

ANALISIS PERFORMA ALGORITMA SPECK PADA RASPBERRY PI

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:

Panji Mansyur Ansyah

NIM:135150307111020



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

PENGESAHAN

ANALISIS PERFORMA ALGORITMA SPECK PADA
RASPBERRY PI

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Panji Mansyur Ansyah
NIM:135150307111020

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Mochammad Hannats Hanafi Ichsan, S.ST, M.T
NIK: 201405 881229 1 001

Dosen Pembimbing II

Ari Kusyanti, S. T, M.Sc
NIP: 19831228 201803 2 002

Mengetahui

Ketua Jurusan Teknik Informatika



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

A



PERNYATAAN ORISINALITAS


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

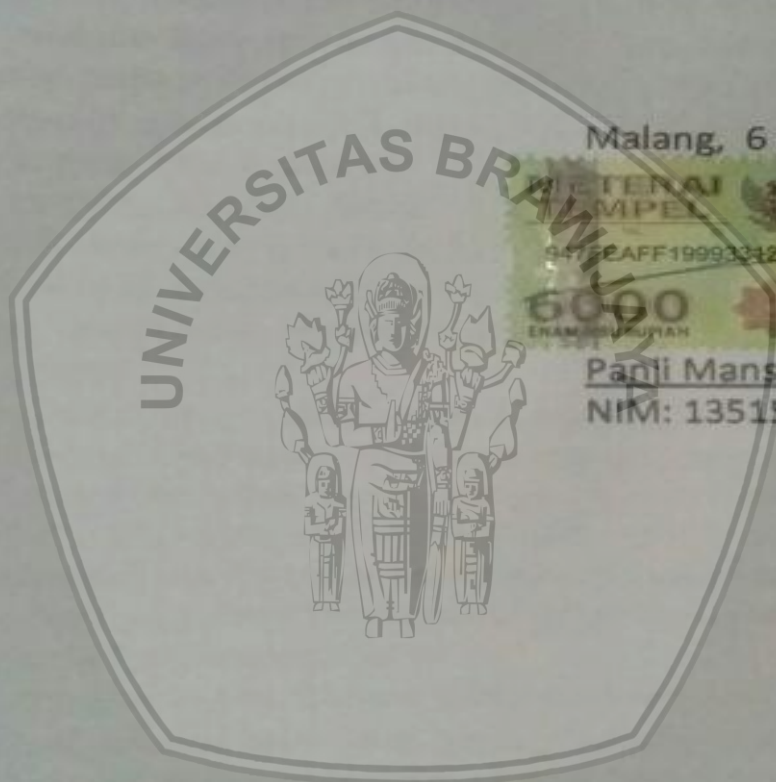
Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat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

Malang, 6 Agustus 2018

BIJUTENAI
MPEL
947EEAFF199933125

6000
ENAM RIBU RUPIAH


Panji Mansyur Ansyah
NIM: 135150307111020



KATA PENGANTAR

Puja dan puji syukur penulis panjatkan kepada Allah SWT karena berkat rahmat dan hidayah-Nya, penulis dapat menyelesaikan penelitian dan Laporan Skripsi untuk memperoleh gelar Sarjana Komputer yang berjudul **Analisis Performa Algoritma SPECK pada Raspberry Pi**

Dalam pelaksanaan dan penyusunan laporan skripsi ini, tidak sedikit hambatan yang penulis hadapi. Namun penulis menyadari bahwa kelancaran dalam penyusunan laporan ini tidak lain berkat bantuan, dorongan dan bimbingan dari berbagai pihak, sehingga kendala-kendala yang penulis hadapi dapat teratasi. Penghargaan dan terima kasih yang sebesar-besarnya penulis sampaikan kepada:

1. Bapak Pamuji Joko Raharjo, Ibu Sunik selaku kedua orangtua penulis yang tiada henti memberikan semangat beserta dukungan baik berupa do'a maupun materi selama penulis melakukan penelitian.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D selaku ketua Jurusan Teknik Informatika periode 2016 – saat ini.
3. Bapak Sabriansyah Rizqika Akbar, S.T., M.Eng. selaku ketua jurusan program teknik komputer periode 2016 – saat ini.
4. Bapak Mochammad Hannats Hanafi S.ST, M.T selaku pembimbing I yang telah banyak membantu memberikan arahan dan bimbingan selama pengerjaan skripsi.
5. Ibu Ari Kusyanti S.T, M.Sc. selaku pembimbing II yang telah banyak membantu memberi arahan dan bimbingan selama proses pengerjaan skripsi.
6. Muhammad Abdul Aziz dan Pandi Aldrige, dan teman-temanku satu kos Panjahitan X/17 yang selalu mengisi hari-hari penulis untuk menemani dan selalu menyemangati penulis untuk segera menyelesaikan penulisan skripsi ini.
7. Dan seluruh pihak yang tidak bisa saya sebutkan satu persatu yang juga telah berperan dalam penyelesaian skripsi ini.

Akhir kata penulis menyadari bahwa pada penyusunan laporan skripsi ini masih terdapat banyak kekurangan yang perlu disempurnakan. Oleh sebab itu kritik beserta saran sangat membangun dan penulis harapkan. Semoga skripsi ini bisa bermanfaat bagi pihak yang membacanya.

Malang, 6 Agustus 2015

Penulis

Panjiansyah13@gmail.com

ABSTRAK

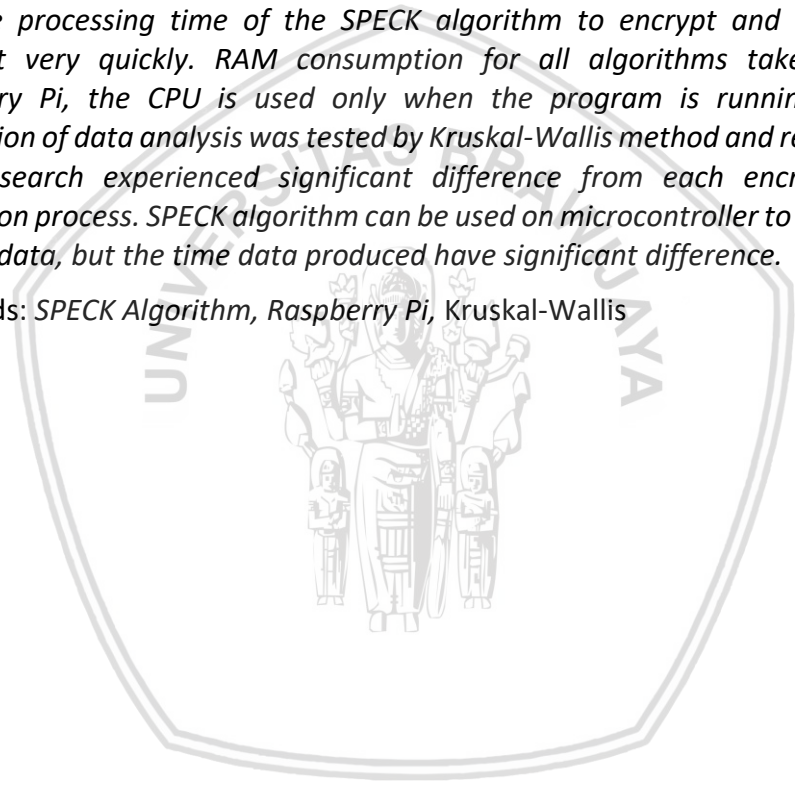
Algoritme SPECK adalah algoritme enkripsi dan dekripsi yang merupakan keluarga dari SIMON *Cipher*. Algoritme ini merupakan algoritme terbaru yang ditemukan oleh Nasional *Security Agency*(NSA). SPECK *block Cipher* memiliki 10 macam *block* yang terdiri dari 32 bit yang paling terkecil hingga 128 bit *block* yang terbesar. Setiap *block size* memiliki kriteria tersendiri agar algoritme berjalan sempurna. Proses enkripsi dan dekripsi sudah teramat banyak diimplementasikan dalam *software*. Penelitian ini dilakukan untuk mengetahui performa algoritme SPECK jika diterapkan pada mikrokontroler Raspberry Pi. Dalam penelitian ini menggunakan metode pengumpulan data dengan menjalankan algoritme SPECK pada Raspberry Pi dan pengujian dilakukan sebanyak 1024 kali pada setiap algoritme SPECK. Hasil dari penelitian menyimpulkan bahwa waktu proses dari algoritme SPECK untuk mengenkripsi dan mendekripsikan *plaintext* sangat cepat. Konsumsi RAM untuk semua algoritme membutuhkan 0,5% pada Raspberry Pi, CPU hanya digunakan saat program berjalan saja sebesar 2,6%. Perhitungan analisis data diuji dengan metode Kruskal-Wallis dan menghasilkan data dari penelitian mengalami perbedaan signifikan dari setiap proses enkripsi dan dekripsi. Algoritme SPECK mampu digunakan pada mikrokontroler untuk mengenkripsi dan mendekripsi data, namun data waktu yang dihasilkan memiliki perbedaan signifikan.

Kata kunci: Algoritme SPECK, *Raspberry pi*, Kruskal-Wallis

ABSTRACT

The SPECK algorithm is the encryption and decryption algorithm that is the family of SIMON Cipher. This algorithm is the latest algorithm found by the National Security Agency (NSA). SPECK block Cipher has 10 kinds of block consisting of 32 bit of the most smallest to 128 bit block biggest. Each block size has its own criteria for the algorithm to run perfectly. Encryption and decryption process is very much implemented in software. This research is done to know SPECK algorithm performance if applied to Raspberry Pi microcontroller. In this study using data collection method by running SPECK algorithm on Raspberry Pi and testing done as much as 1024 times on each SPECK algorithm. The results of the study concluded that the processing time of the SPECK algorithm to encrypt and decrypt the plaintext very quickly. RAM consumption for all algorithms takes 0.5% on Raspberry Pi, the CPU is used only when the program is running at 2.6%. Calculation of data analysis was tested by Kruskal-Wallis method and resulted data from research experienced significant difference from each encryption and decryption process. SPECK algorithm can be used on microcontroller to encrypt and decrypt data, but the time data produced have significant difference.

