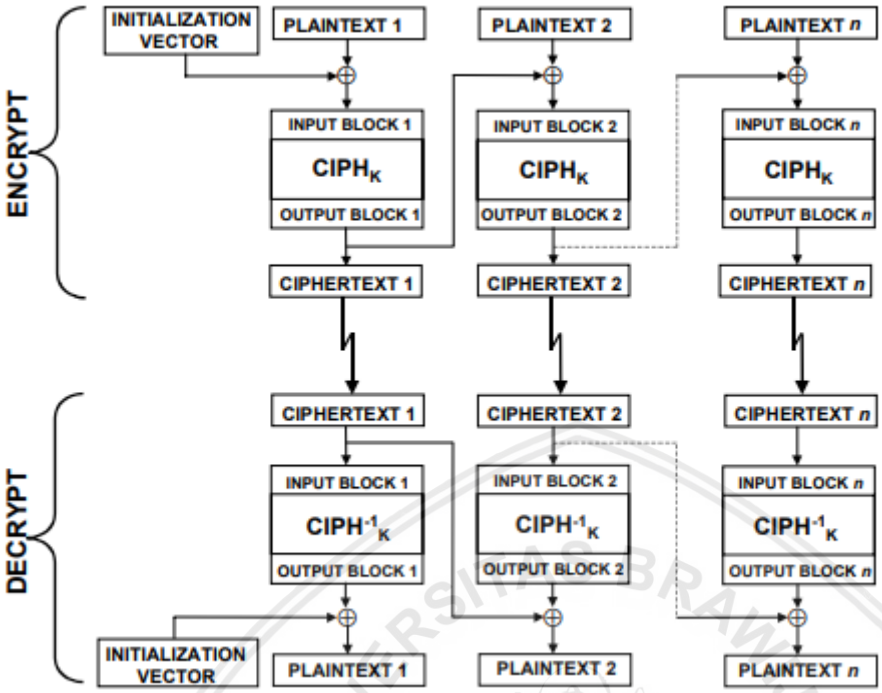
Gambar 3.9 merupakan diagram proses algoritma enkripsi dan dekripsi AES yang terdiri dari 4 fungsi transformasi yaitu; *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey*. Pada awal proses enkripsi, *plaintext* akan mengalami transformasi *Byte* pada fungsi

AddRoundKey. Kemudian, *state* akan mengalami transformasi pada fungsi *SubBytes*, *ShiftRows*, *MixColumns*, dan *AddRoundKey* secara berulangkali (proses Nr). Proses Nr dalam algoritma 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 dekripsi AES pada dasarnya mirip dengan struktur proses enkirpsi, namun menggunakan fungsi invers yaitu; *InvSubBytes*, *InvShiftRows* dan *InvMixColumns*. Sedangkan untuk mode enkripsi yang digunakan pada kajian ini menggunakan mode ECB. Pemilihan mode ECB didasarkan pada sifatnya yang sederhana dengan desain algoritma paralel untuk enkripsi data yang paling cepat di antara mode enkripsi lainnya (Wang, 2019). Namun *ciphertext* yang dihasilkan nantinya bersifat deterministik sesuai kunci yang telah didefinisikan. Dari sifat mode ECB tersebut dapat dikatakan mode ini memiliki kekurangan dimana mode ini akan menghasilkan *ciphertext* yang identik jika digunakan untuk mengenkripsi lebih dari satu blok *plaintext* dengan kunci yang sama. Mode enkripsi *block cipher* lain yang disarankan dan dijadikan standar adalah mode CBC memiliki penambahan beberapa operasi lain *Initialization Vector* (IV). Sehingga mode CBC mampu menghasilkan ciphertext berbeda meskipun dengan *plaintext* dan kunci yang sama. Akan tetapi mode CBC tidak didesain untuk komputasi paralel yang menyebabkan performa mode CBC tidak lebih baik dari ECB. Mode enkripsi *block cipher* ECB dan CBC ditunjukkan pada Gambar 3.10 dan 3.11.



Gambar 3.10 Mode ECB
Sumber: (Dworkin, 2001)



Gambar 3.11 Mode CBC
Sumber: (Dworkin, 2001)

3.4.1 *Hex/Un-Hex*

Hex/Un-Hex merupakan salah satu fungsi dari seluruh fungsi pada 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*. Hasil dari fungsi *Hex* ditunjukkan pada Tabel 3.5.

Tabel 3.5
Hasil konversi pada fungsi *Hex*

Data Input	Panjang Data	Hasil Fungsi Hex	Panjang *Padding*
Hello!	6 *Byte*	48656c6c6f2100000000000000000000	10 Byte
Hello!123	10 *Byte*	48656c6c6f2131323334000000000000	6 Byte
Hello!123456	12 *Byte*	48656c6c6f2131323334353600000000	4 Byte
t 26.5, h 55.79	15 *Byte*	742032362e352c20682035352e373900	1 Byte
0123456789abcdef	16 *Byte*	30313233343536373839616263646566	0 Byte

Tabel 3.5 menunjukkan panjang *padding* akan mengikuti variasi dari ukuran data *input*. Panjang *padding* dapat diketahui dengan selisih dari 16 *Byte* karakter dikurangi dengan panjang karakter data *input*. Sedangkan untuk fungsi *Un-Hex* adalah fungsi pengembalian dari bentuk heksadesimal ke bentuk *string*. Hasil dari fungsi *Un-Hex* dapat dilihat pada Tabel 3.6.

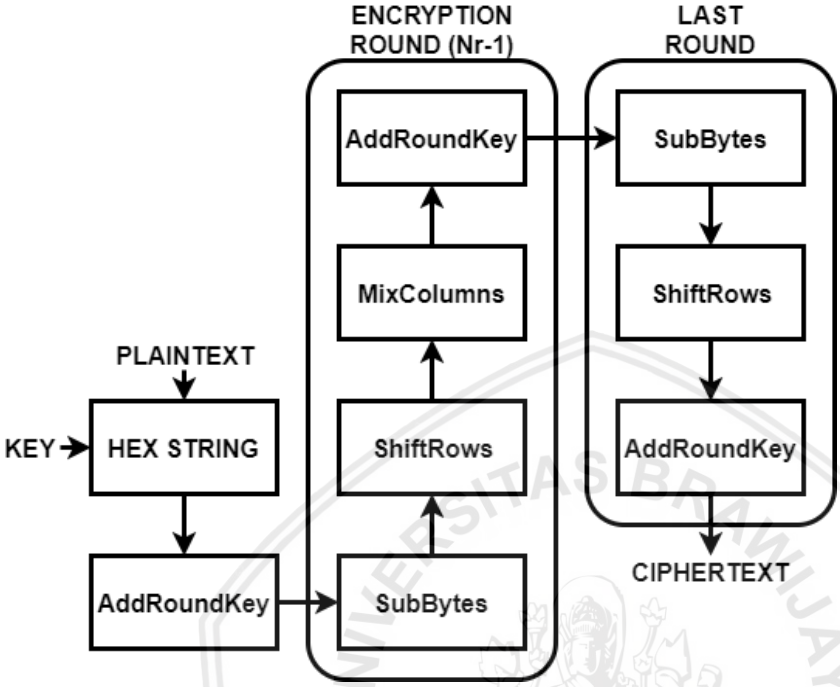
Tabel 3.6
Hasil keluaran fungsi *Un-Hex*

Data asli	Hasil fungsi *Hex*	Hasil fungsi *Un-Hex*	Sesuai / Tidak
Hello!	48656c6c6f2100000000000000000000	Hello!	Sesuai
Hello!123	48656c6c6f2131323334000000000000	Hello!123	Sesuai
Hello!123456	48656c6c6f2131323334353600000000	Hello!123456	Sesuai
t 26.5, h 55.79	742032362e352c20682035352e373900	t 26.5, h 55.79	Sesuai
0123456789abcdef	30313233343536373839616263646566	0123456789abcdef	Sesuai

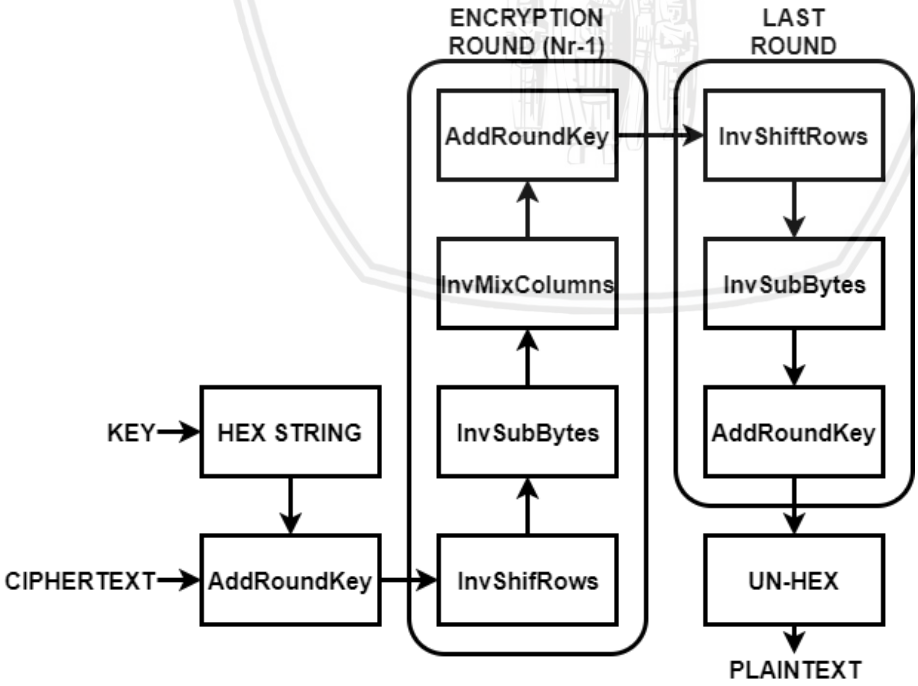
Tabel 3.6 menunjukkan dari lima kali hasil keluaran fungsi *Un-Hex* didapatkan bahwa fungsi ini mampu mengembalikan semua hasil konversi heksadesimal dari fungsi *Hex* menjadi bentuk karakter yang sesuai dengan data aslinya (data *input*).

Konversi heksadesimal dengan fungsi *Hex/Un-Hex* ini dibutuhkan karena *plaintext* dan kunci pada implementasi perangkat lunak algoritma enkripsi AES membutuhkan masukan dengan format heksadesimal. Sedangkan keluaran dari hasil enkripsi (*ciphertext*) juga merupakan format heksadesimal yang jika didekripsi menjadi *plaintext* akan menghasilkan format heksadesimal pula. Pengembalian dari format heksadesimal menjadi format dalam bentuk *string* perlu dilakukan agar lebih mudah dibaca dan dimengerti. Hal ini bertujuan

agar pada program enkripsi AES yang dibuat tidak memerlukan fungsi konversi dalam bentuk lain yang akan membuat program menjadi semakin kompleks. Gambar 3.12 menunjukkan proses enkripsi, sedangkan gambar 3.13menunjukkan proses dekripsi.



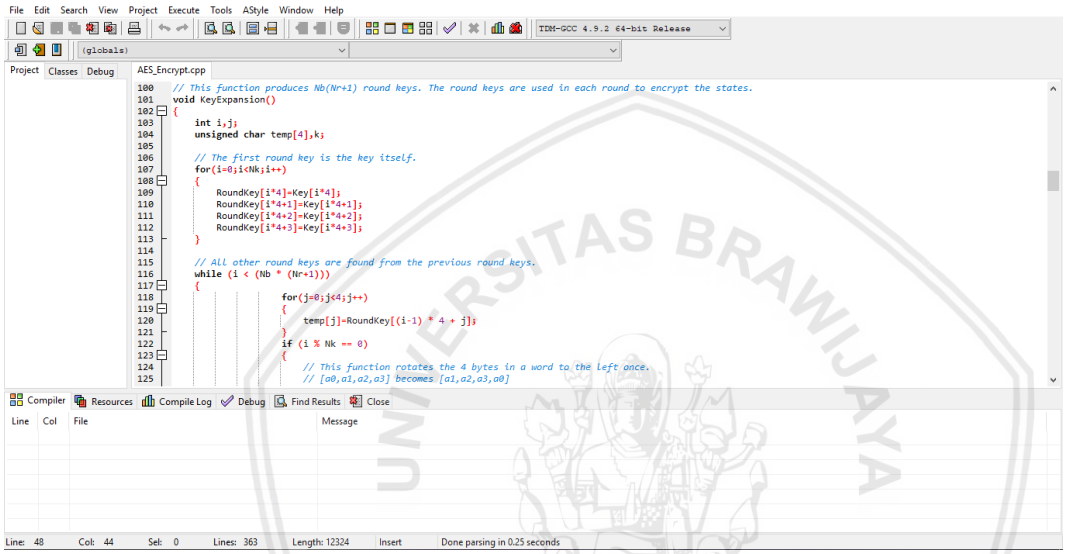
Gambar 3.12 Diagram alir program enkripsi AES-128



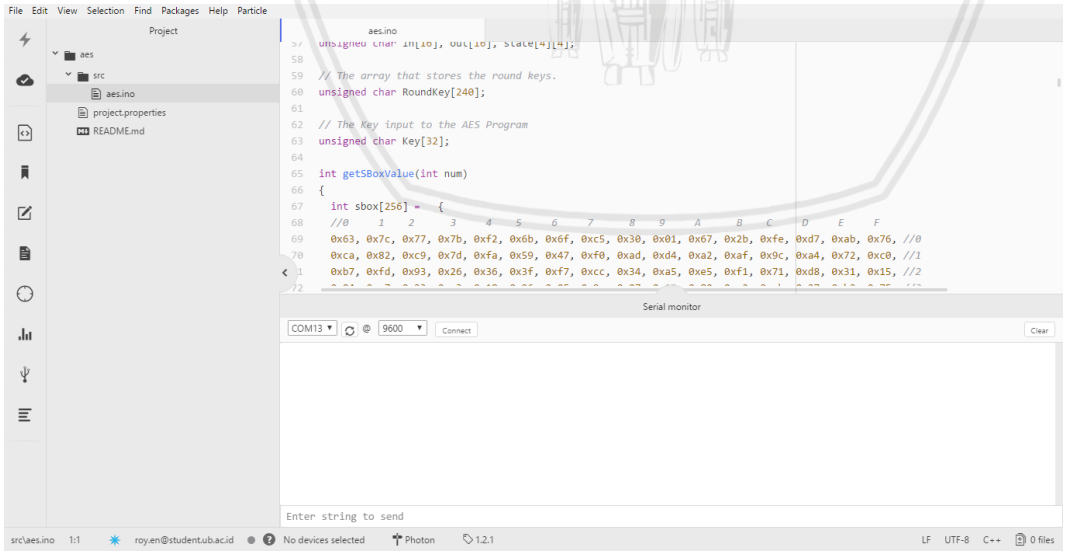
Gambar 3.13 Diagram alir program dekripsi AES-128

Diagram alir program enkripsi/dekripsi AES-128 di atas menunjukkan penambahan fungsi *Hex String* yang digunakan untuk melakukan konversi dari format karakter menjadi

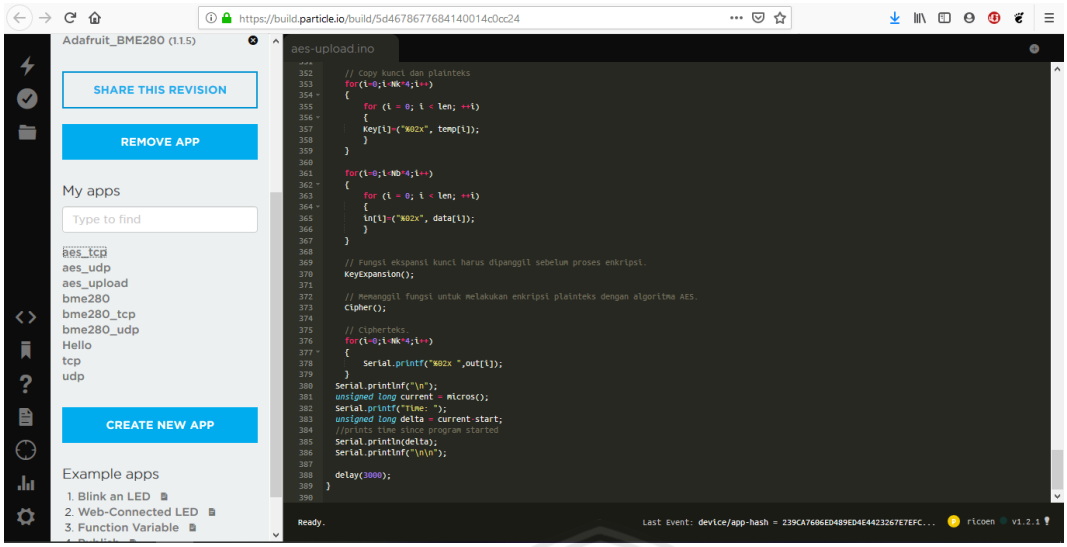
notasi heksadesimal dalam bentuk array. Program enkripsi/dekripsi ditulis dengan menggunakan bahasa pemrograman C. Program ini ditulis dengan menggunakan *DEV-C++* untuk mempermudah proses *debugging*, sedangkan untuk proses upload program ke mikrokontroler menggunakan *Particle-Dev IDE* yang berbasis *ATOM text editor* atau bisa juga menggunakan *Particle Build (Particle Web IDE)* yang berbasis aplikasi web dan *cloud* seperti yang ditunjukkan pada Gambar 3.14 hingga Gambar 3.16.