Keywords: SPECK Algorithm, Raspberry Pi, Kruskal-Wallis



DAFTAR ISI

PENGESAHAN	ii
PERNYATAAN ORISINALITAS	iii
KATA PENGANTAR.....	iv
ABSTRAK.....	v
ABSTRACT	vi
DAFTAR ISI.....	vii
DAFTAR TABEL.....	x
DAFTAR GAMBAR.....	xii
DAFTAR LAMPIRAN	xiii
BAB 1 PENDAHULUAN.....	1
1.1 Latar belakang.....	1
1.2 Rumusan masalah.....	3
1.3 Tujuan	3
1.4 Manfaat.....	3
1.5 Batasan masalah	3
1.6 Sistematika pembahasan.....	4
BAB 2 LANDASAN KEPUSTAKAAN	5
2.1 Kajian Pustaka	5
2.2 Dasar Teori.....	5
2.2.1 Kriptografi	5
2.2.2 Confidentiality	6
2.2.3 Symetrik Key.....	6
2.2.4 Asymetric Key	7
2.2.5 Algoritme SPECK.....	7
2.2.6 Raspberry Pi	8
2.2.7 Kruskal-Wallis	9
BAB 3 METODOLOGI	10
3.1 Studi Literatur	10
3.2 Studi Literatur	11
3.3 Analisis Kebutuhan	11
3.3.2 Kebutuhan Perangkat Keras.....	11



3.3.3	Kebutuhan Perangkat Lunak	11
3.4	Perancangan Sistem.....	12
3.4.1	Perancangan Perangkat Lunak.....	12
3.5	Perancangan Hardware	13
3.6	Pengujian dan Analisis Hasil.....	13
3.6.1	Pengujian Fungsional Hardware	13
3.6.1.1	Tujuan Pengujian.....	13
3.6.1.2	Prosedur Pengujian	13
3.7	Kesimpulan.....	13
BAB 4	HASIL.....	14
4.1	Implementasi <i>hardware</i>	14
4.1.1	Implementasi Raspberry Pi	14
4.1.2	Prosedur Pengujian	14
4.1.3	Pengujian Raspberry Pi	15
4.2	Implementasi <i>software</i>	15
4.2.1	Perancangan Proses Enkripsi dan Dekripsi Data.....	15
4.2.2	Proses manualisasi enkripsi	16
4.2.3	Implementasi Algoritme SPECK.....	17
4.2.3.1	Implementasi kode program input pada Algoritme SPECK	18
4.2.3.2	Implementasi kode program key pada Algoritme SPECK	19
4.2.3.3	Implementasi kode program proses enkripsi dan dekripsi.....	21
4.2.3.4	Implementasi kode putaran bit Algoritme SPECK.....	22
BAB 5	pembahasan	24
5.1	Hasil Pengujian.....	24
5.1.1	Hasil pengujian <i>Test Vektor</i>	24
5.1.1.1	Test Vektor	24
5.1.2	Hasil Analisis Waktu Algoritme SPECK	27
5.1.3	Uji Kruskal-Wallis.....	30
5.1.3.1	Uji Kruskal-Wallis H	30
5.1.4	Uji <i>Post Hoc Test</i>	32
5.1.4.1	Waktu Key schedule.....	33
5.1.4.2	Waktu Enkripsi	34



5.1.4.3 Waktu Dekripsi	36
5.1.5 Hasil pengujian CPU dan RAM	38
BAB 6 penutup	39
6.1 Kesimpulan.....	39
6.2 Saran	39
DAFTAR PUSTAKA.....	40
LAMPIRAN A DATA UJI WAKTU ALGORITME SPECK	41
A.1 Tabel Data Uji Waktu Algoritme SPECK.....	41
LAMPIRAN B Kode Program Utama Algoritme SPECK	89
B.1 Kode Program Algoritme SPECK	89



DAFTAR TABEL

Tabel 2.1 Parameter Ukuran <i>Block</i> dan <i>Key</i> Serta <i>Round</i>	7
Tabel 4.1 Kode program inialisasi SPECK <i>block size</i> 128bit <i>key size</i> 128 bit	17
Tabel 4.2 Kode program inialisasi SPECK <i>block size</i> 128bit <i>key size</i> 192 bit	17
Tabel 4.3 Kode program inialisasi SPECK <i>block size</i> 128bit <i>key size</i> 256 bit	17
Tabel 4.4 Kode program <i>Input</i> SPECK <i>block size</i> 128bit <i>key size</i> 128 bit	18
Tabel 4.5 Kode program <i>Input</i> SPECK <i>block size</i> 128bit <i>key size</i> 192 bit	18
Tabel 4.6 Kode program <i>Round</i> dan <i>key</i> SPECK <i>block size</i> 128bit <i>key size</i> 256bit	19
Tabel 4.7 Kode program <i>key</i> SPECK <i>block size</i> 128bit <i>key size</i> 128bit	19
Tabel 4.8 Kode program <i>key</i> SPECK <i>block size</i> 128bit <i>key size</i> 192bit	20
Tabel 4.9 Kode program <i>key</i> SPECK <i>block size</i> 128bit <i>key size</i> 256bit	20
Tabel 4.10 Kode program enkripsi, dekripsi SPECK <i>block</i> 128bit <i>key</i> 128bit.....	21
Tabel 4.11 Kode program enkripsi, dekripsi SPECK <i>block</i> 128bit <i>key</i> 192bit.....	21
Tabel 4.12 Kode program enkripsi, dekripsi SPECK <i>block</i> 128bit <i>key</i> 256bit.....	22
Tabel 4.13 Kode program <i>circular</i> bit SPECK <i>block size</i> 128bit <i>key size</i> 128bit	22
Tabel 4.14 Kode program <i>circular</i> bit SPECK <i>block size</i> 128bit <i>key size</i> 192bit	23
Tabel 4.15 Kode program <i>circular</i> bit SPECK <i>block size</i> 128bit <i>key size</i> 256bit	23
Tabel 5.1 Hasil Pengujian Pada Raspberry Pi	25
Tabel 5.2 Hasil Pengujian Pada Raspberry Pi	26
Tabel 5.3 Hasil Pengujian Pada Raspberry Pi	26
Tabel 5.4 Hasil Pengujian Pada Raspberry Pi	27
Tabel 5.5 Rata – rata waktu algorima SPECK	28
Tabel 5.6 Rata – rata waktu algorima SPECK	29
Tabel 5.7 Rata – rata waktu algorima SPECK	29
Tabel 5.8 Deskriptif Kruskal-Wallis waktu <i>Key schedule</i>	30
Tabel 5.9 Kruskal-Wallis test waktu <i>Key schedule</i>	30
Tabel 5.10 Hasil Uji Kruskal-Wallis waktu <i>Key schedule</i>	31
Tabel 5.11 Deskriptif Kruskal-Wallis waktu enkripsi.....	31
Tabel 5.12 Kruskal-Wallis test waktu enkripsi	31
Tabel 5.13 Hasil uji Kruskal-Wallis waktu enkripsi.....	31
Tabel 5.14 Hasil uji Kruskal-Wallis waktu dekripsi.....	32



Tabel 5.15 Hasil uji Kruskal-Wallis waktu dekripsi	32
Tabel 5.16 Hasil uji Kruskal-Wallis waktu dekripsi	32
Tabel 5.17 Hasil uji Kruskal-Wallis waktu <i>key schedule</i>	33
Tabel 5.18 Hasil uji <i>post hoc tests</i> waktu <i>key schedule</i>	33
Tabel 5.19 Hasil uji <i>post hoc tests</i> waktu <i>key schedule</i>	33
Tabel 5.20 Hasil uji <i>post hoc tests</i> waktu <i>key schedule</i>	33
Tabel 5.21 Hasil uji Kruskal-Wallis tests waktu enkripsi	34
Tabel 5.22 Hasil uji <i>post hoc tests</i> waktu enkripsi	35
Tabel 5.23 Hasil uji <i>post hoc tests</i> waktu enkripsi	35
Tabel 5.24 Hasil uji <i>post hoc tests</i> waktu enkripsi	35
Tabel 5.25 Hasil uji Kruskal-Wallis tests waktu dekripsi	36
Tabel 5.26 Hasil uji <i>post hoc tests</i> waktu dekripsi	36
Tabel 5.27 Hasil uji <i>post hoc tests</i> waktu dekripsi	37
Tabel 5.28 Hasil uji <i>post hoc tests</i> waktu dekripsi	37



DAFTAR GAMBAR

Gambar 2.1 Raspberry Pi	8
Gambar 3.1 Metodologi Penelitian.....	10
Gambar 3.2 Diagram Alir Cara Kerja Sistem	12
Gambar 3.3 Block Diagram Sistem.....	13
Gambar 4.1 Perancangan Raspberry Pi	14
Gambar 4.2 Akses Raspberry Pi	15
Gambar 4.3 Diagram Alir Perancangan Perangkat Lunak Algoritme SPECK.....	15
Gambar 5.1 Grafik Waktu Algoritme SPECK	28
Gambar 5.2 Grafik Waktu Algoritme SPECK	28
Gambar 5.3 Grafik Waktu Algoritme SPECK	29
Gambar 5.4 RAM dan CPU saat dijalankan	38
Gambar 5.5 RAM dan CPU saat program sudah berjalan.....	38



DAFTAR LAMPIRAN

LAMPIRAN A Data uji waktu Algoritme SPECK.....	41.
A.1 Tabel data uji waktu Algoritme SPECK.....	41.
LAMPIRAN B Kode Program Utama Algoritme SPECK.....	89.
B.1 Kode Program algoirma SPECK.....	89.



BAB 1 PENDAHULUAN

1.1 Latar belakang

Algoritme mempunyai sejarah yang menarik, kata ini muncul dalam kamus *Webster* sampai akhir tahun 1957. Algoritme mempunyai arti proses perhitungan dalam bahasa arab. Definisi terminologi algoritme adalah urutan langkah-langkah logis untuk menyelesaikan masalah yang disusun secara sistematis. Kriptografi merupakan ilmu yang mempelajari dimana pengamanan sebuah komunikasi yang nantinya akan hadir pihak ketiga. Algoritme kriptografi merupakan langkah-langkah logis bagaimana menyembunyikan pesan dari orang-orang yang tidak berhak atas pesan tersebut. Kriptografi memiliki dua kategori *Stream Cipher* dan *Block Cipher*. Kedua kategori tersebut memiliki perbedaan dalam proses pengamanan data. *Block Cipher* yang sampai sekarang dikembangkan oleh para ahli kriptografi. Berbagai macam *Block Cipher* yang sudah digunakan sampai sekarang dan salah satunya adalah SPECK (Ariyus, 2008).

SPECK *Cipher* adalah sebuah algoritme enkripsi dan dekripsi yang merupakan keluarga dari SIMON *Cipher*. Algoritme ini merupakan algoritme terbaru yang diluncurkan oleh NSA pada tahun 2013. SPECK block *Cipher* memiliki 10 macam block yang terdiri dari 32 bit yang paling terkecil hingga 128 bit block yang terbesar. Setiap block size memiliki kriteria tersendiri agar algoritme berjalan sempurna. Enkripsi dan dekripsi SPECK di implementasikan pada *software* dan *hardware* (Adrian-Vasile Duka, dkk, 2017).

Proses enkripsi dan dekripsi sudah teramat banyak diimplementasikan dalam *software*. Sedangkan pada perangkat seperti mikrokontroler hanya dalam beberapa alat yang menggunakan enkripsi dan dekripsi. Perangkat yang menggunakan metode enkripsi dan dekripsi tersebut digunakan untuk menjaga data atau identitas dari pengguna (*Confidentiality*). Sehingga Agar dapat memahami SPECK *Cipher* maka diangkat masalah dari implementasi SPECK pada *hardware*. Untuk mengetahui kemampuan dari algoritme SPECK saat dijalankan pada Raspberry Pi, kecepatan proses dari algoritme SPECK, dan performa SPECK saat di implementasikan pada *hardware*.

Cara kerja dari algoritme SPECK dengan pemrosesan dari blok – blok yang berisikan bit dengan panjang yang sudah ditentukan oleh parameter dari SPECK. SPECK yang sudah dimasukkan *input* data *plaintext* dan *key* (*string*, *integer*, heksa) atau apapun bentuk dari masukkan, maka *input* tersebut akan diubah menjadi sekelompok biner. Biner tersebut masuk kedalam 2 proses yakni, proses *Key Schedule* dan proses SPECK *round*. SPECK *round* ini ditentukan dalam parameter algoritme SPECK. Dalam proses tersebut akan berjalan berulang-ulang sehingga menghasilkan sebuah *Ciphertext*. *Ciphertext* ini akan dimasukkan lagi dalam proses yang sama namun prosesnya dilakukan secara berlawanan dari proses sebelumnya, yang akan menghasilkan *deciphertext*.

Penelitian ini memakai algoritme SPECK untuk mengetahui dan membuktikan bahwa algoritme SPECK ini juga mampu diimplementasikan pada mikrokontroler maupun Raspberry Pi. Sehingga penggunaan algoritme SPECK mampu dikembangkan pada *software* dan *hardware*. Perangkat yang diujikan untuk

algoritme SPECK yaitu Raspberry Pi, karena pada literatur yang dipakai sebagai dasar teori pada penelitian ini menjelaskan bahwa algoritme SPECK sudah diimplementasikan pada mikrokontroler, sehingga pada penelitian ini memilih Raspberry Pi untuk dijadikan sebagai pemroses algoritme SPECK. Pemilihan Raspberry Pi juga berguna untuk menjalankan algoritme SPECK yang memiliki panjang bit melebihi 64 bit. Sehingga dapat membantu proses penelitian ini untuk menganalisis seluruh tipe dari SPECK.

Raspberry Pi adalah sebuah komputer mini yang berbentuk seperti mikrokontroler yang memiliki kinerja lebih rendah dari pada PC *desktop*. Raspberry Pi mengkonsumsi daya sangat rendah yaitu 3,5 Watt. Raspberry Pi juga memiliki banyak *port* seperti USB, HDMI, LAN dll. *Port* tersebut hanya dimiliki pada Raspberry Pi, karena ini merupakan pengembangan fitur dari Raspberry Pi sebelumnya.

Penelitian ini memfokuskan pada *confidentiality* yaitu menjaga data atau informasi pribadi dari pihak yang tidak memiliki hak untuk mengetahui informasi tersebut. Sehingga dilakukan proses pengkodean atau enkripsi data dengan dekripsi data. Perangkat ini juga memiliki port yang sudah tersedia pada *board* Raspberry Pi. Pada penelitian ini menggunakan *port* LAN untuk menghubungkan dengan jaringan lokal pada laptop dengan cara *setting* IP laptop dan IP Raspberry Pi berada pada satu area *network*. Dengan memanfaatkan teknologi dari Raspberry Pi sebagai pemroses algoritme SPECK mampu dikendalikan menggunakan laptop sehingga mempermudah untuk menganalisis proses algoritme SPECK (Permana, 2014).

Dalam paper NSA melakukan penelitian yang menguji performa algoritme SPECK pada perangkat keras dan perangkat Lunak. Perangkat keras yang digunakan NSA mikrokontroler 8-bit yang mempunyai *clock speed* 100 kHz dan perangkat Lunak 16 kHz. SIMON telah dioptimalkan untuk berjalan pada perangkat keras dan SPECK telah di optimalkan berjalan pada perangkat lunak. Namun kedua keluarga block Cipher ini mampu berjalan dan bekerja dengan baik pada perangkat lunak dan perangkat keras. *User* mampu menggunakan kedua Cipher dengan mencocokkan kebutuhan aplikasi dan perangkat keras tanpa mengganggu kinerja. Pengujian yang dilakukan NSA dengan Memberikan batasan kebutuhan daya yang seminimal mungkin untuk mengoptimalkan implementasi dalam mikrokontroler yang berbiaya rendah dengan keterbatasan SRAM dan keterbatasan daya (Ray Beaulieu dkk, 2013).

Pada tahun 2004 Arif Kusbandono VLSI-RG Dept. Teknik Elektro ITB melakukan tesisnya pada algoritme AES-128 byte pada Mikrokontroler 8051 pada penelitian tersebut melakukan pengujian algoritme AES menggunakan metode optimasi menggunakan bahasa C dan *assembly* pada mikrokontroler yang terbatas kapasitas program 4kb dan memori data 128byte, disini penulis telah mampu mengimplementasikan optimasi hingga mencapai ukuran program total 3407byte; kecepatan komputasi 3658 *cycle* dan 5648 *cycle*, berturut-turut untuk enkripsi dan dekripsi. (Kusbandono, 2004).

Dalam paper penelitian Boris Ryabko dengan judul "The distinguishing attack on SPECK, SIMON, Simeck, HIGHT and LEA" dalam penelitian ini menganalisis

Cipher block ringan, SPECK dan SIMON dengan ketahanan terhadap serangan pembeda. Penelitian ini menggunakan blok *Cipher* berbasis ARX yang dirancang hanya menggunakan rotasi dan XOR modular (Boris Ryabko, 2018).

Dari penjelasan paper diatas bahwa algoritme SPECK itu mampu bekerja pada perangkat lunak dan perangkat keras. Penulis ingin menganalisis performa algoritme SPECK yang akan di pasang pada perangkat keras Raspberry Pi. Pengujian akan berjalan melalui Performa yang dianalisis adalah proses kecepatan algoritme SPECK, memori yang dibutuhkan untuk menjalankan program, keakuratan hasil enkripsi dan dekripsi berdasarkan *test vektor*. Analisis data waktu proses akan diuji dengan Kruskal-Wallis untuk mengetahui perbedaan waktu data yang telah diuji.

1.2 Rumusan masalah

Berdasarkan latar belakang yang ada, maka dapat dirumuskan beberapa permasalahan berikut :

1. Bagaimana cara implementasi algoritme SPECK pada Raspberry Pi ?
2. Bagaimana performansi algoritme SPECK pada Raspberry Pi ?
3. Bagaimana validasi *plaintext* dan *Ciphertext* algoritme SPECK pada Raspberry Pi ?

1.3 Tujuan

Adapun beberapa tujuan dari penelitian yang dilaksanakan antara lain:

1. Dapat mengetahui implementasi algoritme SPECK pada Raspberry Pi.
2. Dapat mengetahui performa algoritme SPECK.
3. Dapat mengetahui validasi hasil enkripsi dan dekripsi.

1.4 Manfaat

Melalui penelitian ini diharapkan dapat memberikan manfaat bagi :

1. Algoritme SPECK mampu dikembangkan pada mikrokontroler yang lain dengan penambahan fungsi tertentu.
2. Sebagai bahan referensi bagi peneliti yang akan memfokuskan penelitiannya pada Algoritme SPECK.
3. Algoritme SPECK mampu diterapkan di berbagai aplikasi perusahaan maupun lainnya sebagai security.

1.5 Batasan masalah

Dalam perancangan sistem ini, terdapat batasan masalah diantaranya adalah:

1. Penggunaan sistem hanya pada Raspberry Pi.
2. Sistem menggunakan manual input dengan nilai angka desimal.
3. Sistem ini menampilkan hasil enkripsinya dan dekripsi.
4. Hardware yang digunakan Laptop, Raspberry Pi dan Kabel LAN.
5. Sistem ini menampilkan hasil enkripsi dan dekripsi dalam bentuk desimal dan heksa.
6. Penelitian ini hanya memfokuskan pada *confidentiality*.

1.6 Sistematika pembahasan

Sebagai acuan dalam penulisan agar dapat terarah sesuai dengan yang diharapkan, maka disusun sistematika penulisan Bab dan subbab.

Bab I Pendahuluan

Pada bab ini berisi tentang algoritme SPECK yang di implementasi pada perangkat lunak dan perangkat keras, tujuan skripsi ini mengimplementasikan algoritme SPECK pada Raspberry Pi.

Bab II Landasan Teori

Pada bab ini berisi tentang dasar teori algoritme SPECK dan dasar dari mikrokontroler Raspberry Pi.

Bab III Metodologi

Pada bab ini berisi tentang metode-metode perancangan model sistem yang dipakai dalam penelitian secara spesifik yaitu, studi literatur, analisis kebutuhan, pengolahan data, perancangan sistem dan implementasi.

Bab IV Hasil

Pada bab ini akan dibahas bagaimana merancang algoritme SPECK dan proses berjalannya enkripsi, dekripsi, serta bagaimana cara mengimplementasikannya pada Raspberry Pi.

Bab V Pembahasan

Pada bab ini membahas mengenai hasil dari pengujian terhadap objek penelitian.

Bab VI Penutup

Pada bab ini membahas mengenai kesimpulan dan saran dari seluruh isi dari laporan penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Pada tahun 2004 Arif Kusbandono VLSI-RG Dept. Teknik Elektro ITB melakukan tesisnya pada algoritme AES-128 *byte* pada Mikrokontroler 8051 pada penelitian tersebut melakukan pengujian algoritme AES pada mikrokontroler yang terbatas kapasitas program 4kb dan memori data 128*byte*, disini penulis telah mampu mengimplemmentasikan optimasi hingga mencapai ukuran program total 3407*byte*; kecepatan komputasi 3658 *cycle* dan 5648 *cycle*, berturut-turut untuk enkripsi dan dekripsi (Kusbandono, 2004).

Dalam paper NSA melakukan penelitian yang menguji performa algoritme SPECK pada perangkat keras dan perangkat Lunak. Perangkat keras yang digunakan NSA mikrokontroler 8-bit yang mempunyai *clock speed* 100 kHz dan perangkat Lunak 16 kHz. SIMON telah dioptimalkan untuk berjalan pada perangkat keras dan SPECK telah di optimalkan berjalan pada perangkat lunak. Namun kedua keluarga *block Cipher* ini mampu berjalan dan bekerja dengan baik pada perangkat lunak dan perangkat keras. *User* mampu menggunakan kedua Cipher dengan mencocokkan kebutuhan aplikasi dan perangkat keras tanpa mengganggu kinerja. Pengujian yang dilakukan NSA dengan Memberikan batasan kebutuhan daya yang seminimal mungkin untuk mengoptimalkan implementasi dalam mikrokontroler yang berbiaya rendah dengan keterbatasan SRAM dan keterbatasan daya (Ray Beaulieu dkk, 2013).