Gambar 3.14 Pembuatan program enkripsi AES-128 menggunakan *DEV-C++*



Gambar 3.15 Antarmuka *Particle-Dev IDE*



Gambar 3.16 Antar muka *Particle Build* (*Particle Web IDE*)

3.4.3 Performa AES

Performa dari suatu algoritma enkripsi dapat ditentukan dari nilai *throughput* yang didasarkan pada waktu enkripsi dan ukuran *plaintext*. Kedua faktor tersebut menjadi dasar perhitungan *throughput* untuk menghasilkan suatu *ciphertext* (Kumar & Tiwari, 2012). 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. Waktu eksekusi ditentukan dengan mengambil nilai *median* pada program (Chiu, 2018) atau waktu akhir dikurangi waktu awal dari fungsi *get_current_time* atau fungsi untuk mengambil waktu eksekusi (Moreno & Fischmeister, 2017).

Penentuan waktu enkripsi dapat dinyatakan dengan Persamaan (3.1). Sedangkan *throughput* merupakan skema perhitungan sebagai ukuran *plaintext* dalam *Byte* dibandingkan dengan waktu enkripsi (Elminaam, et al., 2010).

$$Excecution\ Time = end\ time\ (s) - start\ time\ (s) \quad \text{.....(3.1)}$$

Berdasarkan perhitungan *encryption time* pada Persamaan (3.1) maka performa enkripsi (*troughput*) dapat dicari dengan menggunakan Persamaan (3.3) sebagai berikut:

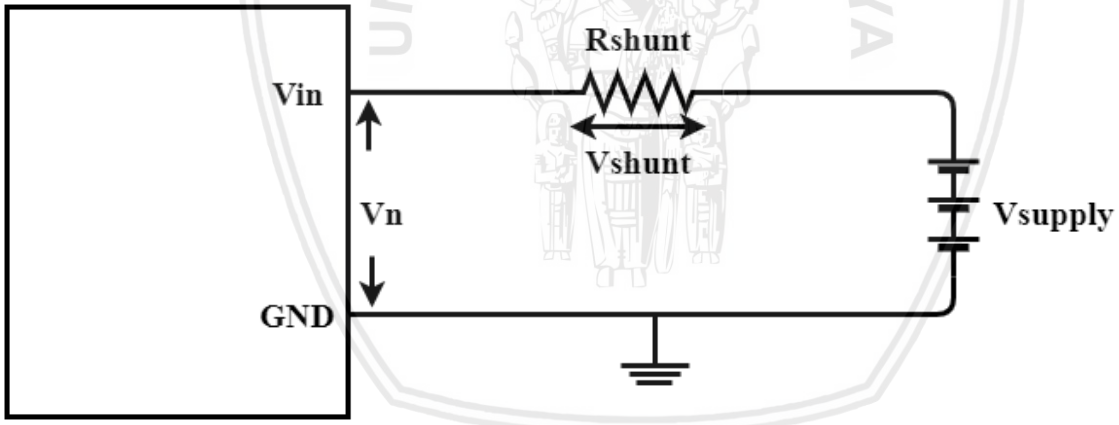
$$Throughput = \frac{plaintext\ size\ (bit)}{excecution\ time\ (s)} \quad \text{.....(3.2)}$$

Kekuatan algoritma enkripsi AES-128 dapat diketahui dengan cara menghitung nilai *Avalanche Effect* yang merupakan perbandingan antara jumlah bit yang membalik dengan jumlah total bit pada *ciphertext* (Echeverri, 2017) yang dinyatakan secara matematis pada Persamaan (3.3).

$$Avalanche\ Effect\ (\%) = \frac{\text{Jumlah bit yang terbalik } (bit)}{\text{Jumlah bit ciphertext } (bit)} \times 100 \quad \text{.....(3.3)}$$

3.4.4 Konsumsi Energi

Pengukuran dilakukan untuk membandingkan konsumsi energi yang digunakan oleh modul IoT ketika dalam kondisi *idle* maupun saat menjalankan rutin program. Konsumsi energi yang digunakan oleh program ditentukan dengan pengukuran tegangan rata-rata dalam rentang waktu tertentu pada R_{SHUNT} menggunakan *oscilloscope*. Tegangan pada R_{SHUNT} juga dapat disebut sebagai V_{SHUNT}. Rangkaian pengukuran nilai V_{SHUNT} ditunjukkan pada Gambar 3.17.



Gambar 3.17 Rangkaian pengukuran tegangan *Shunt*

Setelah nilai V_{SHUNT} didapatkan, kemudian melakukan perhitungan I_{SHUNT} agar nilai daya (P) dapat dicari. Pengukuran daya dapat dilakukan dengan menggunakan hukum *Ohm* (Abdelgawad, 2014). Daya dapat didefinisikan sebagai perkalian dari tegangan dikalikan dengan arus. Secara matematis perhitungan daya ditunjukkan pada Persamaan (3.4) sebagai berikut:

$$P = V_n \times I = (V_{Supply} - V_{SHUNT}) \times \frac{V_{SHUNT}}{R_{SHUNT}} \quad \text{.....(3.4)}$$

Dengan:

P = Daya (Watt)

I = Arus (Ampere)

R_{SHUNT} = Resistor *Shunt* (100 Ω)

V_n = (Volt)

V_{SHUNT} = Tegangan yang terukur pada R_{SHUNT} (Volt)

V_{Supply} = Tegangan *supply* (Nilai tegangan terukur 4.8 V dari 3 buah baterai 1.5 V)

Setelah mendapatkan nilai daya, kemudian melakukan perhitungan energi yang digunakan dengan menggunakan dasar hukum *Joule*. Nilai daya yang terukur dalam satuan Watt (W) dapat didefinisikan sebagai nilai dari energi per satuan waktu (J *per second*) (Donnay, 2013). Sehingga daya dan energi memiliki hubungan yang dapat dinotasikan secara matematis pada Persamaan (3.5). Perhitungan energi dilakukan dengan konversi satuan daya Watt menjadi miliWatt (mW) dan satuan energi Joule menjadi miliJoule (mJ).

$$E = P \times t \quad (3.5)$$

Dengan:

E = Energi (mJ)

P = Daya (mW)

t = Waktu (milidetik)

3.6 Parameter Pengujian

Setelah program enkripsi dan dekripsi AES-128 diterapkan, langkah selanjutnya yaitu melakukan pengujian dan analisis terhadap keseluruhan sistem. Pengujian dilakukan dengan menentukan dan menyusun tujuan pengujian, alat dan bahan yang digunakan, serta prosedur pengujian. Pengujian yang dilakukan meliputi:

1. Kesesuaian hasil enkripsi/dekripsi dan kekuatan *ciphertext*

   Pengujian ini dilakukan untuk mengetahui adanya *error* pada proses enkripsi dan dekripsi dimana hasil dekripsi *ciphertext* tidak sesuai dengan *plaintext*. Selain itu juga untuk mengetahui tingkat keacakan *ciphertext* dengan menggunakan metode *Avalanche Effect*.
2. Performa enkripsi

Pengujian ini dilakukan untuk mengetahui lamanya waktu yang dibutuhkan dan *throughput* dari proses enkripsi pada modul IoT

3. Memori yang digunakan oleh program enkripsi.
   Pengujian ini bertujuan untuk mengetahui besar memori *flash* dan *Random Access Memory* (RAM) yang digunakan oleh program.
4. Pengiriman *ciphertext* melalui UDP dan TCP/IP
   Pengujian ini dilakukan untuk mengetahui keberhasilan pengiriman *ciphertext* melalui UDP dan TCP/IP serta pengaruh dari penerapan program enkripsi AES-128 terhadap *delay* pengiriman paket data.
5. Konsumsi energi oleh program
   Pengujian ini dilakukan untuk mengetahui penggunaan energi yang timbul oleh adanya penerapan enkripsi AES-128 pada modul IoT.

Pengujian dilakukan secara bertahap untuk mendapatkan data dari setiap parameter yang telah ditentukan. Setelelah pengujian dilakukan secara keseluruhan kemudian dilakukan analisis dari hasil pengujian sebagai dasar pengambilan kesimpulan dan saran.

3.7 Pengambilan Kesimpulan dan Saran

Kesimpulan dapat diambil berdasarkan hasil penerapan dan pengujian sesuai dengan tujuan dan rumusan masalah yang ditentukan. Sehingga, apabila hasil yang didapatkan telah sesuai dengan yang direncanakan sebelumnya, maka penerapan program enkripsi AES-128 pada modul IoT tesebut telah berhasil mencapai target. Saran diberikan setelah melihat masih adanya kekurangan dalam sistem dengan harapan adanya pengembangan dan perbaikan terhadap realisasi sistem ini pada kajian selanjutnya agar didapatkan hasil yang lebih baik.

BAB IV
HASIL DAN PEMBAHASAN

Hasil dan pembahasan dibuat dengan melakukan prosedur pengujian dan pengamatan sistem secara keseluruhan. Pengujian yang dilakukan meliputi empat pengujian, yaitu pengujian hasil enkripsi/dekripsi yang sesuai, performa enkripsi pada modul IoT, memori yang digunakan oleh program enkripsi, keberhasilan pengiriman *ciphertext* melalui UDP dan TCP/IP serta pengaruh penerapan program enkripsi AES-128 terhadap *delay* TCP/IP pada modul IoT. Pengujian yang dilakukan bertujuan untuk mengetahui apakah sistem yang dibuat sudah memberikan hasil yang sesuai dengan hasil yang diinginkan sekaligus untuk mengamati respon dari sistem yang dibuat dan menganalisa hasil dari pengujian serta pengamatan yang telah dilakukan.

4.1 Pengujian Kesesuaian dan Kekuatan Hasil Enkripsi/Dekripsi

4.1.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui apakah hasil enkripsi/dekripsi AES-128 dapat menghasilkan *ciphertext* maupun mengembalikan ke bentuk *plaintext* sebagaimana mestinya. Sehingga dapat diketahui apakah implementasi algoritma enkripsi dan dekripsi AES-128 dalam bentuk program yang dibuat telah sesuai dan berjalan sebagaimana mestinya atau tidak. Selain itu juga untuk mengetahui tingkat keacakan *ciphertext* yang dihasilkan dengan melihat nilai *Avalanche Effect*.

4.1.2 Peralatan yang Digunakan

- Modul IoT *Particle Photon*.
- Sensor BME280
- Kabel USB *Micro* B to USB *Type* A.
- Program enkripsi AES-128 untuk modul IoT.
- Program dekripsi AES-128.
- *Terminal* IDE.
- *Minicom* atau *serial monitor* IDE.

- CrypTool 2.1.

4.1.3 Prosedur Pengujian

- Menghubungkan sensor BME280 dan modul IoT *Particle Photon*.
- Menghubungkan modul IoT *Particle Photon* ke komputer melalui USB.
- Mengunggah program enkripsi AES-128 ke modul IoT.
- Menyiapkan program dekripsi AES-128, kemudian me-*run* program tersebut.
- Membuka *terminal* atau *serial monitor* IDE.
- Mengamati dan mencatat hasil keluaran enkripsi dan dekripsi.
- Mengubah satu bit input *plaintext*.
- Menghitung jumlah bit yang berubah.
- Menghitung nilai *Avalanche Effect*.

4.1.4 Hasil Pengujian

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). Setelah prosedur pengujian dilakukan, didapatkan hasil enkripsi berupa *ciphertext* yang ditunjukkan pada Tabel 4.1 sebagai berikut:

Tabel 4.1
Hasil enkripsi data sensor

Plaintext	*Ciphertext*
t 27.2, h 57.69	21 8e ec a4 6f 9e 9d 65 70 05 56 cc 2c a4 67 3c
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 57.65	d0 6f 11 ff a2 85 dd 33 9d 61 25 73 28 4f ad 2f
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	2f 60 85 eb 54 8c 3f 6a 2f 7f 23 f1 30 65 b2 9d
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 57.52	fd 00 62 4f b7 3b b6 0a de 7c b6 2e 4d d1 91 a4
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 57.62	0c 6e 3e 25 ba 90 88 58 be ad 99 fa 88 07 cf 88
t 27.4, h 57.61	48 14 ba 4f 9b b4 72 3f ad 94 ee ed 56 c8 2e 9f

Pengujian dekripsi dengan *ciphertext* dan kunci enkripsi yang sama menghasilkan *plaintext* yang ditunjukkan pada tabel 4.2 sebagai berikut:

Tabel 4.2
Hasil dekripsi

Plaintext (sebelum dienkripsi)	*Ciphertext*	*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 4.2 menunjukkan bahwa program enkripsi dapat menghasilkan *ciphertext* yang juga dapat dikembalikan menjadi *plaintext* oleh program dekripsi, sehingga dapat dikatakan bahwa program enkripsi dan dekripsi dapat berjalan dengan baik. Namun, dapat dilihat pula terdapat dua *ciphertext* yang sama dikarenakan *plaintext* dari data sensor juga menghasilkan keluaran yang sama. *Plaintext* tersebut adalah "t 27.4, h 57.66", sehingga akan menghasilkan *ciphertext* yang sama pula, yaitu "*2f 60 85 eb 54 8c 3f 6a 2f 7f 23 f1 30 65 b2 9d*". Hal tersebut disebabkan karena mode enkripsi yang digunakan adalah mode ECB di mana mode ini akan menghasilkan *ciphertext* sesuai dengan kunci yang telah didefinisikan. Sehingga hasil enkripsi dapat dikatakan masih memiliki kekurangan. Mode ECB yang menghasilkan *ciphertext* sama beberapa kali akan lebih rentan terhadap serangan. Sehingga diperlukan mode enkripsi lain yang lebih aman.

Selain untuk mengetahui kesesuaian hasil enkripsi dan dekripsi, pengujian ini juga memiliki tujuan untuk mengetahui tingkat keamanan dari algoritma enkripsi AES-128 yang telah diterapkan pada modul IoT. Untuk mengetahui tingkat keamanan AES-128 yang telah diterapkan maka dilakukan perhitungan nilai *Avalanche Effect* menggunakan Persamaan (3.4). Pengujian *Avalanche Effect* dilakukan dengan cara mengubah satu bit dari *plaintext*. Perubahan satu bit ditunjukkan dengan nilai heksadesimal yang diberi warna merah. Hasil yang diperoleh dari perhitungan yang telah dilakukan dapat dilihat pada Tabel 4.3 sebagai berikut:

Tabel 4.3
Hasil pengujian *Avalanche Effect* dengan perubahan 1 bit *plaintext*

	Input	*Ciphertext*	Nilai *Avalanche Effect*
Plaintext 1	48 65 6C 6C 6F 20 77 6F 72 6C 64 21 00 00 00 00	A9 3A 4D 66 B0 19 8F 3F A8 BB 62 13 B0 ED 9E E5	53,10%
Modified plaintext 1	48 65 6C 6C 6F 20 77 6F 72 6C 64 23 00 00 00 00	A9 CB 3F 7C C3 27 14 EC C4 41 F9 F2 A7 26 2C 57	
Plaintext 2	48 61 69 20 74 65 6d 61 6e 20 74 65 6d 61 6e 21	F4 0C 6C 95 9A DE 1F 94 A8 15 5E 41 FA C3 BF D7	53,90%
Modified plaintext 2	48 61 69 20 74 65 6d 61 6e 20 74 65 6d 61 7e 21	58 38 EC 04 85 25 F3 63 3F 59 27 A4 FB 0E C8 CA	
Plaintext 3	30 31 32 33 34 35 36 37 38 39 61 62 63 64 65 66	48 F1 D8 B7 A7 31 30 DC 66 05 98 68 76 63 8B B5	55,50%
Modified plaintext 3	30 31 32 33 34 35 36 37 38 39 61 62 63 64 65 67	D8 CC 93 4B AA C7 CA 0D 5C DD 70 92 8F 75 9A 7A	
Plaintext 4	74 20 32 35 2e 31 2c 20 68 20 35 36 2e 32 35 00	C5 3F D8 B2 F2 CB 10 92 17 E9 A2 FF A0 64 A2 FA	50,80%
Modified plaintext 4	74 20 32 35 2e 31 2c 20 68 20 35 36 2e 32 37 00	DE 34 C4 D2 03 1C 28 C7 0B 56 38 87 C3 97 CA 77	
Plaintext 5	74 20 32 35 2e 31 2c 20 68 20 35 38 2e 32 35 00	F0 A0 36 BC CE 00 41 04 7A 87 D4 DC 9F 56 FA BC	54,70%
Modified plaintext 5	74 20 32 35 2e 31 2c 20 68 20 35 38 2e 33 35 00	E0 58 58 03 B9 38 9E 86 25 F6 C3 24 14 0F 73 92	
Plaintext 6	48 69 20 77 65 6c 63 6f 6d 65 20 67 75 79 73 21	8D 96 5A 16 EB 93 66 76 A8 AE F6 8E 95 22 A1 4E	50%
Modified plaintext 6	48 69 20 77 65 6c 63 6f 6d 65 20 67 75 79 73 23	45 B9 9D 6F E6 A4 5B 35 BC B8 26 B3 DE CB B2 E5	

Berdasarkan Tabel 4.3 lima dari enam kali hasil pengujian didapatkan nilai *Avalanche Effect* lebih dari 50%. Jika hasil pengujian dirata-rata maka akan menghasilkan nilai *Avalanche Effect* sebesar 53%. Sehingga meskipun memiliki kekurangan pada *ciphertext* yang dihasilkan namun dapat dikatakan bahwa algoritma ini masih memiliki tingkat keacakan yang tinggi dan dapat dikatakan sebagai algoritma enkripsi yang aman.

4.2 Pengujian Performa Enkripsi

4.2.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui bagaimana performa dari program enkripsi AES-128 pada modul IoT *Particle Photon*. Parameter yang dilihat pada pengujian ini adalah waktu yang dibutuhkan untuk melakukan enkripsi dan *throughput* yang dihasilkan oleh modul IoT dalam satu kali proses enkripsi.

4.2.2 Peralatan yang Digunakan

- Modul IoT *Particle Photon*.
- Sensor BME280.
- Kabel USB *Micro* B to USB *Type* A.
- Program enkripsi AES-128 untuk modul IoT.
- Program dekripsi AES-128.
- *Terminal* IDE.
- *Minicom* atau *serial monitor* IDE.

4.2.3 Prosedur Pengujian

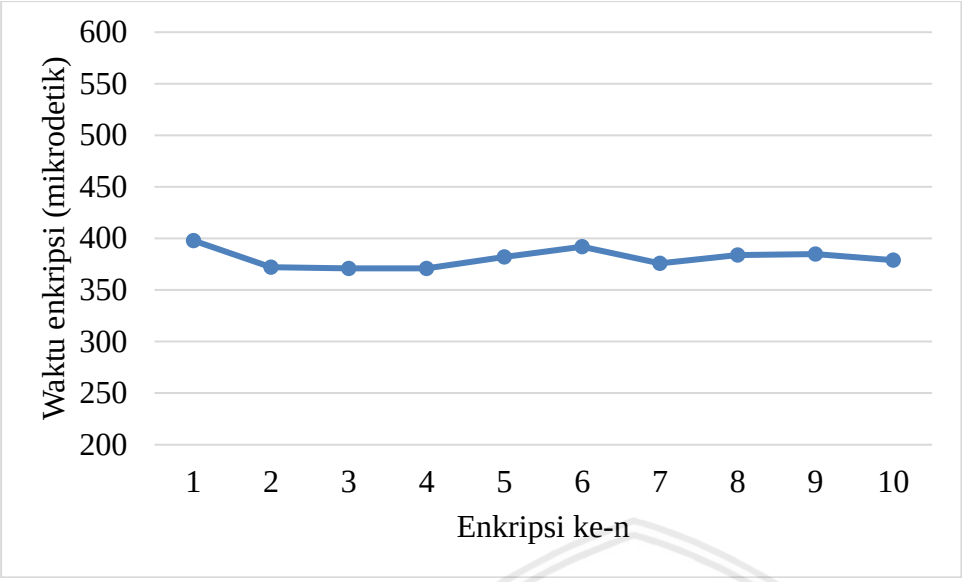
- Menghubungkan sensor BME280 dan modul IoT *Particle Photon*.
- Menghubungkan modul IoT *Particle Photon* ke komputer melalui USB.
- Mengunggah program enkripsi AES-128 ke modul IoT.
- Membuka *terminal* atau *serial monitor* IDE.
- Mengamati dan mencatat hasil keluaran berupa waktu enkripsi.

4.2.4 Hasil Pengujian

Hasil yang didapatkan setelah melakukan prosedur pengujian yaitu berupa waktu yang dibutuhkan untuk melakukan satu kali proses enkripsi. Hasil pengujian ditunjukkan pada Tabel 4.4 dan Gambar 4.1 sebagai berikut:

Tabel 4.4
Hasil pengujian waktu enkripsi

Enkripsi ke-n	Waktu enkripsi (mikrodetik)
1	398
2	372
3	371
4	371
5	382
6	392
7	376
8	384
9	385
10	379

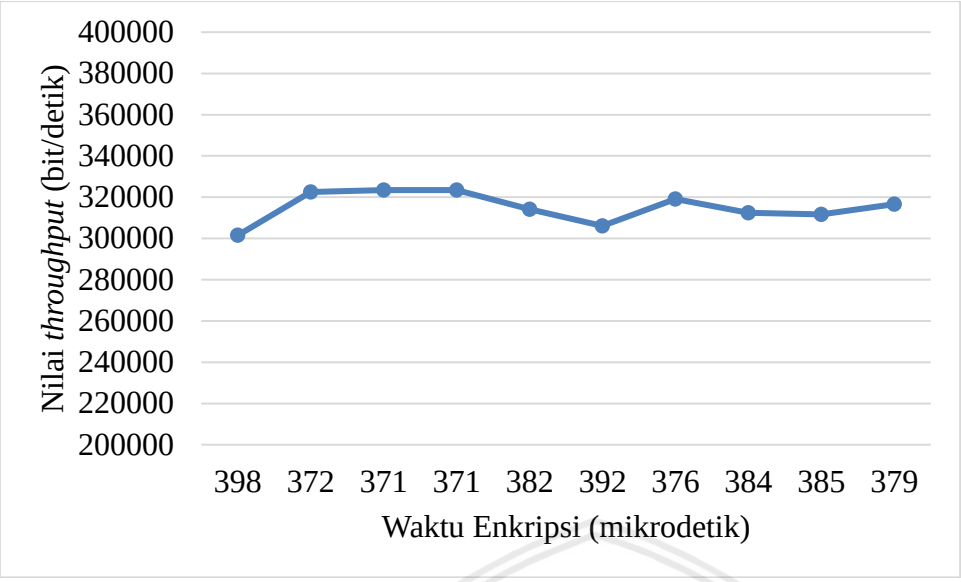


Gambar 4.1 Grafik pengujian waktu enkripsi

Berdasarkan Tabel 4.4. dan Gambar 4.1 didapatkan lama waktu yang diperlukan untuk satu kali proses enkripsi. Hasil pengujian menunjukkan rentang waktu dari 371 mikrodetik hingga 398 mikrodetik dengan waktu rata-rata selama 381 mikrodetik. Adapun *throughput* enkripsi dapat dicari dengan menggunakan Persamaan (3.3). Nilai *throughput* dari hasil pengujian dapat dilihat pada Tabel 4.5 dan Gambar 4.2 sebagai berikut:

Tabel 4.5
Nilai throughput program enkripsi AES-128

Waktu (mikrodetik)	Ukuran *plaintext* (bit)	*Throughput* (bit/detik)
398	120	301507
372	120	322581
371	120	323450
371	120	323450
382	120	314136
392	120	306122
376	120	319149
384	120	312500
385	120	311688
379	120	316622



Gambar 4.2 Grafik nilai *throughput* program enkripsi AES-128

Tabel 4.5 dan Gambar 4.2 didapatkan nilai *throughput* pada rentang 301507 bit/detik hingga 323450 bit/detik dengan rata-rata *throughput* sebesar 315120 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*.

4.3 Pengujian Memori Enkripsi

4.3.1 Tujuan Pengujian

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

4.3.2 Peralatan yang Digunakan

- Modul IoT *Particle Photon*.
- Sensor BME280
- Kabel USB *Micro* B to USB *Type* A.
- Program enkripsi AES-128 pada modul IoT.
- Program dekripsi AES-128 pada perangkat penerima.
- *Terminal* IDE.
- *Minicom* atau *serial monitor* IDE.
- *Particle Build* (*Particle Web* IDE).

4.3.3 Prosedur Pengujian

- Menghubungkan sensor BME280 dan modul IoT *Particle Photon*.
- Menghubungkan modul IoT *Particle Photon* ke komputer melalui USB.

- Mengunggah program enkripsi AES-128 ke modul IoT.
- Membuka *Particle Build* (*Particle Web* IDE).
- Membuka *terminal* atau *serial monitor* IDE.
- Mengamati dan mencatat hasil keluaran berupa memori *flash* dan RAM.

4.3.4 Hasil Pengujian

Hasil yang didapatkan setelah melakukan prosedur pengujian yaitu berupa memori *flash* yang digunakan setelah adanya penambahan program enkripsi. Hasil yang didapatkan pada pengujian ini dapat dilihat pada Tabel 4.6. Persentase penggunaan memori *flash* oleh program pada modul IoT *Particle Photon* dapat dilihat pada Tabel 4.7.

Tabel 4.6
Ukuran memori *flash* yang digunakan oleh tiap bagian program

Bagian Program	Ukuran memori *flash* (*Byte*)
Program pembaca nilai sensor	10.680
Program pembaca dan pengirim data sensor	12.480
Program pembaca dan enkripsi data sensor	14.068
Program pembaca, enkripsi dan pengirim data sensor	16.024

Tabel 4.7
Persentase penggunaan memori *flash*

Memori *flash* yang digunakan (*Byte*)	Tersedia (*Byte*)	Persentase (%)
10.680	125.000	8,54%
12.480		9,98%
14.068		11,25%
16.024		12,81%

Tabel 4.7 menunjukkan hasil pengukuran penggunaan memori *flash* oleh program yang dibagi menjadi empat bagian. Keempat bagian program tersebut yaitu program pembaca nilai sensor, program pembaca dan pengirim data sensor, program pembaca dan enkripsi data sensor, serta program pembaca, enkripsi dan pengirim data sensor. Dari hasil pengujian didapatkan penggunaan memori *flash* oleh keseluruhan program sebesar 16.024 *Byte* yang berarti hanya menggunakan 12,81% dari total memori *flash* yang tersedia sebesar 125.000 *Byte*.

Sedangkan untuk penggunaan RAM oleh program pada modul IoT dapat dilihat pada Tabel 4.8 dan Tabel 4.9 sebagai berikut:

Tabel 4.8
RAM yang digunakan oleh tiap bagian program

Bagian Program	RAM yang digunakan (*Byte*)
Program pembaca nilai sensor	1.612
Program pembaca dan pengirim data sensor	1.652
Program pembaca dan enkripsi data sensor	2.964
Program pembaca, enkripsi dan pengirim data sensor	3.020

Tabel 4.9
Persentase RAM yang digunakan oleh program

RAM yang digunakan (*Byte*)	Tersedia (*Byte*)	Persentase (%)
1.612	60.000	2,68%
1.652		2,75%
2.964		4,94%
3.020		5,03%

Tabel 4.9 menunjukkan hasil pengukuran penggunaan RAM oleh program pada modul IoT. Dari hasil pengujian didapatkan hasil penggunaan RAM oleh keseluruhan program sebesar 3.020 *Byte* yang berarti masih menyisakan 5,03% dari RAM yang tersedia sebesar 60.000 *Byte*.

4.4 Pengujian Energi

4.4.1 Tujuan Pengujian

Tujuan darai pengujian ini adalah untuk mengetahui dan membandingkan konsumsi energi ketika kondisi *idle* dan ketika program enkripsi dijalankan. Pengujian ini dilakukan dengan mengukur tegangan pada R_{SHUNT} dengan menggunakan *oscilloscope* untuk mencari nilai daya.

4.4.2 Peralatan yang Digunakan

- Modul IoT *Particle Photon*.
- Sensor BME280
- Program IDLE.

- Program enkripsi AES-128 untuk modul IoT.
- Resistor 100 *Ohm.*
- Kabel *jumper* dan capit buaya.
- *Oscilloscope*.
- *Voltmeter*.

4.4.3 Prosedur Pengujian

- Menghubungkan sensor BME280 dan modul IoT *Particle Photon*.
- Menghubungkan pin V_{in} modul IoT dengan sisi negatif R_{SHUNT} dan CH2- *Oscilloscope* menggunakan kabel dan capit buaya.
- Menghubungkan sisi positif baterai dan sisi positif R_{SHUNT} dan CH2+ *Oscilloscope* menggunakan kabel dan capit buaya.
- Meghubungkan pin GND modul IoT dengan kutub negatif V_{Supply} (baterai).
- Mengunggah program *idle* ke modul IoT.
- Mengamati dan mencatat hasil keluaran berupa nilai tegangan ketika *idle* dari *Oscilloscope*.
- Mengunggah progaram enkripsi ke modul IoT.