Dalam paper penelitian Boris Ryabko dengan judul "The distinguishing attack on SPECK, SIMON, Simeck, HIGHT and LEA" dalam penelitian ini menganalisis *Cipher block* ringan, SPECK dan SIMON dengan ketahanan terhadap serangan pembeda. Penelitian ini menggunakan blok *Cipher* berbasis ARX yang dirancang hanya menggunakan rotasi dan XOR modular (Boris Ryabko, 2018).

Dari penelitian tersebut menunjukkan bahwa algoritme SPECK ini juga bisa dijalankan pada mikrokontroler raspberri pi, dengan menggunakan bahasa pemrograman yang bisa digunakan pada Raspberry Pi maka mampu untuk mengimplemmentasikan algoritme SPECK pada Raspberry Pi.

Perbedaan penelitian sebelum dengan penelitian sekarang adalah pada penggunaan mikrokontroler, proses pengujian, dan data yang diuji. Pada penelitian ini akan diujikan dari waktu proses algoritme berjalan, konsumsi RAM saat algoritme berjalan, keakuratan hasil dari *output* berdasarkan *test vektor*.

2.2 Dasar Teori

2.2.1 Kriptografi

Kriptografi merupakan ilmu yang mempelajari dimana pengamanan sebuah komunikasi yang nantinya akan hadir pihak ketiga. Kriptografi juga mampu mengkonstruksi dan menganalisis protokol, dalam berbagai aspek keamanan informasi seperti data rahasia, integritas data, autentikasi dan non-repudansi.

Banyak dijumpai kriptografi modern yang sudah berbentuk aplikasi layaknya ATM, *password* komputer dan E-commerce.

Konversi dari kalimat yang dibaca menjadi kelihatan tidak masuk akal dari pembuat pesan enkripsi membagi teknik pemecah sandi yang dibutuhkan untuk mengembalikan informasi asli dari kalimat yang tidak masuk akal menjadi kalimat yang bisa dibaca. Kriptografi sudah ada saat perang dunia I dan kedatangan komputer. Kriptografi modern ini didasari dengan ilmu matematis dan aplikasi komputer, algoritme yang didesain diasumsikan berdasarkan ketahanan komputasional sehingga sangat sulit untuk dipecahkan oleh musuh. Algoritme yang terdapat pada kriptografi sangat banyak dan berkembang setiap tahunnya.

Tujuan kriptografi adalah:

1. **Confidentiality**. Untuk melindungi identitas pemakai atau isi pesan agar tidak dapat dibaca oleh orang lain yang tidak berhak.
2. **Data Integrity**. Untuk melindungi pesan agar tidak diubah oleh orang lain.
3. **Availability**. Untuk menjamin ketersediaan sumber data.
4. **Authentication**. Untuk menjamin keaslian pesan.
5. **Non repudiation**. Membuktikan suatu pesan berasal dari seseorang, apabila ia menyangkal mengirim pesan tersebut.

Dalam dunia kriptografi, pesan yang akan dirahasiakan disebut *plaintext*. Pesan yang sudah diacak disebut *ciphertext*. Proses untuk mengkonversi *plaintext* menjadi *ciphertext* disebut enkripsi. Proses untuk mengembalikan *plaintext* dan *ciphertext* disebut deskripsi. Algoritme kriptografi *ciphers* adalah fungsi-fungsi matematika yang digunakan untuk melakukan enkripsi dan deskripsi. Diperlukan kunci yaitu kode untuk melakukan enkripsi dan deskripsi (Drs. Ario Suryo Kusumo, 2004).

2.2.2 Confidentiality

Confidensial merupakan salah satu aspek dari keamanan. *Confidensial* adalah sikap menjaga keamanan sebuah data agar tidak mampu diketahui oleh pihak ketiga. Dalam penelitian ini memfokuskan pada keamanan sebuah data yang diuji dengan angka yang mana ini menjadi dasar dari pada keamanan data selanjutnya. Proses keamanan data melalui beberapa tahap dengan menggunakan berbagai elemen. Elemen yang diperlukan untuk melakukan proses keamanan data seperti, data asli, kunci untuk merubah nilai data asli menjadi nilai data yang tidak masuk akal untuk dibaca oleh orang lain (Ariyus, 2008).

Pengamanan pada kriptografi dibedakan menjadi 2 bentuk yakni *symetric algorithm* dan *asymetric algorithm*. Kedua bentuk tersebut memiliki perbedaan dalam melakukan proses pengamanan. Setiap algoritme memiliki Kunci yang sesuai dengan parameter setiap algoritme yang dipakai. Jumlah kunci yang dibutuhkan juga menyesuaikan dengan algoritme enkripsi yang akan di pakai.

2.2.3 Symetrik Key

Simetrik key adalah algoritme enkripsi dan dekripsi yang menggunakan *key* sama dalam proses enkripsi dan dekripsi. Algoritme yang menggunakan *symetric key* dibedakan menjadi 2 kategori yaitu *Stream Cipher* dan *Block Cipher*. Perbedaan

dari kedua kategori tersebut hanya pada operasi penyandiannya. *Stream Cipher* menggunakan proses penyandian data berorientasi 1 bit atau 1 *byte* data. Sedangkan *Block Cipher* penyandiannya berorientasikan pada panjang bit atau *byte* data (perblok). Contoh algoritma yang menggunakan kunci simetris adalah DES, Blowfish, Twofish, MARS, IDEA, AES, SPECK (Komputer, 2013).

2.2.4 Asymmetric Key

Asymmetric key adalah algoritme enkripsi dan dekripsi yang menggunakan 2 *key* yang berbeda. *Key* pertama dinamakan *Public key* yang artinya semua orang bisa mengetahui nilai kunci yang akan dipakai. *Key* kedua dinamakan *Private key* kunci ini hanya disimpan untuk pribadi. Didalam algoritme ini setiap orang yang bertukar pesan akan memiliki kunci masing – masing. Sehingga pesan yang akan dikirimkan oleh pihak pertama kepada pihak kedua bisa dibaca dengan cara mendekripsikan dengan masing-masing kunci yang sudah dipegang. Algoritme yang terkenal yang menggunakan *Asymmetric key* adalah Algoritme RSA (Komputer, 2013).

2.2.5 Algoritme SPECK

Algoritme SPECK merupakan algoritme yang masih mempunyai keluarga dengan algoritme *lightweight block Cipher*. Algoritme ini di publikasikan oleh Nasional security Agency (NSA) pada bulan Juni 2013. Algoritme SPECK di gunakan untuk mengoptimalkan dalam suatu *software implementations*. SPECK mendukung berbagai *block* dan *key*. Sebuah *block* selalu terdapat 2 kata yang panjangnya 16, 24, 32, 48, atau 64 bit untuk ukuran katanya. *Key* yang sesuai adalah 2, 3, atau 4 kata. Didalam SPECK terdapat fungsi 2 putaran, menambahkan kata di sebelah kanan ke kata yang sebelah kiri kemudian di XOR kan kata sebelah kiri ke kata sebelah kanan (Ray Beaulieu dkk, 2015).

Tabel 2.1 Parameter Ukuran Block dan Key Serta Round

Ukuran Block (bit)	Ukuran Key (bit)	Round
2x16 = 32	4x16 = 64	22
2x24 = 48	3x24 = 72	22
	4x24 = 96	23
2x32 = 64	3x32 = 96	26
	4x32 = 128	27
2x48 = 96	2x48 = 96	28
	3x48 = 144	29
2x64 = 128	2x64 = 128	32
	3x64 = 192	33
	4x64 = 256	34

Pada algoritme SPECK untuk seluruh tipe *Block* dan *Key* memiliki persamaan dalam menggunakan operasi sistematika. Pada penelitian ini akan digunakan 3 algoritme SPECK yaitu SPECK *block size* 128 dan *key* 128, SPECK *block size* 128 dan *key* 128, SPECK *block size* 192 dan *key* 256. Karena pada varian pengujian yang diambil dari kesamaan dari panjang *plaintext* dengan varian panjang *key* yang berbeda. Proses yang dibutuhkan dalam algoritme SPECK berikut :

- Bitwise XOR, \oplus
- Bitwise AND, $\&$,
- Pergeseran melingkar ke kiri, S^i , dari j bits
- Pergeseran melingkar ke kanan, S^{-j} , dari j bits, dan
- Penambahan modular, $+$

Untuk proses *round* $k \in GF(2)^n$, kunci tergantung SIMON $2n$ *round function* merupakan dua tahap *Feistel map* $R_k : GF(2)^n \times GF(2)^n \rightarrow GF(2)^n \times GF(2)^n$ didefinisikan sebagai

$$R_k(x, y) = (y \oplus f(x) \oplus k, x) \tag{2.1}$$

Dimana, $f(x) = (Sx \& S^8x) \oplus S^2x$ dan k merupakan *round key*. Kunci independen SPECK $2n$ *round function* pada

$$R_k : GF(2)^n \times GF(2)^n \rightarrow GF(2)^n \times GF(2)^n \tag{2.2}$$

Didefinisikan sebagai

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k) \tag{2.3}$$

2.2.6 Raspberry Pi

Raspberry Pi sering disebut sebagai raspi, merupakan sebuah komputer papan tunggal yang berukuran kecil yang mampu digunakan untuk mengerjakan pekerjaan yang spesifik. Raspberry Pi adalah model paling awal Raspberry Pi generasi ketiga. Raspberry Pi juga memiliki spesifikasi sebagai berikut:

1. Broadcom BCM2837B0, Cortex-A53 (ARMv8) SoC 64-bit @ 1.4GHz.
2. 1Gb LPDDR2 SDRAM.
3. 2.4GHz dan 5GHz IEE 802.11.b/g /n/ ac LAN nirkabel, Bluetooth 4.2, BLE.
4. Gigabit Ethernet melalui USB 2.0 (throughput maksimum 300 Mbps).
5. Header GPIO 40-pin diperpanjang.
6. HDMI ukuran penuh.
7. 4 Port USB 2.0.
8. Port kamera CSI untuk menghubungkan kamera Raspberry Pi.
9. DSI menampilkan port untuk menghubungkan layar sentuh raspberri pi.
10. Output stereo 4-kutub dan port video komposit.
11. Port Micro SD untuk memuat sistem operasi dan menyimpan data.
12. Masukkan daya DC 5V/2,5A.
13. Dukungan power-over-Ethernet(PoE) (Matt Richardson, Shawn Wallace, 2012).