4.4.4 Hasil Pengujian

Setelah melakukan prosedur pengujian, hasil yang didapatkan adalah berupa tegangan pada R_{SHUNT} atau V_{SHUNT} yang merupakan hasil dari keluaran pada layar *oscilloscope*. Tabel 4.10 merupakan hasil pengukuran yang didapatkan.

Tabel 4.10
Tegangan *Shunt* ketika *idle* dan enkripsi

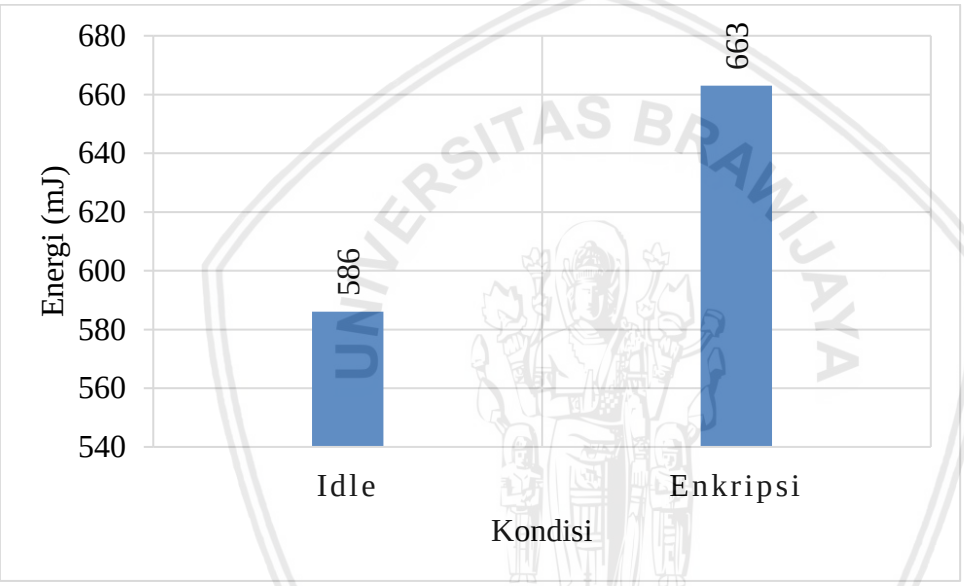
V_{SHUNT} ketika kondisi *idle*	V_{SHUNT} ketika enkripsi
104 mV	118 mV

Pada Tabel 4.10 menunjukkan bahwa terdapat selisih tegangan ketika kondisi *idle* dan ketika melakukan proses enkripsi. Tegangan di atas merupakan tegangan rata-rata dalam rentang waktu pengukuran selama 12 milidetik. Sehingga jika dilakukan perhitungan daya dan energi dengan menggunakan Persamaan 3.4 dan Persamaan 3.5 akan didapatkan hasil seperti Tabel 4.11 sebagai berikut:

Tabel 4.11
Hasil Perhitungan arus, daya dan energi

Kondisi	V_{Supply} (mV)	V_{SHUNT} (mV)	*I* (mA)	*P* (mW)	*E* (mJ)
Idle	4800	104	10,4	49	586
Enkripsi	4800	118	11,8	55	663

Berdasarkan Tabel 4.11, didapatkan hasil perhitungan daya pada kondisi *idle* yaitu sebesar 49 mW dan 55 mW ketika melakukan enkripsi. Didapatkan pula hasil perhitungan energi pada kondisi *idle* sebesar 586 mJ dan 663 mJ ketika melakukan enkripsi. Selisih energi saat kondisi *idle* dan enkripsi dapat dilihat pula pada grafik dalam Gambar 4.3.



Gambar 4.3 Perbandingan energi saat *idle* dan enkripsi

4.5 Pengujian Pengiriman *Ciphertext* Melalui UDP dan TCP/IP

4.5.1 Tujuan Pengujian

Pengujian ini bertujuan untuk mengetahui keberhasilan pengiriman *ciphertext* melalui UDP dan TCP/IP. Keberhasilan pengiriman ditentukan oleh kesesuaian *ciphertext* yang dikirim oleh modul IoT dengan *ciphertext* yang diterima oleh perangkat penerima.

4.5.2 Peralatan yang Digunakan

- Modul IoT *Particle Photon*.
- Sensor BME280
- Kabel USB *Micro* B to USB *Type* A.
- Program enkripsi AES-128 untuk modul IoT.
- Program dekripsi AES-128.

- *Terminal*.
- *Wireshark*.

4.5.3 Prosedur Pengujian

- Menghubungkan sensor BME280 dan modul IoT *Particle Photon*.
- Menghubungkan modul IoT *Particle Photon* ke komputer melalui USB.
- Mengunggah program ke modul IoT.
- Membuka *Wireshark* untuk melakukan *sniffing* dan analisa paket data (*ciphertext*) yang dikirimkan.
- Membuka *terminal* untuk me-*run* program penerima.
- Mengamati dan mencatat hasil keluaran berupa *ciphertext* yang dikirimkan melalui UDP dan TCP/IP.

4.5.4 Hasil Pengujian

Setelah melakukan prosedur pengujian didapatkan hasil *ciphertext* lewat UDP dan TCP/IP yang terbaca oleh *Wireshark* serta keluaran dari program penerima yang dijalankan pada perangkat penerima. Hasil pengujian dapat dilihat pada Tabel 4.12 dan 4.13 sebagai berikut:

Tabel 4.12
Ciphertext lewat UDP yang terbaca *Wireshark*

Ciphertext lewat UDP yang terbaca *Wireshark*	*Ciphertext* terkirim pada program penerima UDP
abbe5dff2ac1bc04880e4ff0de7f04ca	abbe5dff2ac1bc04880e4ff0de7f04ca
ea960505d9665de8dde1bcfc8a587673	ea960505d9665de8dde1bcfc8a587673
34ef50f6c2ea1d7817e446af362a009f	34ef50f6c2ea1d7817e446af362a009f
9e0fff59c9224f230ad37313dd325045	9e0fff59c9224f230ad37313dd325045
ba6b2e50371b2cf07d400c9f73089e66	ba6b2e50371b2cf07d400c9f73089e66
79a3969759191871de140f9d76babab3	79a3969759191871de140f9d76babab3
e5dea98e2a663c3975c65a27926ebdbc	e5dea98e2a663c3975c65a27926ebdbc
ffb10339d0fa5254205ef8c1805b6723	ffb10339d0fa5254205ef8c1805b6723
f4dedd56dd7d5adef5130de241f920d0	f4dedd56dd7d5adef5130de241f920d0
5b75328751860b42d46df7c63465f5e5	5b75328751860b42d46df7c63465f5e5

Tabel 4.13
Ciphertext lewat TCP/IP yang terbaca *Wireshark*

Ciphertext lewat TCP/IP yang terbaca *Wireshark*	*Ciphertext* terkirim pada program penerima TCP/IP
785eb2d63feb964b3cecd5dd00c04a84	785eb2d63feb964b3cecd5dd00c04a84
a230a042c313133ccbcfbb09bfbcdfbb	a230a042c313133ccbcfbb09bfbcdfbb
f2ad2f8cb0a1d52f1ecc9aa77090afa2	f2ad2f8cb0a1d52f1ecc9aa77090afa2
29089ddfb1c76a7600898d444c40d868	29089ddfb1c76a7600898d444c40d868
83cdc86f43d2c99c8cfcc767afff3337	83cdc86f43d2c99c8cfcc767afff3337
3a8cb1c7689565b4518410c19fdeca8e	3a8cb1c7689565b4518410c19fdeca8e
aa44eda35615de00c94a92061691a57b	aa44eda35615de00c94a92061691a57b
2c5d4c85a73f915dcd031d50dff45baa	2c5d4c85a73f915dcd031d50dff45baa
430bc258215c8a3c53e5ac35fe22d229	430bc258215c8a3c53e5ac35fe22d229
f6d9c34b5df0be202b0813b582fad7e7	f6d9c34b5df0be202b0813b582fad7e7

Tabel 4.12 dan Tabel 4.13 menunjukkan bahwa pada setiap pengiriman *ciphertext* berhasil dilewatkan melalui UDP dan TCP/IP. Setiap paket yang terkirim ke penerima dan terbaca oleh *Wireshark* berbentuk *ciphertext* dan menghasilkan keluaran yang sama dengan program penerima, sehingga data sensor yang dikirimkan dari modul IoT ke perangkat penerima berhasil terenkripsi.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari kajian ini diuraikan sebagai berikut:

1. Penerapan program enkripsi AES-128 pada modul IoT mampu memberikan hasil enkripsi berupa *ciphertext* yang dapat dikembalikan menjadi pesan asli atau *plaintext*. Hasil pengujian juga menunjukkan bahwa *ciphertext* memiliki nilai *Avalanche Effect* di atas 50%.
2. Pada pengujian performa (waktu dan *throughput*) didapatkan waktu rata-rata untuk melakukan proses enkripsi selama 381 mikrodetik. Didapatkan pula nilai *throughput* rata-rata sebesar 315120 bit/detik.
3. Dari hasil pengujian penggunaan memori didapatkan memori *flash* yang terpakai hanya sebesar 12,81% dari 125.000 *Byte* memori *flash* yang tersedia. Didapatkan pula penggunaan RAM oleh keseluruhan program hanya memakai RAM sebesar 5,03% dari 60.000 *Byte* RAM yang tersedia.
4. Konsumsi energi yang dihasilkan dari adanya penerapan AES-128 pada modul IoT memiliki selisih sebesar 77 mJ terhadap konsumsi energi tanpa enkripsi atau dalam kondisi *idle*.
5. Pengiriman *ciphertext* melalui UDP dan TCP/IP berhasil dilakukan.

5.2 Saran

Berdasarkan hasil kajian, penulis mengharapkan adanya perbaikan pada algoritma enkripsi seperti penggunaan mode AES yang lebih aman seperti CBC dan GCM. Namun penggunaan mode GCM lebih disarankan karena mode ini memiliki sifat autentikasi serta mode ini di desain paralel seperti mode ECB untuk enkripsi cepat yang baik diterapkan pada perangkat dengan sumber daya terbatas seperti modul IoT berbasis *embedded system*. Selain itu optimalisasi pada program enkripsi juga diperlukan agar didapatkan program enkripsi dengan performa yang lebih baik untuk diterapkan pada modul IoT berbasis *embedded system*.

DAFTAR PUSTAKA

Abdelgawad, A., 2014. *Distributed data fusion algorithm for Wireless Sensor Network.* Miami, IEEE.

Ada, L., 2017. *Adafruit BME280 Humidity + Barometric Pressure + TemperatureSensor Breakout.* [Online] Available at: https://www.marutsu.co.jp/contents/shop/marutsu/ds/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout.pdf [Diakses 15 Mei 2019].

Allen, R., 2017. *Network Sniffers: The Complete Guide.* [Online] Available at: https://networksecuritytools.com/network-sniffers-guide/ [Diakses 12 Agustus 2019].

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

Blazhevski, D., Bozhinovski, A., Stojchevska, B. & Pachovski, V., 2013. *Modes of Operation of The AES Algorithm.* Bitola, Faculty of Computer Science and Engineering, Skopje, Macedonia, pp. 212-216.

Boneh, D. & Shoup, V., 2015. *A Graduate Course in Applied Cryptograph.* [Online] Available at: https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf [Diakses 17 Mei 2019].

Bosch, 2018. *BME280 Combined humidity and pressure sensor.* [Online] Available at: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf [Diakses 15 Mei 2019].

Carroll, E. et al., 2019. *McAfee Labs 2019 Threats Predictions Report.* [Online] Available at: https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/mcafee-labs-2019-threats-predictions/ [Diakses 10 Agustus 2019].

Chakraborty, R. et al., 2011. Triple SV: A Bit Level Symmetric Block Cipher Having High Avalanche Effect. *International Journal of Advanced Computer Science and Applications,* 2(7), pp. 61-68.

Chiu, S.-H., 2018. *Introduction of Linear Algebra,* Houston: University of Houston.

Daemen, J. & Rijmen, V., 2001. *The Design of Rijndael AES - The Advanced Encryption Standard.* Brussels: Springer.

Dobbertin, H., Knudsen, L. & Robshaw, M., 2005. Advanced Encryption Standard - AES. Dalam: H. Dobbertin, V. Rijmen & A. Sowa, penyunt. *4th International Conference, AES 2004 Bonn, Germany, May 10-12, 2004 Revised Selected and Invited Papers.* Berlin: Springer, pp. 2-5.

Donnay, V. J., 2013. *Solar Panels, Energy and Area Under the Curve Teacher Guide,* Bryn Mawr: Bryn Mawr College.

Dworkin, M., 2001. *Recommendation for Block Cipher Modes of Operation,* Washington: National Institute of Standards and Technology (NIST).

Echeverri, C., 2017. *Visualization of the Avalanche,* Mannheim: Faculty for Business Informatics & Business Mathematics, University of Mannheim.

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.

Frustaci, M., Pace, P. & Aloi, G., 2017. *Securing the IoT world: issues and perspectives.* Helsinki, IEEE.

Gatlan, S., 2019. *IoT Attacks Escalating with a 217.5% Increase in Volume.* [Online] Available at: https://www.bleepingcomputer.com/news/security/iot-attacks-escalating-with-a-2175-percent-increase-in-volume/
[Diakses 10 August 2019].

Hadiyuwono, D. & Pambudi, 2011. *User Datagram Protocol (UDP).* [Online] Available at: http://te.ugm.ac.id/~risanuri/v01/wp-content/uploads/2011/02/udp.pdf
[Diakses 17 Mei 2019].

Hart, W., 2017. *Photon Datasheet (V016).* [Online] Available at: https://docs.particle.io/datasheets/wi-fi/photon-datasheet/ [Diakses 15 May 2019].

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

Karlsson, J., 2017. *Internet of Things –Does Particle Photon rely too much on its own cloud solution?*, Linköping: Linköping University.

Kumar, A. & Tiwari, N., 2012. Effective Implementation and Avalanche Effect of AES. *International Journal of Security, Privacy and Trust Management (IJSPTM),* 1(3/4), pp. 31-35.

Kwon, M., 2012. *A Tutorial on Network Latency and its Measurements.* [Online] Available at: https://www.cs.rit.edu/~jmk/papers/kwon.igi14.pdf [Diakses 17 Mei 2019].

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.

Lydia, P., Britt, D. T. & al., e., 2006. *TCP/IP Tutorial and Technical Overview.* 8th penyunt. s.l.:IBM.

Mesander, B., 2014. *Using UDP in Internet-of-Things Devices.* [Online] Available at: https://cardinalpeak.com/blog/using-udp-in-internet-of-things-devices/ [Diakses 17 Mei 2019].

Miller, J., 2018. *Cryptopals challenge 7: Implement AES in ECB mode with Python.* [Online] Available at: https://laconicwolf.com/2018/07/22/cryptopals-challenge-7-implement-aes-in-ecb-mode-with-python/ [Diakses 11 August 2019].

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

NIST, 2001. *Federal Information Processing Standards Publication 197 Announcing the Advanced Encryption Standard (AES),* s.l.: National Institute of Standards and Technology (NIST).

Noura, B., Hadj, Y. W. E., Mohsen, M. & Rached, T., 2012. *A compact 32-Bit AES design for embedded system.* Gammarth, IEEE.

Oluwabukola, O. et al., 2013. A Packet Sniffer (PSniffer) Application for Network Security in Java. *Issues in Informing Science and Information Technology,* Volume 10, pp. 389-400.

Raju, B., Krishna, A. & Mishra, G., 2017. *Implementation of an Efficient Dynamic AES.* Uttar Pradesh, IEEE, pp. 39-43.

Razzaq, M. A., Qureshi, M. A., Gill, S. H. & Ullah, S., 2017. 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.

Rodriguez, E., 2014. *TCP vs. UDP.* [Online] Available at: http://www.skullbox.net/tcpudp.php [Diakses 17 Mei 2019].

Rupam, Verma, A. & Singh, A., 2013. An Approach to Detect Packets Using PacketSniffing. *International Journal of Computer Science & Engineering Survey (IJCSES),* 4(3), pp. 21-33.

Shichao, 2015. *User Datagram Protocol (UDP) and IP Fragmentation.* [Online] Available at: https://notes.shichao.io/tcpv1/ch10/ [Diakses 17 May 2019].

Sukaridhoto, S., 2014. *Buku Jaringan Komputer I.* [Online] Available at: http://dhoto.lecturer.pens.ac.id/publications/book/2014/Dhoto-JaringanKomputer1.pdf [Diakses 21 Juni 2019].

Tenouk, 2019. *LINUX SOCKET PART 16 Advanced TCP/IP-The TCP/IP Protocols Details.* [Online] Available at: https://www.tenouk.com/Module43.html [Diakses 21 Juni 2019].

Tsai, K.-L.et al., 2018. AES-128 Based Secure Low Power Communication for LoRaWAN IoT Environments. *Special Section on Security and Trusted Computing for Indutrial Internet of Things,* Volume 6, pp. 45325-45333.

Ukil, A., Sen, J. & Koilakonda, S., 2014. *Embedded Security for Internet of Things,* Kolkata: IoT Security Lab.

Vashi, S. et al., 2017. *Internet of Things (IoT) - A Vision, Architectural Elements, and Security Issues.* Palladam, IEEE.

Wang, S., 2019. *The performance test on the AES modes.* [Online] Available at: https://www.highgo.ca/2019/08/22/the-performance-test-on-the-aes-modes/
[Diakses 2 October 2019].

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

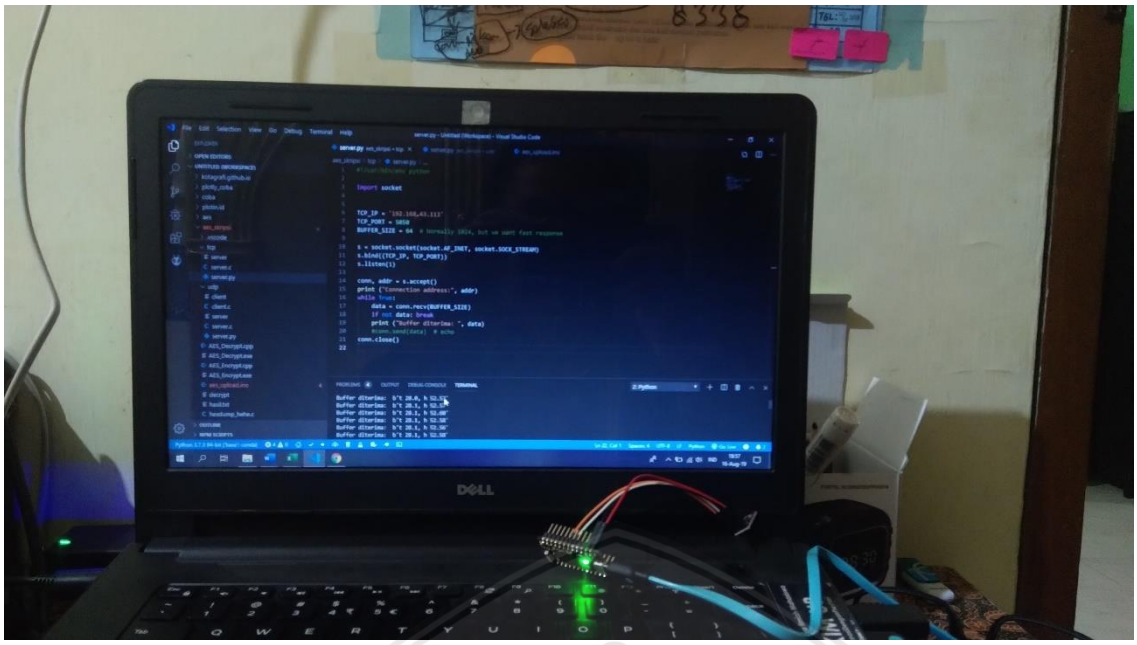
Wei, Z., Yuqing, Z. & Peng, L., 2018. The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved. *IEEE Paper Manuscript,* pp. 1-11.

Witoolkollachit, P., 2015. *The avalanche effect of various hash functions between encrypted raw images versus non-encrypted images: A comparison study* , Bangkok: Department of Information Technology, King Mongkut's University of Technology.

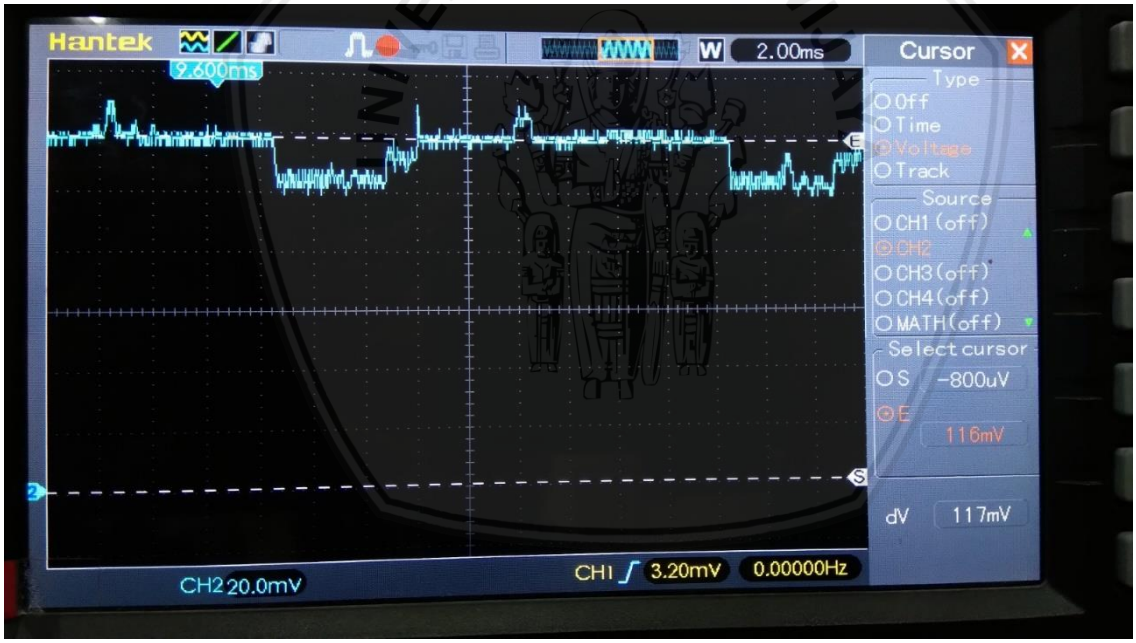
Zhang, F., 2016. *Lab 1: Packet Sniffing and Wireshark,* Detroit: College of Engineering, Wayne State University.

LAMPIRAN 1

DOKUMENTASI ALAT



Gambar 1. Penerapan program enkripsi pada modul IoT



Gambar 2. Pengambilan data menggunakan *oscilloscope*

LAMPIRAN 2

SOURCE CODE (C/C++ DAN PYTHON)

```
/*
 * Project aes
 * Description: Program enkripsi
 * Date: 30 July 2019
 * References of implementation: Nayuki Project, Niyaz PK, Particle Reference
*/

#include <Adafruit_BME280.h>
#include "Adafruit_Sensor.h"
#include "Adafruit_BME280.h"

#define BME_SCK D1
#define BME_MOSI D0

#include<stdio.h>
#include <string.h>

Adafruit_BME280 bme; // I2C BME280
//Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO,  BME_SCK);

double temp;
double humidity;

void setup() {
  Serial.begin(9600);
  Serial.println(F("BME280 test"));

  // if (!bme.begin(0x76)) {
  if (!bme.begin()) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}

#define Nb 4

int Nr=0;

int Nk=0;

unsigned char in[16], out[16], state[4][4];

unsigned char RoundKey[240];

unsigned char Key[32];

int getSBoxValue(int num)
{
      int sbox[256] =   {
```

```
    //0    1   2    3    4   5    6    7    8   9    A    B   C    D    E    F
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76, //0
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0, //1
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8,
0x31, 0x15, //2
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75, //3
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84, //4
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf, //5
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c,
0x9f, 0xa8, //6
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff,
0xf3, 0xd2, //7
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73, //8
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb, //9
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79, //A
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08, //B
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a, //C
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e, //D
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf, //E
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16 }; //F
    return sbox[num];
}

int Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83,
0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc,
```

```
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb  };

void KeyExpansion()
{
    int i,j;
    unsigned char temp[4],k;

    for(i=0;i<Nk;i++)
    {
        RoundKey[i*4]=Key[i*4];
        RoundKey[i*4+1]=Key[i*4+1];
        RoundKey[i*4+2]=Key[i*4+2];
        RoundKey[i*4+3]=Key[i*4+3];
    }

    while (i < (Nb * (Nr+1)))
    {
        for(j=0;j<4;j++)
        {
            temp[j]=RoundKey[(i-1) * 4 + j];
        }
        if (i % Nk == 0)
        {
            {
                k = temp[0];
                temp[0] = temp[1];
                temp[1] = temp[2];
                temp[2] = temp[3];
                temp[3] = k;
            }
```

```
            {
                temp[0]=getSBoxValue(temp[0]);
                temp[1]=getSBoxValue(temp[1]);
                temp[2]=getSBoxValue(temp[2]);
                temp[3]=getSBoxValue(temp[3]);
            }

            temp[0] =  temp[0] ^ Rcon[i/Nk];
        }
        else if (Nk > 6 && i % Nk == 4)
        {
            {
                temp[0]=getSBoxValue(temp[0]);
                temp[1]=getSBoxValue(temp[1]);
                temp[2]=getSBoxValue(temp[2]);
                temp[3]=getSBoxValue(temp[3]);
            }
        }
        RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
        RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
        RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
        RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
        i++;
    }
}

void AddRoundKey(int round)
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
        }
    }
}

void SubBytes()
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[i][j] = getSBoxValue(state[i][j]);

        }
    }
}
```

```
void ShiftRows()
{
	unsigned char temp;

	temp=state[1][0];
	state[1][0]=state[1][1];
	state[1][1]=state[1][2];
	state[1][2]=state[1][3];
	state[1][3]=temp;

	temp=state[2][0];
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	temp=state[3][0];
	state[3][0]=state[3][3];
	state[3][3]=state[3][2];
	state[3][2]=state[3][1];
	state[3][1]=temp;
}

#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

void MixColumns()
{
	int i;
	unsigned char Tmp,Tm,t;
	for(i=0;i<4;i++)
	{
		t=state[0][i];
		Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i] ;
		Tm = state[0][i] ^ state[1][i] ; Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp ;
		Tm = state[1][i] ^ state[2][i] ; Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp ;
		Tm = state[2][i] ^ state[3][i] ; Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp ;
		Tm = state[3][i] ^ t ; Tm = xtime(Tm); state[3][i] ^= Tm ^ Tmp ;
	}
}

void Cipher()
{
	int i,j,round=0;

	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
```

```
            {
                state[j][i] = in[i*4 + j];
            }
        }

        AddRoundKey(0);

        for(round=1;round<Nr;round++)
        {
            SubBytes();
            ShiftRows();
            MixColumns();
            AddRoundKey(round);
        }

        SubBytes();
        ShiftRows();
        AddRoundKey(Nr);

        for(i=0;i<4;i++)
        {
            for(j=0;j<4;j++)
            {
                out[i*4+j]=state[j][i];
            }
        }
}

void loop() {

  int i;

        Nr = 128;

        Nk = Nr / 32;
        Nr = Nk + 6;

  float t = bme.readTemperature(); // *C
  float h = bme.readHumidity(); // %

  String data = "t "+ String::format("%.1f",t)+ ", "+ "h "+ String::format("%.2f",h);

  Serial.printlnf(data);

  char temp[16] = "inikuncinyagan";
  int len = 16;

    for (i = 0; i < len; i++)
    {
       if (i > 0) Serial.printf(" ");
```

```
      Serial.printf("%02X", temp[i]);
    }
    Serial.printlnf("\n");

    for (i = 0; i < len; i++)
    {
       if (i > 0) Serial.printf(" ");
       Serial.printf("%02X", data[i]);
    }
    Serial.printlnf("\n");

      for(i=0;i<Nk*4;i++)
      {
         for (i = 0; i < len; ++i)
         {
           Key[i]=("%02x", temp[i]);
         }
      }

      for(i=0;i<Nb*4;i++)
      {
         for (i = 0; i < len; ++i)
            {
       in[i]=("%02x", data[i]);
            }
      }

      KeyExpansion();

      Cipher();

      for(i=0;i<Nk*4;i++)
      {
            Serial.printf("%02x ",out[i]);
      }
  Serial.printlnf("\n\n");

  delay(1000);
}
```

```
/*
 * Project aes
 * Description: Program dekripsi
 * Date: 30 July 2019
 * References of implementation: Nayuki Project, Niyaz PK, Particle Reference
*/

#include<stdio.h>
#include<string.h>

#define Nb 4

int Nr=0;

int Nk=0;

unsigned char in[16], out[16], state[4][4];

unsigned char RoundKey[240];

unsigned char Key[32];

int getSBoxInvert(int num)
{
int rsbox[256] =
{ 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3,
0xd7, 0xfb
, 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde,
0xe9, 0xcb
, 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,
0xc3, 0x4e
, 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b,
0xd1, 0x25
, 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65,
0xb6, 0x92
, 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,
0x9d, 0x84
, 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3,
0x45, 0x06
, 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13,
0x8a, 0x6b
, 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4,
0xe6, 0x73
, 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75,
0xdf, 0x6e
, 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18,
0xbe, 0x1b
, 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,
0x5a, 0xf4
```

```
, 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xec, 0x5f
, 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9,
0x9c, 0xef
, 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,
0x99, 0x61
, 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0c, 0x7d };

return rsbox[num];
}

int getSBoxValue(int num)
{
    int sbox[256] =   {
    //0    1   2    3    4   5    6    7    8   9    A    B   C    D    E    F
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8,
0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c,
0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff,
0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16 };
    return sbox[num];
}
```

```
int Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83,
0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb  };

void KeyExpansion()
{
    int i,j;
    unsigned char temp[4],k;

    for(i=0;i<Nk;i++)
    {
        RoundKey[i*4]=Key[i*4];
        RoundKey[i*4+1]=Key[i*4+1];
        RoundKey[i*4+2]=Key[i*4+2];
        RoundKey[i*4+3]=Key[i*4+3];
    }

    while (i < (Nb * (Nr+1)))
    {
```

```
            for(j=0;j<4;j++)
            {
                temp[j]=RoundKey[(i-1) * 4 + j];
            }
            if (i % Nk == 0)
            {
                {
                    k = temp[0];
                    temp[0] = temp[1];
                    temp[1] = temp[2];
                    temp[2] = temp[3];
                    temp[3] = k;
                }
                {
                    temp[0]=getSBoxValue(temp[0]);
                    temp[1]=getSBoxValue(temp[1]);
                    temp[2]=getSBoxValue(temp[2]);
                    temp[3]=getSBoxValue(temp[3]);
                }

                temp[0] =  temp[0] ^ Rcon[i/Nk];
            }
            else if (Nk > 6 && i % Nk == 4)
            {
                {
                    temp[0]=getSBoxValue(temp[0]);
                    temp[1]=getSBoxValue(temp[1]);
                    temp[2]=getSBoxValue(temp[2]);
                    temp[3]=getSBoxValue(temp[3]);
                }
            }
            RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
            RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
            RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
            RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
            i++;
        }
    }

    void AddRoundKey(int round)
    {
        int i,j;
        for(i=0;i<4;i++)
        {
            for(j=0;j<4;j++)
            {
                state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
            }
        }
    }
```

```
void InvSubBytes()
{
	int i,j;
	for(i=0;i<4;i++)
	{
		for(j=0;j<4;j++)
		{
			state[i][j] = getSBoxInvert(state[i][j]);

		}
	}
}

void InvShiftRows()
{
	unsigned char temp;

	temp=state[1][3];
	state[1][3]=state[1][2];
	state[1][2]=state[1][1];
	state[1][1]=state[1][0];
	state[1][0]=temp;

	temp=state[2][0];
	state[2][0]=state[2][2];
	state[2][2]=temp;

	temp=state[2][1];
	state[2][1]=state[2][3];
	state[2][3]=temp;

	temp=state[3][0];
	state[3][0]=state[3][1];
	state[3][1]=state[3][2];
	state[3][2]=state[3][3];
	state[3][3]=temp;
}

#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

#define Multiply(x,y) (((y & 1) * x) ^ ((y>>1 & 1) * xtime(x)) ^ ((y>>2 & 1) *
xtime(xtime(x))) ^ ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^ ((y>>4 & 1) *
xtime(xtime(xtime(xtime(x))))))

void InvMixColumns()
{
	int i;
	unsigned char a,b,c,d;
	for(i=0;i<4;i++)
```

```
    {

        a = state[0][i];
        b = state[1][i];
        c = state[2][i];
        d = state[3][i];


        state[0][i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^
Multiply(d, 0x09);
        state[1][i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^
Multiply(d, 0x0d);
        state[2][i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^
Multiply(d, 0x0b);
        state[3][i] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^
Multiply(d, 0x0e);
    }
}

void InvCipher()
{
    int i,j,round=0;

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            state[j][i] = in[i*4 + j];
        }
    }

    AddRoundKey(Nr);

    for(round=Nr-1;round>0;round--)
    {
        InvShiftRows();
        InvSubBytes();
        AddRoundKey(round);
        InvMixColumns();
    }

    InvShiftRows();
    InvSubBytes();
    AddRoundKey(0);

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            out[i*4+j]=state[j][i];
```

```
            }
        }
}
int main()
{
        int i;
        int j;

        Nr = 128;

        Nk = Nr / 32;
        Nr = Nk + 6;

        printf("Enter the Key in hexadecimal: ");
        for(i=0;i<Nk*4;i++)
        {
            scanf("%02x",&Key[i]);
        }

        printf("Enter the CipherText in hexadecimal: ");
        for(i=0;i<Nb*4;i++)
        {
            scanf("%02x",&in[i]);
        }

        KeyExpansion();

        InvCipher();

        char enc[64];
        printf("\nText after decryption:\n");
        printf("HEX: ");
        for(i=0;i<Nb*4;i++)
        {
            printf("%02x ",out[i]);
            snprintf(enc, sizeof(enc),
"%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%
02x", out[0], out[1], out[2], out[3], out[4], out[5], out[6], out[7], out[8], out[9], out[10],
out[11], out[12], out[13], out[14], out[15]);
        }
        printf("\n");

        char string[64];
        int len = strlen(enc);
        for (int i = 0, j = 0; j < len; ++i, j += 2) {
        int val[1];
    sscanf(enc + j, "%02x", val);
    string[i] = val[0];
    string[i + 1] = '\0';
        }
```

```
    printf("\n");
    printf("Plaintext: %s", string);
    return 0;
}
```

```
#Program penerima UDP

import socket

UDP_IP = "192.168.43.113"
UDP_PORT = 1337

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

while True:
   data, addr = sock.recvfrom(33)
   print ("Buffer diterima: ", data)
```

```
#Program penerima TCP/IP

import socket

TCP_IP = '192.168.43.113'
TCP_PORT = 5050
BUFFER_SIZE = 64

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print ("Connection address:", addr)
while True:
   data = conn.recv(BUFFER_SIZE)
   if not data: break
   print ("Buffer diterima: ", data)
   #conn.send(data)  # echo
conn.close()
```