Gambar 2.1 Raspberry Pi

Sumber: <https://www.Raspberry.Pipi.org/>



2.2.7 Kruskal-Wallis

Kruskal-Wallis merupakan teknik statistik yang dapat digunakan untuk perhitungan statistik non-parametrik. Prosedur ini digunakan ketika data yang akan dibandingkan antara dua variabel yang diukur dari sampel yang berbeda (bebas), dan kelompok yang diperbandingkan itu lebih dari 2 kelompok (Junaidi, 2015). Uji Kruskal-Wallis yang sering disebut juga uji H, berkaitan dengan tiga atau lebih sampel acak yang independen dengan tujuan untuk mengetahui apakah sampel-sampel tersebut berasal dari populasi yang memiliki mean yang sama. Uji Kruskal-Wallis juga mengasumsikan varians yang sama, tetapi uji ini hanya mensyaratkan bahwa populasi-populasi yang dikaji bersifat kontinu dan mempunyai bentuk yang sama (bentuknya bisa miring kanan, bimodal, platikurtik, dll) (Herlina Budiono, 2014).

Rumus Kruskal-Wallis

$$H = \frac{12}{N(n+1)} \sum_{i=1}^k \frac{Ri^2}{ni} - 3(N + 1) \quad (2.4)$$

Keterangan :

H = nilai Kruskal-Wallis

K = perlakuan pada sampel

Ni = banyak sampel pengukuran pada perlakuan sampel ke i

N = banyak sampel keseluruhan

Ri = jumlah rangking sampel i diukur dari data keseluruhan

Variabel adalah konsep yang memiliki nilai. Variable dependen (variable terikat) merupakan variable yang dipengaruhi atau menjadi akibat karena adanya variabel bebas. Sedangkan variabel independen (variabel bebas) adalah variabel yang mempengaruhi perubahan atau timbulnya variabel dependen (Lubis, 2018).

Uji H data dapat dilakukan dengan menggunakan Kruskal-Wallis. Cara menafsir data dengan hipotesis seperti :

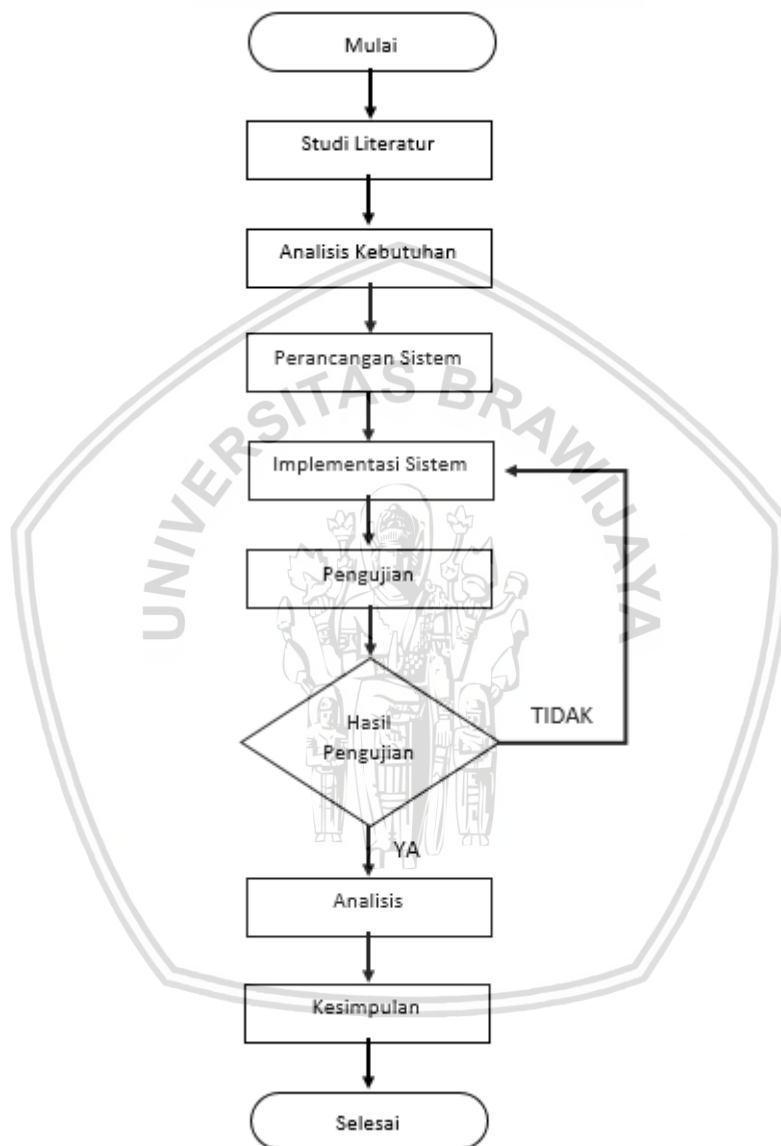
1. H₀ = tidak memiliki perbedaan antara kelompok satu dengan yang lainnya.
2. H₁ = memiliki perbedaan antara kelompok satu dengan yang lainnya.

Sig adalah nilai yang didapatkan dari proses perhitungan Kruskal-Wallis. Jika terdapat nilai sig ,000 maka yang harus dilakukan adalah melakukan uji *Post Hoc Tests*, dimana uji ini akan membandingkan antara kelompok untuk mengetahui letak perbedaan data yang signifikan. Hasil dapat diketahui dengan melihat nilai *Chi-Square*. Apabila nilai *Chi-Square* lebih besar maka dapat diputuskan bahwa data tersebut mengalami perbedaan yang signifikan (Salemba, 2014). Analisis statistik dilakukan Untuk data sebelum perlakuan dilakukan uji Kruskal-Wallis untuk melihat apakah terdapat perbedaan waktu dan jenis SPECK saat Pengujian dilakukan (Rizky Oktora Prihadini Putri dkk, 2014).

BAB 3 METODOLOGI

3.1 Studi Literatur

Bab ini menjelaskan mengenai metode yang digunakan dalam melakukan penelitian tentang implementasi algoritme SPECK pada raspberry pi. Tipe penelitian ini adalah implementasi yang bersifat objektif menggunakan mikrokontroler. langkah-langkah dari penelitian ini dapat dilihat pada gambar.



Gambar 3.1 Metodologi Penelitian

3.2 Studi Literatur

Studi literatur digunakan untuk menambah studi pustaka dan pengetahuan yang dilakukan dalam mengerjakan penulisan laporan dan penelitian. Studi literatur dilaksanakan dengan cara mengumpulkan teori dan pustaka yang berkaitan dengan penelitian ini meliputi:

1. Algoritme SPECK.
Algoritme enkripsi dan dekripsi yang akan diuji dalam penelitian ini. Algoritme ini diuji dengan berbagai *block* yang dimiliki oleh algoritme SPECK.
2. Raspberry Pi.
Perangkat keras yang memiliki fungsi layaknya komputer dengan fitur-fitur yang tertanam pada Raspberry Pi. Perangkat keras ini bekerja dengan daya *power* yang rendah.

3.3 Analisis Kebutuhan

Kebutuhan diperlukan untuk menganalisis apa saja yang dibutuhkan oleh sistem, sehingga sistem dapat berjalan sesuai dengan yang diharapkan. Berikut kebutuhan fungsional yang dibutuhkan oleh sistem dalam penelitian ini:

1. Sistem dapat menampilkan hasil enkripsi dan dekripsi.
2. Sistem mampu memasukkan masukan dengan menggunakan manual *input*.

3.3.2 Kebutuhan Perangkat Keras

kebutuhan perangkat keras yang digunakan untuk membuat sistem pada penelitian ini :

1. Raspberry Pi
Mini komputer yang digunakan untuk memproses program algoritme SPECK. Mini komputer ini dapat berjalan dengan daya 5V. Dengan berbagai fitur yang terdapat pada mini komputer tersebut. Raspberri juga memiliki RAM, CPU dan hampir menyerupai komputer semestinya.
2. Kabel LAN
Kabel yang akan menghubungkan Raspberri dengan laptop. Sehingga semua kegiatan dapat dilakukan pada laptop.
3. Laptop
Asus A455L merupakan laptop yang digunakan untuk menjalankan penelitian. Adapun spesifikasi dari laptop, *Core i3*, dengan *grapich* Invidia 930M, RAM 2 Gb

3.3.3 Kebutuhan Perangkat Lunak

Pada bagian ini menganalisa terkait kebutuhan perangkat lunak apasaja yang dibutuhkan oleh sistem :

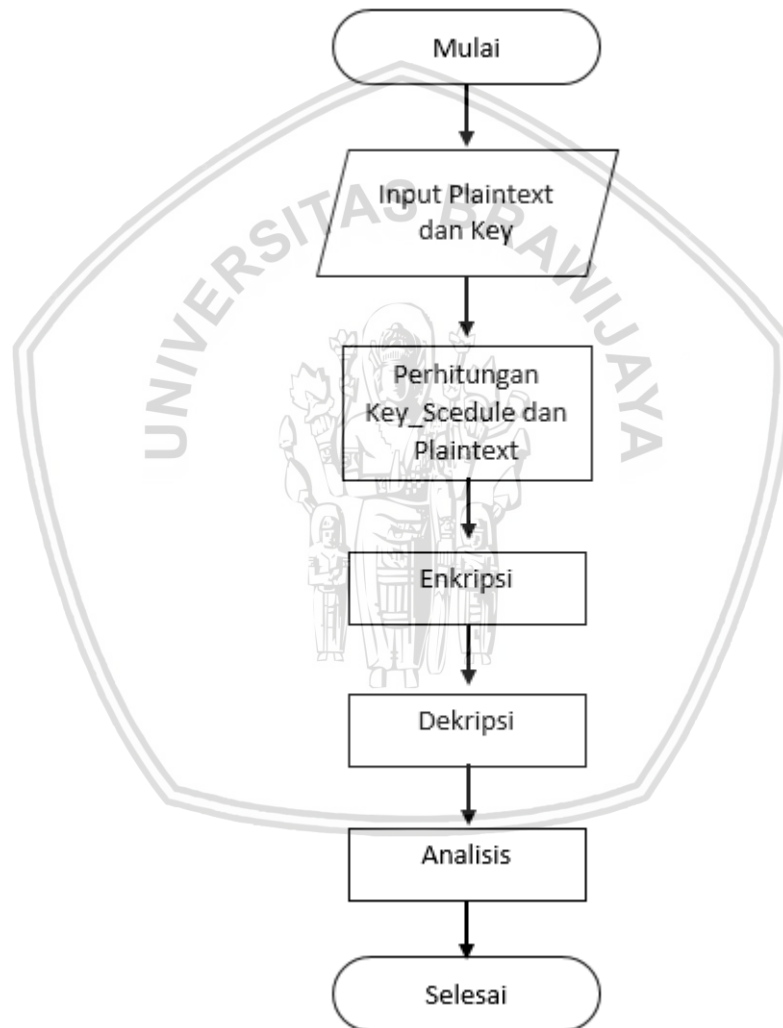
1. Raspbian
2. WinSCP
3. Sublimtext Editor 2
4. Python 2.7
5. TOP

3.4 Perancangan Sistem

Pada perancangan sistem ini terdapat meliputi perancangan pada perangkat lunak saja.

3.4.1 Perancangan Perangkat Lunak

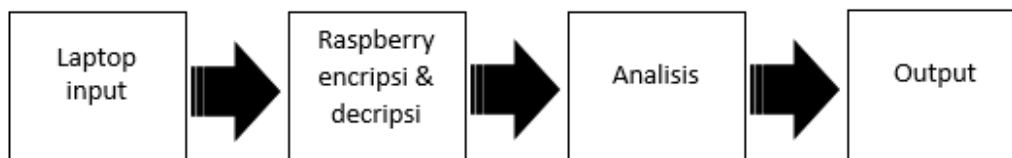
Pada perancangan perangkat lunak berupa algoritme yang digunakan sesuai dengan tujuan utama dari pembuatan sistem yakni pembuatan program algoritme enkripsi dan dekripsi dari SPECK.



Gambar 3.2 Diagram Alir Cara Kerja Sistem

3.5 Perancangan Hardware

Perancangan hardware dalam pembuatan implementasi algoritme SPECK ini meliputi perancangan sistem sebagai proses hasil enkripsi dan dekripsi yang dihubungkan dengan kabel LAN pada Raspberry Pi dan ditampilkan pada terminal *console* di *Putty*.



Gambar 3.3 Block Diagram Sistem

3.6 Pengujian dan Analisis Hasil

Pengujian dan analisis sistem dilakukan untuk mengetahui kinerja dan performa keseluruhan sistem yang telah dirancang sesuai dengan kebutuhan.

3.6.1 Pengujian Fungsional Hardware

3.6.1.1 Tujuan Pengujian

Tujuan pengujian untuk mengetahui performansi sistem yang dirancang dapat memenuhi kebutuhan yang melandasinya.

3.6.1.2 Prosedur Pengujian

1. Pengujian Raspberry Pi

Pengujian fungsional Raspberry Pi dilakukan dengan menghubungkan Raspberry Pi menggunakan kabel LAN yang ditancapkan pada *port* LAN pada laptop.

3.7 Kesimpulan

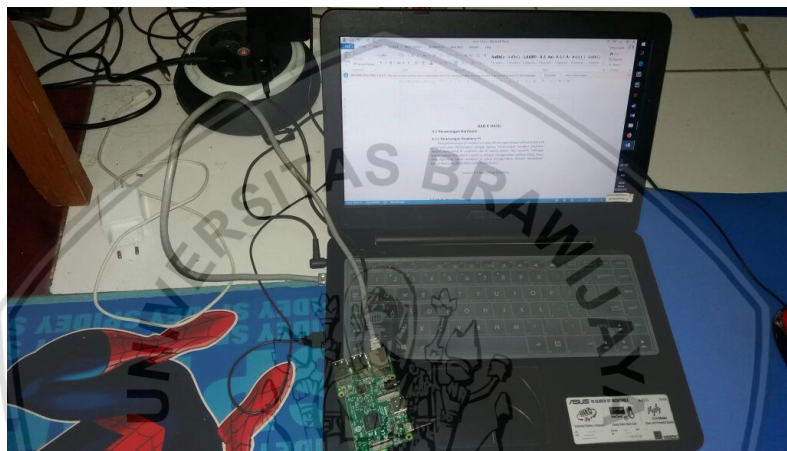
Kesimpulan didapatkan setelah melakukan perancangan, implementasi, pengujian dan analisis terhadap sistem. Kesimpulan ini ditentukan berdasarkan dari hasil pengujian dan analisis yang dibuat. Tujuan dari pembuatan kesimpulan ini adalah diharapkan dapat menjadi acuan dalam penelitian lain untuk mengembangkan penerapan algoritme SPECK pada Raspberry Pi dengan fungsi khusus.

BAB 4 HASIL

4.1 Implementasi *hardware*

4.1.1 Implementasi Raspberry Pi

Pada perancangan ini Raspberry Pi akan dihubungkan dengan sebuah kabel LAN yang kemudian dihubungkan dengan laptop. Perancangan tersebut dilakukan dengan *setting* IP Raspberry Pi dan IP laptop dalam satu *network*. Sehingga laptop mampu mengakses Raspberry Pi dengan menggunakan aplikasi *Putty*. Daya yang digunakan untuk Raspberry Pi cukup menggunakan *charger handphone* karena daya yang dibutuhkan tidak terlalu besar.



Gambar 4.1 Perancangan Raspberry Pi

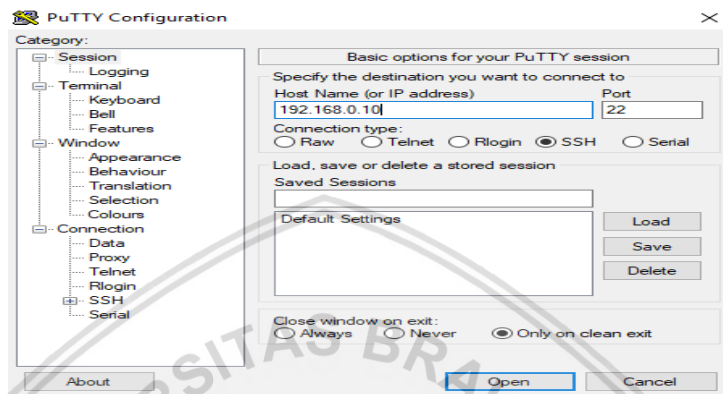
4.1.2 Prosedur Pengujian

Dalam pelaksanaan pengujian terdapat prosedur yang harus dilakukan. Prosedur ditujukan agar pengujian berjalan sesuai dengan tujuan dalam penelitian. Ada beberapa hal yang harus dilakukan sebagai berikut:

1. Pengujian pertama dilakukan terhadap Raspberry Pi apakah sudah berfungsi atau tidak.
2. Program dari algoritme SPECK berjalan sesuai dengan *test vektor*.
3. Jika program sudah sesuai maka dilakukan pengujian waktu dengan varian biner sebanyak 1024 kali dan juga mencatat Konsumsi RAM dan CPU yang digunakan.
4. Menganalisis perbedaan data waktu yang sudah didapatkan.
5. Menyimpulkan dari seluruh hasil yang sudah didapatkan.

4.1.3 Pengujian Raspberry Pi

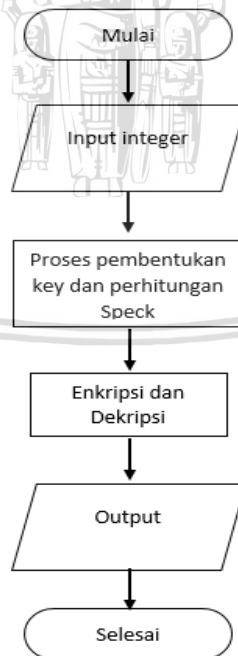
Pengaksesan Raspberry Pi ini dilakukan dengan menggunakan *software Putty*, yang diakses adalah alamat IP dari Raspberry Pi. Jika berhasil akan muncul tampilan *command line* untuk menjalankan data yang ada pada Raspberry Pi. Sebelum itu ip harus di *setting* agar menjadi satu *network* dari *port* LAN. Untuk port yang akan di SSH menggunakan *port* 22. Memasukkan *username* dan *password* Raspberry Pi untuk bisa menjalankan Raspberry Pi.



Gambar 4.2 Akses Raspberry Pi

4.2 Implementasi *software*

4.2.1 Perancangan Proses Enkripsi dan Dekripsi Data



Gambar 4.3 Diagram Alir Perancangan Perangkat Lunak Algoritme SPECK

Proses perancangan perangkat lunak untuk algoritme SPECK ditunjukkan pada gambar 5.2 diatas, hal ini dimaksudkan untuk pemrosesan data *input* mana yang akan di enkripsi dan didekripsi dengan algoritme SPECK.



4.2.2 Proses manualisasi enkripsi

Algoritme SPECK *block size 32bit key size 64bit*

Contoh Data Key1= 2, key2 = 3, key3 = 1, key4 = 4

Proses pencarian key

Round x = 3 dan y = 2 dengan K=0

- Pencarian x :
 - Merubah nilai 3 ke biner : 000000000000011
000000000000011 digeser kekiri (7) = 000011000000000
 - Merubah nilai 3 ke biner : 000000000000011
000000000000011 digeser kekanan (9) = 000011000000000
Hasil pergeseran pertama & kedua di-OR-kan menjadi x = 11000000000
 - $x = x + y = 11000000010$
 - $x < 65535$
 - $x = x ^ k = 11000000010$
 - $x = 1538$
- Pencarian y :
 - Merubah nilai 2 ke biner : 000000000000010
000000000000010 digeser kekiri (2) = 000000000001000
 - Merubah nilai 3 ke biner : 000000000000010
000000000000010 digeser kekanan (14) = 000000000001000
Hasil pergeseran pertama & kedua di-OR-kan menjadi x = 11000000000
 - $y = y ^ x = 11000001010$
 - $y = 1546$

Hasil dari key-1 = 1546

Proses enkripsi

Plaintext1 = 20, Plaintext2 = 40 key =2

- Pencarian x :
 - Merubah nilai 40 ke biner : 000000000101000
000000000101000 digeser kekiri (7) = 010100000000000
 - Merubah nilai 40 ke biner : 000000000101000
000000000101000 digeser kekanan (9) = 010100000000000
Hasil pergeseran pertama & kedua di-OR-kan menjadi x =
101000000000000
 - $x = x + y = 10100000010100$
 - $x < 65535$
 - $x = x ^ k = 10100000010110$
 - $x = 20502$
- Pencarian y :
 - Merubah nilai 20 ke biner : 00000000010100
00000000010100 digeser kekiri (2) = 000000001010000
 - Merubah nilai 20 ke biner : 00000000010100
00000000010100 digeser kekanan (14) = 000000001010000
Hasil pergeseran pertama & kedua di-OR-kan menjadi x = 1010000
 - $y = y ^ x = 10100001000110$
 - $y = 20550$



4.2.3 Implementasi Algoritme SPECK

Pada perancangan algoritme SPECK menjelaskan proses realisasi program untuk sistem enkripsi dan dekripsi untuk algoritme SPECK ini berdasarkan perancangan yang telah dilakukan sebelumnya. Pelaksanaan implementasi perangkat lunak ini sepenuhnya merupakan proses pengkodean yang dilakukan pada Raspberry Pi dengan menggunakan program *Putty* dan *sublimtext 2* untuk mempermudah pengkodean. Algoritme SPECK yang dipilih untuk diuji adalah algoritme SPECK *block size 128bit key size 128bit*, algoritme SPECK *block size 128bit key size 192bit*, algoritme SPECK *block size 128bit key size 256bit*. Karena pengujian ditujukan pada varian *plaintext* yang sama dan varian *key* yang berbeda. Pada Tabel 4.1, Tabel 4.2 dan Tabel 4.3 menunjukkan bahwa inialisasi program pertama kali adalah melakukan "*import datetime as dt*" untuk dapat menghitung waktu proses program selama berjalan sampai selesai. kemudian "*ROUNDS = 32*" digunakan untuk menginisialisasi jumlah *Round* yang dibutuhkan setiap algoritme SPECK.