LAMPIRAN 3

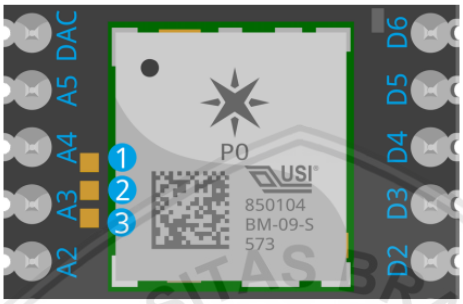
DATASHEET

Datasheet Particle Photon

EXTERNAL COEXISTENCE INTERFACE

Note: Currently this interface is unsupported. If you need this feature for a product, please contact Particle.

The Photon supports coexistence with Bluetooth and other external radios via the three gold pads on the top side of the PCB near pin A3. These pads are 0.035" square, spaced 0.049" apart. This spacing supports the possibility of tacking on a small 1.25mm - 1.27mm pitch 3-pin male header to make it somewhat easier to interface with.



When two radios occupying the same frequency band are used in the same system, such as Wi-Fi and Bluetooth, a coexistence interface can be used to coordinate transmit activity, to ensure optimal performance by arbitrating conflicts between the two radios.

Pad #	PØ Pin Name	PØ Pin #	I/O	Description
1	BTCX_RF_ACTIVE	9	I	Signals Bluetooth is active
2	BTCX_STATUS	10	I	Signals Bluetooth priority status and TX/RX direction
3	BTCX_TXCONF	11	O	Output giving Bluetooth permission to TX

When these pads are programmed to be used as a Bluetooth coexistence interface, they're set as high impedance on power up and reset.

Functional description

OVERVIEW

Particle's Internet of Things hardware development kit, the Photon, provides everything you need to build a connected product. Particle combines a powerful ARM Cortex M3 micro-controller with a Broadcom Wi-Fi chip in a tiny thumbnail-sized module called the PØ (P-zero).

To get you started quickly, Particle adds a rock solid 3.3VDC SMPS power supply, RF and user interface components to the PØ on a small single-sided PCB called the Photon. The design is open source, so when you're ready to integrate the Photon into your product, you can.

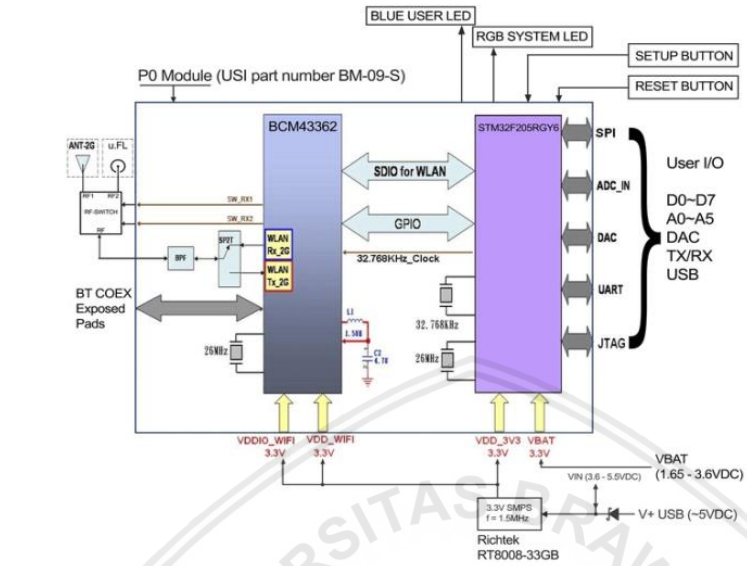
The Photon comes in two physical forms: with headers and without. Prototyping is easy with headers as the Photon plugs directly into standard breadboards and perfboards, and may also be mounted with 0.1" pitch female headers on a PCB. To minimize space required, the Photon form factor without headers has castellated edges. These make it possible to surface mount the Photon directly onto your PCB.

FEATURES

- Particle PØ Wi-Fi module
  - Broadcom BCM43362 Wi-Fi chip
  - 802.11b/g/n Wi-Fi
  - STM32F205RGY6 120Mhz ARM Cortex M3
  - 1MB flash, 128KB RAM
- On-board RGB status LED (ext. drive provided)
- 18 Mixed-signal GPIO and advanced peripherals
- Open source design
- Real-time operating system (FreeRTOS)
- Soft AP setup
- FCC, CE and IC certified

Interfaces

BLOCK DIAGRAM



FCC APPROVED ANTENNAS

Antenna Type	Manufacturer	MFG. Part #	Gain
Dipole antenna	LumenRadio	104-1001	2.15dBi
Chip antenna	Advanced Ceramic X	AT7020-E3R0HBA	1.3dBi

PERIPHERALS AND GPIO

The Photon has ton of capability in a small footprint, with analog, digital and communication interfaces.

Peripheral Type	Qty	Input(I) / Output(O)	FT[1] / 3V3[2]
Digital	18	I/O	FT/3V3
Analog (ADC)	8	I	3V3
Analog (DAC)	2	O	3V3
SPI	2	I/O	3V3
I2S	1	I/O	3V3
I2C	1	I/O	FT
CAN	1	I/O	3V3[4]
USB	1	I/O	3V3
PWM	9[3]	O	3V3

Notes:

[1] FT = 5.0V tolerant pins. All pins except A3 and DAC are 5V tolerant (when not in analog mode). If used as a 5V input the pull-up/pull-down resistor must be disabled.

[2] 3V3 = 3.3V max pins.

[3] PWM is available on D0, D1, D2, D3, A4, A5, WKP, RX, TX with a caveat: PWM timer peripheral is duplicated on two pins (A5/D2) and (A4/D3) for 7 total independent PWM outputs. For example: PWM may be used on A5 while D2 is used as a GPIO, or D2 as a PWM while A5 is used as an analog input. However A5 and D2 cannot be used as independently controlled PWM outputs at the same time.

[4] Technically these pins are 5.0V tolerant, but since you wouldn't operate them with a 5.0V transceiver it's proper to classify them as 3.3V.

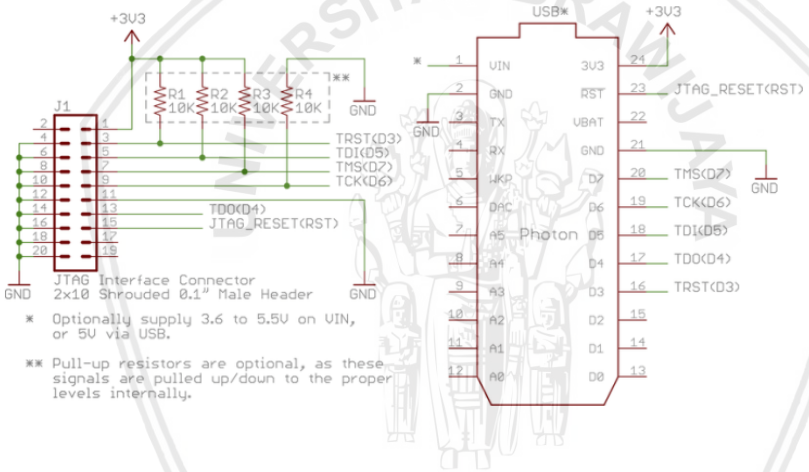
JTAG AND SWD

Pin D3 through D7 are JTAG interface pins. These can be used to reprogram your Photon bootloader or user firmware image with standard JTAG tools such as the ST-Link v2, J-Link, R-Link, OLIMEX ARM-USB-TINI-H, and also the FTDI-based Particle JTAG Programmer. If you are short on available pins, you may also use SWD mode which requires less connections.

Photon Pin	JTAG	SWD	STM32F205RGY6 Pin	PØ Pin #	PØ Pin Name	Default Internal[1]
D7	JTAG_TMS	SWD/SWDIO	PA13	44	MICRO_JTAG_TMS	~40k pull-up
D6	JTAG_TCK	CLK/SWCLK	PA14	40	MICRO_JTAG_TCK	~40k pull-down
D5	JTAG_TDI		PA15	43	MICRO_JTAG_TDI	~40k pull-up
D4	JTAG_TDO		PB3	41	MICRO_JTAG_TDO	Floating
D3	JTAG_TRST		PB4	42	MICRO_JTAG_TRSTN	~40k pull-up
3V3	Power	Power				
GND	Ground	Ground				
RST	Reset	Reset				

Notes: [1] Default state after reset for a short period of time before these pins are restored to GPIO (if JTAG debugging is not required, i.e. `USE_SWD_JTAG=y` is not specified on the command line.)

A standard 20-pin 0.1" shrouded male JTAG interface connector should be wired as follows:



Memory Map

STM32F205RGY6 FLASH LAYOUT OVERVIEW

- Bootloader (16 KB)
- DCT1 (16 KB), stores Wi-Fi credentials, keys, mfg info, system flags, etc..
- DCT2 (16 KB), swap area for DCT1
- EEPROM emulation bank 1 (16 KB)
- EEPROM emulation bank 2 (64 KB)
- Device OS (512 KB) [256 KB Wi-Fi/comms + 256 KB hal/platform/services]
- Factory backup, OTA backup and user application (384 KB) [3 x 128 KB]

DCT LAYOUT

The DCT area of flash memory has been mapped to a separate DFU media device so that we can incrementally update the application data. This allows one item (say, server public key) to be updated without erasing the other items.

DCT layout in `release/stable` found here in firmware.