Tabel 4.1 Kode program inialisasi SPECK *block size 128bit key size 128 bit*

No.	Kode Program
1	<code>import datetime as dt</code>
2	<code>ROUNDS = 32</code>

Tabel 4.2 Kode program inialisasi SPECK *block size 128bit key size 192 bit*

No.	Kode Program
1	<code>import datetime as dt</code>
2	<code>ROUNDS = 33</code>

Tabel 4.2 menunjukkan bahwa inialisasi program pertama kali adalah melakukan "*import datetime as dt*" untuk dapat menghitung waktu proses program selama berjalan sampai selesai. kemudian "*ROUNDS = 33*" digunakan untuk menginisialisasi jumlah *Round* yang dibutuhkan setiap algoritme SPECK.

Tabel 4.3 Kode program inialisasi SPECK *block size 128bit key size 256 bit*

No.	Kode Program
1	<code>import datetime as dt</code>
2	<code>ROUNDS = 34</code>

Tabel 4.3 menunjukkan bahwa inialisasi program pertama kali adalah melakukan "*import datetime as dt*" untuk dapat menghitung waktu proses program selama berjalan sampai selesai. kemudian "*ROUNDS = 34*" digunakan untuk menginisialisasi jumlah *Round* yang dibutuhkan setiap algoritme SPECK.

4.2.3.1 Implementasi kode program *input* pada Algoritme SPECK

Dalam pemasukkan *input* terdapat beberapa ketentuan yang harus dimasukkan, karena batasan dari bit yang disediakan dari algoritme SPECK. Jika *input* melebihi dari batas panjang bit maka program akan mengulang kembali *input* yang belum melebihi batas bit yang ditentukan. Dimana didalam Tabel 4.4 dijelaskan pada baris ke-1 merupakan main program. Baris ke-2 sampai ke-6 merupakan inialisasi *input* yang berbentuk *array* dengan jumlah sesuai dari algoritme SPECK yang digunakan. Baris ke-7 sampai baris ke-14 merupakan perulangan dan pembatasan *input* yang akan dimasukkan.

Tabel 4.4 Kode program *Input* SPECK *block size* 128bit *key size* 128 bit

No.	Kode Program
1	def main():
2	plaintext = [0] * 2
3	key = [0] * 2
4	ciphertext = [0] * 2
5	decrypted = [0] * 2
6	key_schedule = [0] * ROUNDS
7	for i in range(len(plaintext)):
8	while True:
9	tmp_in = raw_input('Plaintext ' + str(i + 1) + ': ')
10	tmp_in = int(tmp_in)
11	if 0 <= tmp_in <= 18446744073709551615:
12	plaintext[i] = tmp_in
13	break
14	

Tabel 4.5 Kode program *Input* SPECK *block size* 128bit *key size* 192 bit

No.	Kode Program
1	def main():
2	plaintext = [0] * 2
3	key = [0] * 3
4	ciphertext = [0] * 2
5	decrypted = [0] * 2
6	key_schedule = [0] * ROUNDS
7	for i in range(len(plaintext)):
8	while True:
9	tmp_in = raw_input('Plaintext ' + str(i + 1) + ': ')
10	tmp_in = int(tmp_in)
11	if 0 <= tmp_in <= 18446744073709551615:
12	plaintext[i] = tmp_in
13	break
14
15	

Tabel 4.5 dijelaskan pada baris ke-1 merupakan main program. Baris ke-2 sampai ke-6 merupakan inialisasi *input* yang berbentuk *array* dengan jumlah sesuai dari algoritme SPECK yang digunakan. Baris ke-7 sampai baris ke-14 merupakan perulangan dan pembatasan *input* yang akan dimasukkan.



Tabel 4.6 Kode program Round dan key SPECK *block size* 128bit *key size* 256bit

No.	Kode Program
1	def main():
2	plaintext = [0] * 2
3	key = [0] * 4
4	ciphertext = [0] * 2
5	decrypted = [0] * 2
6	key_schedule = [0] * ROUNDS
7	for i in range(len(plaintext)):
8	while True:
9	tmp_in = raw_input('Plaintext ' + str(i + 1) + ': ')
10	tmp_in = int(tmp_in)
11	if 0 <= tmp_in <= 18446744073709551615:
12	plaintext[i] = tmp_in
13	break
14
15	

Tabel 4.5 dijelaskan pada baris ke-1 merupakan main program. Baris ke-2 sampai ke-6 merupakan inialisasi *input* yang berbentuk *array* dengan jumlah sesuai dari algoritme SPECK yang digunakan. Baris ke-7 sampai baris ke-14 merupakan perulangan dan pembatasan *input* yang akan dimasukkan.

4.2.3.2 Implementasi kode program key pada Algoritme SPECK

Dalam melakukan implementasi program pencarian *key schedule* harus mengetahui sistem matika berjalannya algoritme SPECK. Proses *key schedule* akan berjalan ketika beberapa fungsi terpanggil seperti fungsi *circular bit*, ini digunakan untuk proses perhitungan didalam pembentukan *key schedule*. Pada baris ke-1 dan ke-3 merupakan fungsi dari pengembalian dan pembentukan pertama kali *key*. Baris ke-2 dan ke-3 merupakan perhitungan yang dibutuhkan mengembalikan nilai *key* saat pertama kali dibentuk. Baris ke-4 merupakan proses perputaran bit dengan *circular*. Baris ke-5 sampai baris ke-8 merupakan perhitungan pergeseran bit. Baris ke-11 sampai ke-14 merupakan pembuatan *key schedule* dengan memasukkan nilai *key* dari *input* yang kemudian dilakukan secara berulang-ulang sesuai dengan panjang *Round*.

Tabel 4.7 Kode program key SPECK *block size* 128bit *key size* 128bit

No.	Kode Program
1	def reversed_SPECK_round(x, y, k):
2	y ^= x
3	y = right_circular_shift(y, 3) left_circular_shift(y, 64 -
4	3)
5	x ^= k
6	x -= y
7	while x < 0:
8	x += (18446744073709551615 + 1)
9
10	def SPECK_setup(key, key_schedule):
11	b = key[0]
12	a = key[1]
13	key_schedule[0] = b
14	for i in range(ROUNDS - 1):
15



Tabel 4.8 Kode program *key* SPECK *block size* 128bit *key size* 192bit

No.	Kode Program
1	<code>def reversed_SPECK_round(x, y, k):</code>
2	<code> y ^= x</code>
3	<code> y = right_circular_shift(y, 3) left_circular_shift(y, 64 -</code>
4	<code>3)</code>
5	<code> x ^= k</code>
6	<code> x -= y</code>
7	<code> while x < 0:</code>
8	<code> x += (18446744073709551615 + 1)</code>
9	<code> </code>
10	<code>def SPECK_setup(key, key_schedule):</code>
11	<code> b = key[0]</code>
12	<code> a0 = key[1]</code>
13	<code> a1 = key[2]</code>
14	<code> a2 = key[3]</code>
15	<code> key_schedule[0] = b</code>
16	<code> for i in range(ROUNDS - 1):</code>
17	<code> </code>

Pada Tabel 4.8 baris ke-1 dan ke-3 merupakan fungsi dari pengembalian dan pembentukan pertama kali *key*. Baris ke-2 dan ke-3 merupakan perhitungan yang dibutuhkan mengembalikan nilai *key* saat pertama kali dibentuk. Baris ke-4 merupakan proses perputaran bit dengan *circular*. Baris ke-5 sampai baris ke-8 merupakan perhitungan pergeseran bit. Baris ke-11 sampai ke-16 merupakan pembuatan *key schedule* dengan memasukkan nilai *key* dari *input* yang kemudian dilakukan secara berulang-ulang sesuai dengan panjang *Round*.

Tabel 4.9 Kode program *key* SPECK *block size* 128bit *key size* 256bit

No.	Kode Program
1	<code>def reversed_SPECK_round(x, y, k):</code>
2	<code> y ^= x</code>
3	<code> y = right_circular_shift(y, 3) left_circular_shift(y, 64 -</code>
4	<code>3)</code>
5	<code> x ^= k</code>
6	<code> x -= y</code>
7	<code> while x < 0:</code>
8	<code> x += (18446744073709551615 + 1)</code>
9	<code> </code>
10	<code>def SPECK_setup(key, key_schedule):</code>
11	<code> b = key[0]</code>
12	<code> a0 = key[1]</code>
13	<code> a1 = key[2]</code>
14	<code> key_schedule[0] = b</code>
15	<code> for i in range(ROUNDS - 1):</code>
16	<code> </code>

Pada Tabel 4.9 baris ke-1 dan ke-3 merupakan fungsi dari pengembalian dan pembentukan pertama kali *key*. Baris ke-2 dan ke-3 merupakan perhitungan yang dibutuhkan mengembalikan nilai *key* saat pertama kali dibentuk. Baris ke-4 merupakan proses perputaran bit dengan *circular*. Baris ke-5 sampai baris ke-8 merupakan perhitungan pergeseran bit. Baris ke-11 sampai ke-15 merupakan pembuatan *key schedule* dengan memasukkan nilai *key* dari *input* yang kemudian dilakukan secara berulang-ulang sesuai dengan panjang *Round*.



4.2.3.3 Implementasi kode program proses enkripsi dan dekripsi

Proses pada enkripsi dan dekripsi ini merupakan gabungan antara semua elemen program yang sudah dibuat. Elemen – elemen yang dibutuhkan saat enkripsi dan dekripsi adalah hasil dari *key* yang sudah diolah kemudian digabungkan dengan hasil perhitungan *plaintext*. Baris ke-1 dan baris ke-10 merupakan fungsi dari proses enkripsi dan dekripsi. baris ke-2 sampai baris ke-9 merupakan penempatan hasil dari enkripsi untuk *plaintext* dan *Ciphertext*. Pada baris ke-11 sampai baris ke-18 merupakan penempatan hasil dari dekripsi untuk *plaintext* dan *Ciphertext*.