Region	Offset	Size
system flags	0	32
version	32	2
device private key	34	1216
device public key	1250	384
ip config	1634	120
feature flags	1754	4
country code	1758	4
claim code	1762	63
claimed	1825	1
ssid prefix	1826	26
device code	1852	6
version string	1858	32
dns resolve	1890	128
reserved1	2018	64
server public key	2082	768
padding	2850	2
flash modules	2852	100
product store	2952	24
antenna selection	2976	1
cloud transport	2977	1
alt device public key	2978	128
alt device private key	3106	192
alt server public key	3298	192
alt server address	3490	128
device id	3618	12
radio flags	3630	1
mode button mirror	3631	32

led mirror	3663	96
led theme	3759	64
reserved2	3823	435

Note: Writing 0xFF to offset 34 (DEFAULT) or 3106 (ALTERNATE) will cause the device to re-generate a new private key on the next boot. Alternate keys are currently unsupported on the Photon but are used on the Electron as UDP/ECC keys. You should not need to use this feature unless your keys are corrupted.

```
// Regenerate Default Keys
echo -en "\xFF" > fillbyte && dfu-util -d 2b04:d006 -a 1 -s 34 -D fillbyte
// Regenerate Alternate Keys
echo -en "\xFF" > fillbyte && dfu-util -d 2b04:d006 -a 1 -s 3106 -D
fillbyte
```

MEMORY MAP (COMMON)

Region	Start Address	End Address	Size
Bootloader	0x8000000	0x8004000	16 KB
DCT1	0x8004000	0x8008000	16 KB
DCT2	0x8008000	0x800C000	16 KB
EEPROM1	0x800C000	0x8010000	16 KB
EEPROM2	0x8010000	0x8020000	64 KB

MEMORY MAP (MODULAR FIRMWARE - DEFAULT)

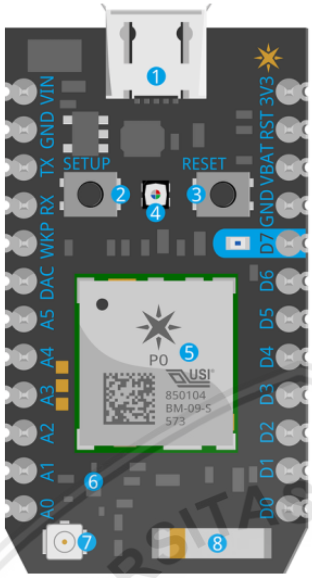
Region	Start Address	End Address	Size
System Part 1	0x8020000	0x8060000	256 KB
System Part 2	0x8060000	0x80A0000	256 KB
User Part	0x80A0000	0x80C0000	128 KB
OTA Backup	0x80C0000	0x80E0000	128 KB
Factory Backup	0x80E0000	0x8100000	128 KB

MEMORY MAP (MONOLITHIC FIRMWARE - OPTIONAL)

Region	Start Address	End Address	Size
Firmware	0x8020000	0x8080000	384 KB
Factory Reset	0x8080000	0x80E0000	384 KB
Unused (factory reset modular)	0x80E0000	0x8100000	128 KB

Pin and button definition

PIN MARKINGS



1. USB
2. SETUP BUTTON
3. RESET BUTTON
4. RGB LED
5. P0 Wi-Fi MODULE
6. RF SWITCH
7. u.FL ANT CONN.
8. CHIP ANTENNA

PIN DESCRIPTION

Pin	Description
VIN	This pin can be used as an input or output. As an input, supply 3.6 to 5.5VDC to power the Photon. When the Photon is powered via the USB port, this pin will output a voltage of approximately 4.8VDC due to a reverse polarity protection series Schottky diode between VUSB and VIN. When used as an output, the max load on VIN is 1A.
RST	Active-low reset input. On-board circuitry contains a 1k ohm pull-up resistor between RST and 3V3, and 0.1uF capacitor between RST and GND.
VBAT	Supply to the internal RTC, backup registers and SRAM when 3V3 is not present (1.65 to 3.6VDC).
3V3	This pin is the output of the on-board regulator and is internally connected to the VDD of the Wi-Fi module. When powering the Photon via VIN or the USB port, this pin will output a voltage of 3.3VDC. This pin can also be used to power the Photon directly (max input 3.3VDC). When used as an output, the max load on 3V3 is 100mA. NOTE: When powering the Photon via this pin, ensure power is disconnected from VIN and USB.
RX	Primarily used as UART RX, but can also be used as a digital GPIO or PWM[2].
TX	Primarily used as UART TX, but can also be used as a digital GPIO or PWM[2].
WKP	Active-high wakeup pin, wakes the module from sleep/standby modes. When not used as a WAKEUP, this pin can also be used as a digital GPIO, ADC input or PWM[2]. Can be referred to as `A7` when used as an ADC.
DAC	12-bit Digital-to-Analog (D/A) output (0-4095), referred to as `DAC` or `DAC1` in software. Can also be used as a digital GPIO or ADC. Can be referred to as `A6` when used as an ADC. A3 is a second DAC output used as `DAC2` in software.
A0~A7	12-bit Analog-to-Digital (A/D) inputs (0-4095), and also digital GPIOs. `A6` and `A7` are code convenience mappings, which means pins are not actually labeled as such but you may use code like `analogRead(A7)`. `A6` maps to the DAC pin and `A7` maps to the WKP pin. A4,A5,A7 may also be used as a PWM[2] output.
D0~D7	Digital only GPIO pins. D0~D3 may also be used as a PWM[2] output.

Notes: [1] In addition to the 24 pins around the outside of the Photon, there are 7 pads on the bottom the Photon PCB that can be used to connect to extra signals: RGB LED outputs, SETUP button, SMPS enable line and USB D+/D-. Photon Pins #25-31 are described in the Pin out diagrams. Also refer to the Recommended PCB land pattern photon without headers section for their location on the bottom of the Photon.

[2] PWM is available on D0, D1, D2, D3, A4, A5, WKP, RX, TX with a caveat: PWM timer peripheral is duplicated on two pins (A5/D2) and (A4/D3) for 7 total independent PWM outputs. For example: PWM may be used on A5 while D2 is used as a GPIO, or D2 as a PWM while A5 is used as an analog input. However A5 and D2 cannot be used as independently controlled PWM outputs at the same time.

Technical specification

ABSOLUTE MAXIMUM RATINGS

Parameter	Symbol	Min	Typ	Max	Unit
Supply Input Voltage	$V_{VIN-MAX}$			+6.5	V
Supply Output Current	$I_{VIN-MAX-L}$			1	A
Supply Output Current	$I_{3V3-MAX-L}$			100	mA
Storage Temperature	T_{stg}	-40		+85	°C
Enable Voltage	V_{EN}			V_{VIN}+0.6	V
ESD Susceptibility HBM (Human Body Mode)	V_{ESD}			2	kV

RECOMMENDED OPERATING CONDITIONS

Parameter	Symbol	Min	Typ	Max	Unit
Supply Input Voltage	V_{VIN}	+3.6		+5.5	V
Supply Input Voltage	V_{3V3}	+3.0	+3.3	+3.6	V
Supply Output Voltage	V_{VIN}		+4.8		V
Supply Output Voltage	V_{3V3}		+3.3		V
Supply Input Voltage	V_{VBAT}	+1.65		+3.6	V
Supply Input Current (VBAT)	I_{VBAT}			19	uA
Operating Current (Wi-Fi on)	$I_{VIN\ avg}$		80	100	mA
Operating Current (Wi-Fi on)	$I_{VIN\ pk}$	235[1]		430[1]	mA
Operating Current (Wi-Fi on, w/powersave)	$I_{VIN\ avg}$		18	100[2]	mA
Operating Current (Wi-Fi off)	$I_{VIN\ avg}$		30	40	mA
Sleep Current (5V @ VIN)	I_{Qs}		1	2	mA
Deep Sleep Current (5V @ VIN)	I_{Qds}		80	100	uA
Operating Temperature	T_{op}	-20		+60	°C
Humidity Range Non condensing, relative humidity				95	%

Notes:

[1] These numbers represent the extreme range of short peak current bursts when transmitting and receiving in 802.11b/g/n modes at different power levels. Average TX current consumption in will be 80-100mA.

[2] These are very short average current bursts when transmitting and receiving. On average if minimizing frequency of TX/RX events, current consumption in powersave mode will be 18mA

WI-FI SPECIFICATIONS

Feature	Description
WLAN Standards	IEEE 802 11b/g/n
Antenna Port	Single Antenna
Frequency Band	2.412GHz -- 2.462GHz (United States of America and Canada)
	2.412GHz -- 2.472GHz (EU/Japan)
Sub Channels	1 -- 11 (United States of America and Canada)
	1 -- 13 (EU/Japan)

Modulation	DSSS, CCK, OFDM, BPSK, QPSK, 16QAM, 64QAM

PØ module Wi-Fi output power		Typ.	Tol.	Unit
RF Average Output Power, 802.11b CCK Mode	1M	Avail. upon request	+/- 1.5	dBm
	11M	-	+/- 1.5	dBm
RF Average Output Power, 802.11g OFDM Mode	6M	-	+/- 1.5	dBm
	54M	-	+/- 1.5	dBm
RF Average Output Power, 802.11n OFDM Mode	MCS0	-	+/- 1.5	dBm
	MCS7	-	+/- 1.5	dBm

I/O CHARACTERISTICS

These specifications are based on the STM32F205RGY6 datasheet, with reference to Photon pin nomenclature.

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Standard I/O input low level voltage	V_{IL}		-0.3		0.28*(V_{3V3}-2)+0.8	V
I/O FT[1] input low level voltage	V_{IL}		-0.3		0.32*(V_{3V3}-2)+0.75	V
Standard I/O input high level voltage	V_{IH}		0.41*(V_{3V3}-2)+1.3		V_{3V3}+0.3	V
I/O FT[1] input high level voltage	V_{IH}	V_{3V3} > 2V	0.42*(V_{3V3}-2)+1		5.5	V
	V_{IH}	V_{3V3} ≤ 2V	0.42*(V_{3V3}-2)+1		5.2	V
Standard I/O Schmitt trigger voltage hysteresis[2]	V_{hys}		200			mV
I/O FT Schmitt trigger voltage hysteresis[2]	V_{hys}		5% V_{3V3}[3]			mV
Input/Output current max	I_{io}				±25	mA
Input/Output current total	$I_{io\ total}$				±120	mA
Input leakage current[4]	I_{lkg}	GND ≤ V_{io} ≤ V_{3V3} GPIOs			±1	µA
Input leakage current[4]	I_{lkg}	R_{PU}	V_{io} = 5V, I/O FT		3	µA
Weak pull-up equivalent resistor[5]	R_{PU}	V_{io} = GND	30	40	50	k Ω
Weak pull-down equivalent resistor[5]	R_{PD}	V_{io} = V_{3V3}	30	40	50	k Ω
I/O pin capacitance	C_{IO}			5		pF
DAC output voltage (buffers enabled by default)	V_{DAC}		0.2		V_{3V3}-0.2	V
DAC output resistive load (buffers enabled by default)	R_{DAC}		5			k Ω
DAC output capacitive load (buffers enabled by default)	C_{DAC}				50	pF

Notes:

[1] FT = Five-volt tolerant. In order to sustain a voltage higher than V_{3V3}+0.3 the internal pull-up/pull-down resistors must be disabled.

[2] Hysteresis voltage between Schmitt trigger switching levels. Based on characterization, not tested in production.

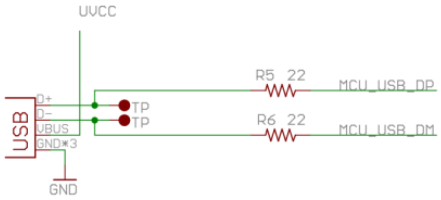
[3] With a minimum of 100mV.

[4] Leakage could be higher than max. if negative current is injected on adjacent pins.

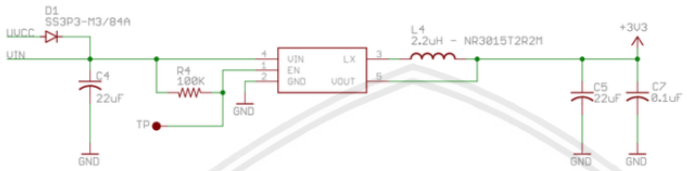
[5] Pull-up and pull-down resistors are designed with a true resistance in series with switchable PMOS/NMOS. This PMOS/NMOS contribution to the series resistance is minimum (~10% order).

Schematic

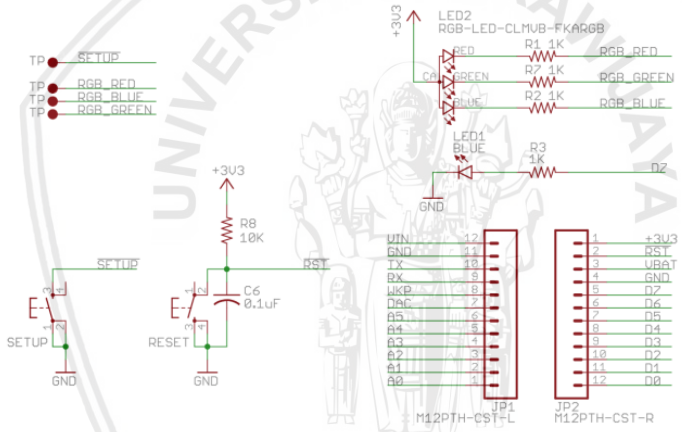
SCHEMATIC - USB



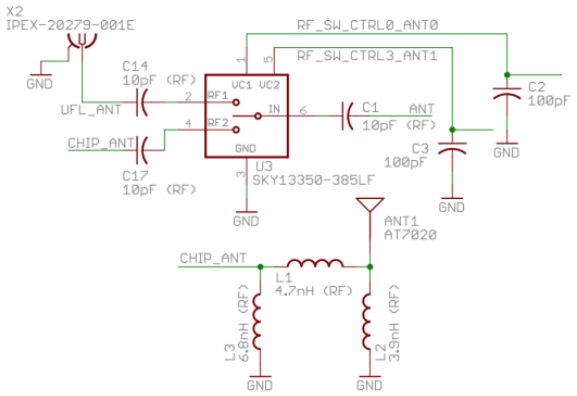
SCHEMATIC - POWER



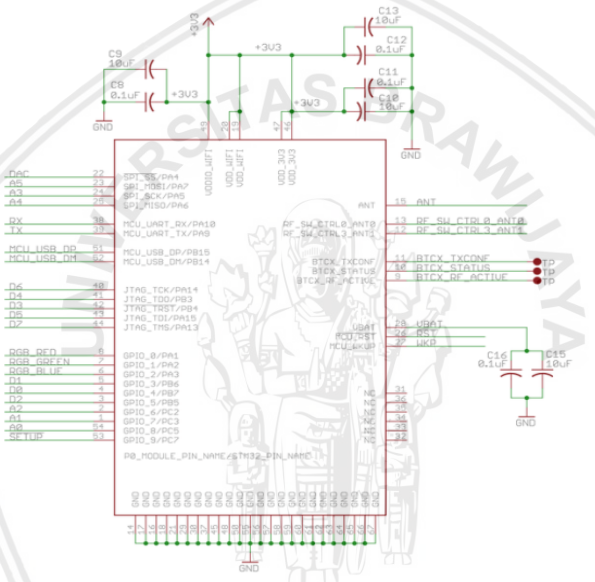
SCHEMATIC - USER I/O



SCHEMATIC - RF



SCHEMATIC - PØ WI-FI MODULE



Datasheet BME280

1. Specification

If not stated otherwise,

- All values are valid over the full voltage range
- All minimum/maximum values are given for the full accuracy temperature range
- Minimum/maximum values of drifts, offsets and temperature coefficients are ±3σ values over lifetime
- Typical values of currents and state machine timings are determined at 25 °C
- Minimum/maximum values of currents are determined using corner lots over complete temperature range
- Minimum/maximum values of state machine timings are determined using corner lots over 0...+65 °C temperature range

The specification tables are split into humidity, pressure, and temperature part of BME280.

1.1 General electrical specification

Table 1: Electrical parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Supply Voltage Internal Domains	V_{DD}	ripple max. 50 mVpp	1.71	1.8	3.6	V
Supply Voltage I/O Domain	V_{DDIO}		1.2	1.8	3.6	V
Sleep current	I_{DDSL}			0.1	0.3	µA
Standby current (inactive period of normal mode)	I_{DDSB}			0.2	0.5	µA
Current during humidity measurement	I_{DDH}	Max value at 85 °C		340		µA
Current during pressure measurement	I_{DDP}	Max value at -40 °C		714		µA
Current during temperature measurement	I_{DDT}	Max value at 85 °C		350		µA
Start-up time	$t_{startup}$	Time to first communication after both V_{DD} > 1.58 V and V_{DDIO} > 0.65 V			2	ms
Power supply rejection ratio (DC)	PSRR	full V_{DD} range			±0.01 ±5	%RH/V Pa/V
Standby time accuracy	$\Delta t_{standby}$			±5	±25	%

1.2 Humidity parameter specification[2]

Table 2: Humidity parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating range[3]	R_H	For temperatures < 0 °C and > 60 °C see Figure 1	-40	25	85	°C
			0		100	%RH
Supply current	$I_{DD,H}$	1 Hz forced mode, humidity and temperature		1.8	2.8	µA
Absolute accuracy tolerance	A_H	20...80 %RH, 25 °C, including hysteresis		±3		%RH
Hysteresis[4]	H_H	10→90→10 %RH, 25 °C		±1		%RH
Nonlinearity[5]	NL_H	10→90 %RH, 25 °C		1		%RH
Response time to complete 63% of step[6]	$\tau_{63\%}$	90→0 or 0→90 %RH, 25°C		1		s
Resolution	R_H			0.008		%RH
Noise in humidity (RMS)	N_H	Highest oversampling, see chapter 3.6		0.02		%RH
Long term stability	ΔH_{stab}	10...90 %RH, 25 °C		0.5		%RH/ year

[2] Target values
[3] When exceeding the operating range (e.g. for soldering), humidity sensing performance is temporarily degraded and reconditioning is recommended as described in section 7.8. Operating range only for non-condensing environment.
[4] For hysteresis measurement the sequence 10→30→50→70→90→70→50→30→10 %RH is used. The hysteresis is defined as the difference between measurements of the humidity up / down branch and the averaged curve of both branches
[5] Non-linear contributions to the sensor data are corrected during the calculation of the relative humidity by the compensation formulas described in section 4.2.3.
[6] The air-flow in direction to the vent-hole of the device has to be dimensioned in a way that a sufficient air exchange inside to outside will be possible. To observe effects on the response time-scale of the device an air-flow velocity of approx. 1 m/s is needed.

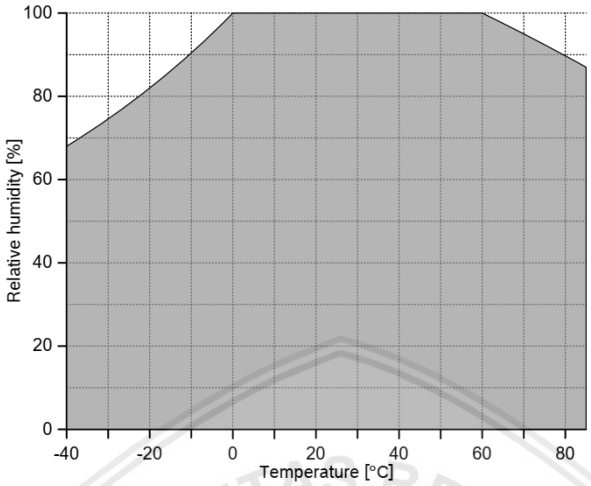


Figure 1: humidity sensor operating range

1.6 Pressure sensor specification

Table 3: Pressure parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating temperature range	T_A	operational	-40	25	+85	°C
		full accuracy	0		+65	
Operating pressure range	P	full accuracy	300		1100	hPa
Supply current	$I_{DD,LP}$	1 Hz forced mode, pressure and temperature, lowest power		2.8	4.2	µA
Temperature coefficient of offset[7]	TCO_P	25...65 °C, 900 hPa		±1.5		Pa/K
				±12.6		cm/K
Absolute accuracy pressure	$A_{P,full}$	300 . . . 1100 hPa 0 . . . 65 °C		±1.0		hPa
Relative accuracy pressure V_{DD} = 3.3V	A_{rel}	700 ... 900hPa 25 . . . 40 °C		±0.12		hPa

[7] When changing temperature by e.g. 10 °C at constant pressure / altitude, the measured pressure / altitude will change by (10 × TCO_P).

Resolution of pressure output data	R_P	Highest oversampling		0.18		Pa
Noise in pressure	$N_{P,fullBW}$	Full bandwidth, highest oversampling See chapter 3.6		1.3		Pa
				11		cm
	$N_{P,filtered}$	Reduced bandwidth, highest oversampling See chapter 3.6		0.2		Pa
				1.7		cm
Solder drift		Minimum solder height 50µm	-0.5		+2.0	hPa
Long term stability[8]	ΔP_{stab}	per year		±1.0		hPa
Possible sampling rate	f_{sample_P}	Lowest oversampling, see chapter 9.2	157	182		Hz

1.7 Temperature sensor specification

Table 4: Pressure parameter specification

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Operating range	T	Operational	-40	25	85	°C
		Full accuracy	0		65	°C
Supply current	$I_{DD,T}$	1 Hz forced mode, temperature measurement only		1.0		µA
Absolute accuracy temperature[9]	$A_{T,25}$	25 °C		±0.5		°C
	$A_{T,full}$	0...65 °C		±1.0		°C
Output resolution	R_T	API output resolution		0.01		°C
RMS noise	N_T	Lowest oversampling		0.005		°C

2. Absolute maximum ratings

The absolute maximum ratings are determined over complete temperature range using corner lots. The values are provided in Table 5.

[8] Long term stability is specified in the full accuracy operating pressure range 0 ... 65 °C
[9] Temperature measured by the internal temperature sensor. This temperature value depends on the PCB temperature, sensor element self-heating and ambient temperature and is typically above ambient temperature.

Table 5: Absolute maximum ratings

Parameter	Condition	Min	Max	Unit
Voltage at any supply pin	V_{DD} and V_{DDIO} pin	-0.3	4.25	V
Voltage at any interface pin		-0.3	V_{DDIO} + 0.3	V
Storage temperature	≤ 65% RH	-45	+85	°C
Pressure		0	20 000	hPa
ESD	HBM, at any pin		±2	kV
	CDM		±500	V
	Machine model		±200	V
Condensation	No power supplied	Allowed		

3. Functional description

3.1 Block diagram

Figure 2 shows a simplified block diagram of the BME280:

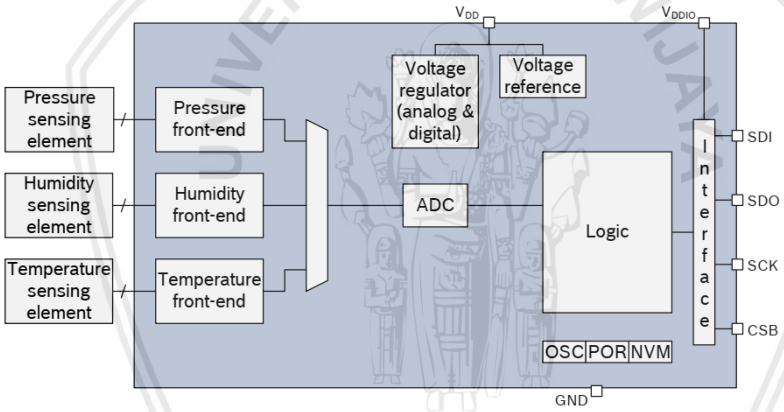


Figure 2: Block diagram of BME280

3.2 Power management

The BME280 has two distinct power supply pins

- V_{DD} is the main power supply for all internal analog and digital functional blocks
- V_{DDIO} is a separate power supply pin used for the supply of the digital interface

A power-on reset (POR) generator is built in; it resets the logic part and the register values after both V_{DD} and V_{DDIO} reach their minimum levels. There are no limitations on slope and sequence

of raising the V_{DD} and V_{DDIO} levels. After powering up, the sensor settles in sleep mode (described in chapter 3.3.2).

It is prohibited to keep any interface pin (SDI, SDO, SCK or CSB) at a logical high level when V_{DDIO} is switched off. Such a configuration can permanently damage the device due an excessive current flow through the ESD protection diodes.

If V_{DDIO} is supplied, but V_{DD} is not, the interface pins are kept at a high-Z level. The bus can therefore already be used freely before the BME280 V_{DD} supply is established.

Resetting the sensor is possible by cycling V_{DD} level or by writing a soft reset command. Cycling the V_{DDIO} level will not cause a reset.

3.3 Sensor modes

The BME280 offers three sensor modes: sleep mode, forced mode and normal mode. These can be selected using the *mode*[1:0] setting (see chapter 5.4.5). The available modes are:

- Sleep mode: no operation, all registers accessible, lowest power, selected after startup
- Forced mode: perform one measurement, store results and return to sleep mode
- Normal mode: perpetual cycling of measurements and inactive periods.

The modes will be explained in detail in chapters 3.3.2 (sleep mode), 3.3.3 (forced mode) and 3.3.4 (normal mode).

3.3.1 Sensor mode transitions

The supported mode transitions are shown in Figure 3. If the device is currently performing a measurement, execution of mode switching commands is delayed until the end of the currently running measurement period. Further mode change commands or other write commands to the register *ctrl_hum* are ignored until the mode change command has been executed. Mode transitions other than the ones shown below are tested for stability but do not represent recommended use of the device.

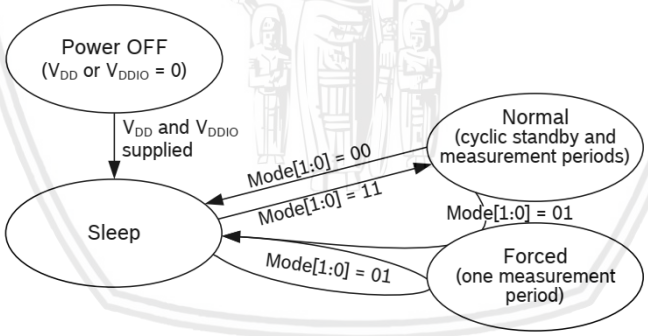


Figure 3: Sensor mode transition diagram

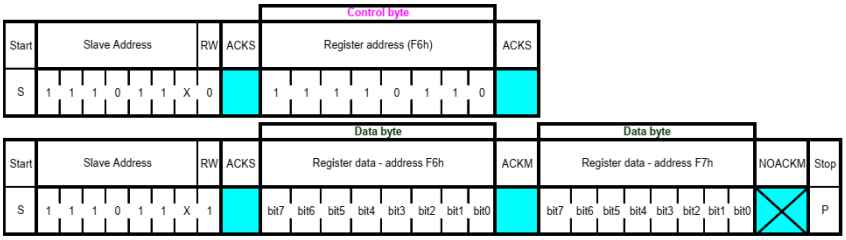


Figure 10: I²C multiple byte read

6.3 SPI interface

The SPI interface is compatible with SPI mode '00' (CPOL = CPHA = '0') and mode '11' (CPOL = CPHA = '1'). The automatic selection between mode '00' and '11' is determined by the value of SCK after the CSB falling edge.

The SPI interface has two modes: 4-wire and 3-wire. The protocol is the same for both. The 3-wire mode is selected by setting '1' to the register spi3w_en. The pad SDI is used as a data pad in 3-wire mode.

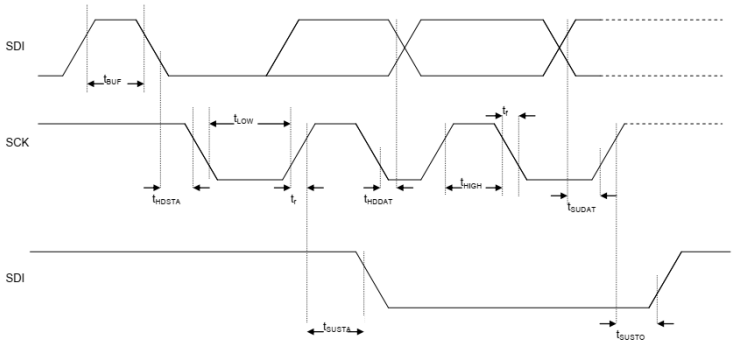


Figure 14: I²C timing diagram

Table 33: I²C timings

Parameter	Symbol	Condition	Min	Typ	Max	Unit
SDI setup time	$t_{SU;DAT}$	S&F Mode HS mode	160 30			ns ns
SDI hold time	$t_{HD;DAT}$	S&F Mode, C_b≤100 pF S&F Mode, C_b≤400 pF HS mode, C_b≤100 pF HS mode, C_b≤400 pF	80 90 18 24		 115 150	ns ns ns ns
SCK low pulse	t_{LOW}	HS mode, C_b≤100 pF V_{DDIO} = 1.62 V	160			ns
SCK low pulse	t_{LOW}	HS mode, C_b≤100 pF V_{DDIO} = 1.2 V	210			ns

The above-mentioned I²C specific timings correspond to the following internal added delays:

- Input delay between SDI and SCK inputs: SDI is more delayed than SCK by typically 100 ns in Standard and Fast Modes and by typically 20 ns in High Speed Mode.
- Output delay from SCK falling edge to SDI output propagation is typically 140 ns in Standard and Fast Modes and typically 70 ns in High Speed Mode.

6.4.3 SPI timings

The SPI timing diagram is in Figure 15, while the corresponding values are given in Table 34. All timings apply both to 4- and 3-wire SPI.

7. Pin-out and connection diagram

7.1 Pin-out

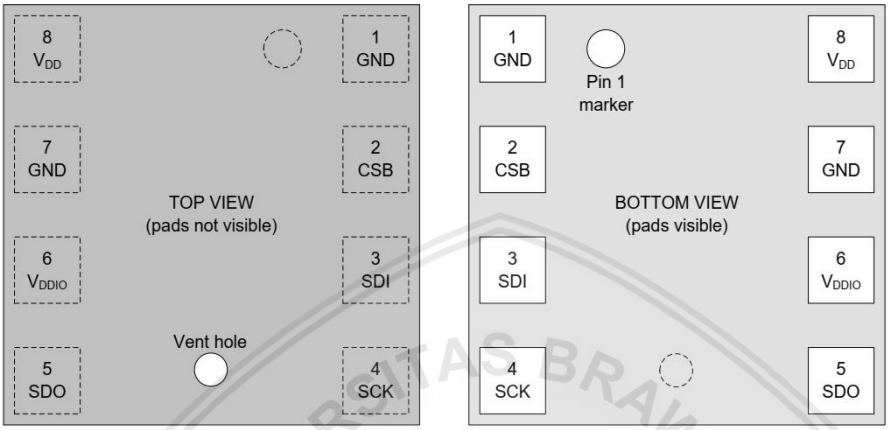


Figure 16: Pin-out top and bottom view

Note: The pin numbering of BME280 is performed in the untypical clockwise direction when seen in top view and counter-clockwise when seen in bottom view.

Table 35: Pin description

Pin	Name	I/O Type	Description	Connect to		
				SPI 4W	SPI 3W	I²C
1	GND	Supply	Ground	GND		
2	CSB	In	Chip select	CSB	CSB	V_{DDIO}
3	SDI	In/Out	Serial data input	SDI	SDI/SDO	SDA
4	SCK	In	Serial clock input	SCK	SCK	SCL
5	SDO	In/Out	Serial data output	SDO	DNC	GND for default address
6	V_{DDIO}	Supply	Digital / Interface supply	V_{DDIO}		
7	GND	Supply	Ground	GND		
8	V_{DD}	Supply	Analog supply	V_{DD}		

7.2 Connection diagram I²C

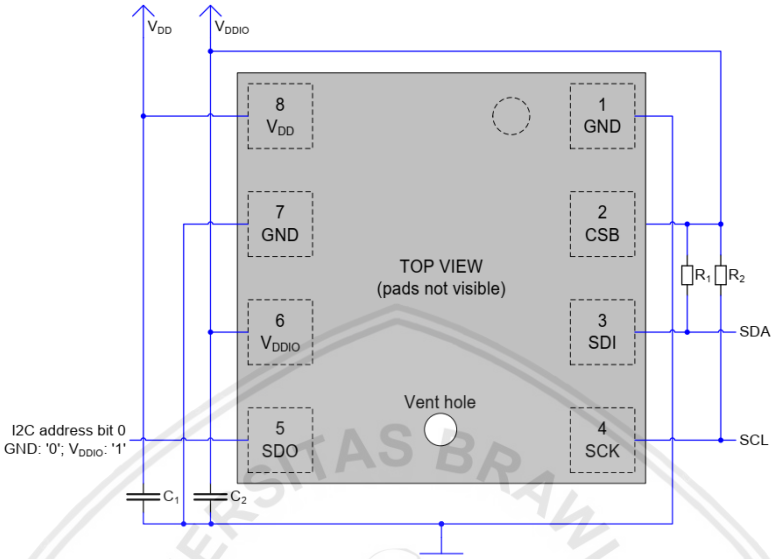


Figure 17: I²C connection diagram

Notes:

- The recommended value for C_1, C_2 is 100 nF
- The value for the pull-up resistors R_1, R_2 should be based on the interface timing and the bus load; a normal value is 4.7 kΩ
- A direct connection between CSB and V_{DDIO} is required