Tabel 4.10 Kode program enkripsi, dekripsi SPECK *block* 128bit *key* 128bit

No.	Kode Program
1	def SPECK_encrypt(plaintext, key_schedule, ciphertext):
2	ciphertext[0] = plaintext[0]
3	ciphertext[1] = plaintext[1]
4	for i in range(ROUNDS):
5	tmp_res = SPECK_round(ciphertext[1], ciphertext[0],
6	key_schedule[i])
7	ciphertext[1] = tmp_res[0]
8	ciphertext[0] = tmp_res[1]
9	return ciphertext
10	def SPECK_decrypt(ciphertext, key_schedule, decrypted):
11	decrypted[0] = ciphertext[0]
12	decrypted[1] = ciphertext[1]
13	for i in reversed(range(ROUNDS)):
14	tmp_res = reversed_SPECK_round(decrypted[1], decrypted[0],
15	key_schedule[i])
16	decrypted[1] = tmp_res[0]
17	decrypted[0] = tmp_res[1]
18	return decrypted

Tabel 4.11 Kode program enkripsi, dekripsi SPECK *block* 128bit *key* 192bit

No.	Kode Program
1	def SPECK_encrypt(plaintext, key_schedule, ciphertext):
2	ciphertext[0] = plaintext[0]
3	ciphertext[1] = plaintext[1]
4	for i in range(ROUNDS):
5	tmp_res = SPECK_round(ciphertext[1], ciphertext[0],
6	key_schedule[i])
7	ciphertext[1] = tmp_res[0]
8	ciphertext[0] = tmp_res[1]
9	return ciphertext
10	def SPECK_decrypt(ciphertext, key_schedule, decrypted):
11	decrypted[0] = ciphertext[0]
12	decrypted[1] = ciphertext[1]
13	for i in reversed(range(ROUNDS)):
14	tmp_res = reversed_SPECK_round(decrypted[1], decrypted[0],
15	key_schedule[i])
16	decrypted[1] = tmp_res[0]
17	decrypted[0] = tmp_res[1]
18	return decrypted

Pada Tabel 4.11 Baris ke-1 dan baris ke-10 merupakan fungsi dari proses enkripsi dan dekripsi. baris ke-2 sampai baris ke-9 merupakan penempatan hasil dari enkripsi untuk *plaintext* dan *Ciphertext*. Pada baris ke-11 sampai baris ke-18 merupakan penempatan hasil dari dekripsi untuk *plaintext* dan *Ciphertext*.



Tabel 4.12 Kode program enkripsi, dekripsi SPECK *block* 128bit *key* 256bit

No.	Kode Program
1	def SPECK_encrypt(plaintext, key_schedule, ciphertext):
2	ciphertext[0] = plaintext[0]
3	ciphertext[1] = plaintext[1]
4	for i in range(ROUNDS):
5	tmp_res = SPECK_round(ciphertext[1], ciphertext[0],
6	key_schedule[i])
7	ciphertext[1] = tmp_res[0]
8	ciphertext[0] = tmp_res[1]
9	return ciphertext
10	def SPECK_decrypt(ciphertext, key_schedule, decrypted):
11	decrypted[0] = ciphertext[0]
12	decrypted[1] = ciphertext[1]
13	for i in reversed(range(ROUNDS)):
14	tmp_res = reversed_SPECK_round(decrypted[1], decrypted[0],
15	key_schedule[i])
16	decrypted[1] = tmp_res[0]
17	decrypted[0] = tmp_res[1]
18	return decrypted

Pada Tabel 4.12 Baris ke-1 dan baris ke-10 merupakan fungsi dari proses enkripsi dan dekripsi. baris ke-2 sampai baris ke-9 merupakan penempatan hasil dari enkripsi untuk *plaintext* dan *Ciphertext*. Pada baris ke-11 sampai baris ke-18 merupakan penempatan hasil dari dekripsi untuk *plaintext* dan *Ciphertext*.

4.2.3.4 Implementasi kode putaran bit Algoritme SPECK

Putaran bit dalam Algoritme SPECK ini merupakan putaran yang *circular* yang artinya bit yang digeser tidak menambah atau mengurangi jumlah bit sebelumnya. Putaran ini dilakukan untuk merubah nilai dari masukkan yang nantinya akan disatukan kembali menjadi *Ciphertext*. Proses putaran ini dilakukan ketika *input* sudah diubah menjadi biner. Baris ke-1 dan baris ke-8 merupakan fungsi dari putaran biner kekiri dan kekanan. Baris ke-2 sampai baris ke-6 merupakan pergeseran bit ke kiri sesuai dengan aturan pada Algoritme SPECK. Baris ke-9 sampai ke-12 merupakan pergeseran bit ke kanan sesuai dengan aturan pada Algoritme SPECK.

Tabel 4.13 Kode program *circular* bit SPECK *block size* 128bit *key size* 128bit

No.	Kode Program
1	def left_circular_shift(value, shift):
2	tmp_bin = bin(value)[2:]
3	binary = ['0'] * 64
4	index = 63
5	for i in reversed(tmp_bin):
6	binary[index] = i
7
8	def right_circular_shift(value, shift):
9	tmp_bin = bin(value)[2:]
10	binary = ['0'] * 64
11	index = 63
12	for i in reversed(tmp_bin):
13



Tabel 4.14 Kode program *circular* bit SPECK *block size* 128bit *key size* 192bit

No.	Kode Program
1	def left_circular_shift(value, shift):
2	tmp_bin = bin(value)[2:]
3	binary = ['0'] * 64
4	index = 63
5	for i in reversed(tmp_bin):
6	binary[index] = i
7
8	def right_circular_shift(value, shift):
9	tmp_bin = bin(value)[2:]
10	binary = ['0'] * 64
11	index = 63
12	for i in reversed(tmp_bin):
13

Tabel 4.14 Baris ke-1 dan baris ke-8 merupakan fungsi dari putaran biner kekiri dan kekanan. Baris ke-2 sampai baris ke-6 merupakan pergeseran bit ke kiri sesuai dengan aturan pada Algoritme SPECK. Baris ke-9 sampai ke-12 merupakan pergeseran bit ke kanan sesuai dengan aturan pada Algoritme SPECK.

Tabel 4.15 Kode program *circular* bit SPECK *block size* 128bit *key size* 256bit

No.	Kode Program
1	def left_circular_shift(value, shift):
2	tmp_bin = bin(value)[2:]
3	binary = ['0'] * 64
4	index = 63
5	for i in reversed(tmp_bin):
6	binary[index] = i
7
8	def right_circular_shift(value, shift):
9	tmp_bin = bin(value)[2:]
10	binary = ['0'] * 64
11	index = 63
12	for i in reversed(tmp_bin):
13

Tabel 4.14 Baris ke-1 dan baris ke-8 merupakan fungsi dari putaran biner kekiri dan kekanan. Baris ke-2 sampai baris ke-6 merupakan pergeseran bit ke kiri sesuai dengan aturan pada Algoritme SPECK. Baris ke-9 sampai ke-12 merupakan pergeseran bit ke kanan sesuai dengan aturan pada Algoritme SPECK.



BAB 5 PEMBAHASAN

5.1 Hasil Pengujian

5.1.1 Hasil pengujian *Test Vektor*

Pengujian *test vektor* dilakukan untuk mengetahui algoritme SPECK sudah benar dalam melakukan proses enkripsi dan dekripsi. *Test vektor* yang digunakan milik NSA dengan memasukkan awal *input* angka heksa yang kemudian dijadikan desimal secara manual. Hasil dari pengujian ini sebagai berikut:

5.1.1.1 *Test Vektor*

Dalam *test vektor* ini *key 1*, *key 2*, *key 3*, *key 4* merupakan masukan masukan dari *test vektor* dengan bentuk bilangan Heksa. Kemudian diubah dengan cara manual sehingga nilai heksa tersebut menjadi angka Desimal. *Plaintext 1*, *Plaintext 2* merupakan masukan yang sudah ditetapkan dalam paper yang kemudian diubah dengan cara manual sehingga menjadi angka desimal. *Ciphertext 1* dan *Ciphertext 2* adalah hasil proses enkripsi dari paper sebagai rujukan bahwa hasil dari program yang dijalankan sesuai dengan *Ciphertext* sebagai berikut:

1. SPECK *block size 128 key size 128*

<i>Key 1</i>	: 0x0706050403020100	decimal = 506097522914230528
<i>Key 2</i>	: 0x0f0e0d0c0b0a0908	decimal = 1084818905618843912
<i>Plaintext 1</i>	: 0x7469206564616d20	decimal = 8388271400802151712
<i>Paintext 2</i>	: 0x6c61766975716520	decimal = 7809653424151160096
<i>Ciphertext 1</i>	: 0x7860fedf5c570d18	
<i>Ciphertext 2</i>	: a65d985179783265	

2. SPECK *block size 128 key size 192*

<i>Key 1</i>	: 0x0706050403020100	decimal = 506097522914230528
<i>Key 2</i>	: 0x0f0e0d0c0b0a0908	decimal = 1084818905618843912
<i>Key 3</i>	: 0x1716151413121110	decimal = 1663540288323457296
<i>Plaintext 1</i>	: 0x43206f7420746e65	decimal = 4836988544347303525
<i>Paintext 2</i>	: 0x7261482066656968	decimal = 8241948097058793832
<i>Ciphertext 1</i>	: 0xf9bc185de03c1886	
<i>Ciphertext 2</i>	: 0x1be4cf3a13135566	

3. SPECK *block size 128 key size 256*

<i>Key 1</i>	: 0x0706050403020100	decimal = 506097522914230528
<i>Key 2</i>	: 0x0f0e0d0c0b0a0908	decimal = 1084818905618843912
<i>Key 3</i>	: 0x1716151413121110	decimal = 1663540288323457296
<i>Key 4</i>	: 0x1f1e1d1c1b1a1918	decimal = 2242261671028070680
<i>Plaintext 1</i>	: 0x202e72656e6f6f70	decimal = 2318916638112444272
<i>Paintext 2</i>	: 0x65736f6874206e49	decimal = 7310309114568011337
<i>Ciphertext 1</i>	: 0x4eeeb48d9c188f43	
<i>Ciphertext 2</i>	: 0x4109010405c0f53e	

Hasil pengujian sebagai berikut:

Dalam hasil pengujian *test vektor* ini menghasilkan *output* dua macam *output* yang pertama *output* dengan bilangan desimal dan kedua *output* dengan bilangan heksa. *Plaintext 1* dan *Plaintext 2* merupakan masukan dari angka awal yang belum diproses yang didapat dari *test vektor* diatas. Kemudian *Key 1*, *key 2*, *key 3*, *key 4* merupakan *input* key yang dibutuhkan dalam proses enkripsi dan dekripsi, nilai *key* didapat dari *test vektor* diatas. *Ciphertext 1*, *Ciphertext 2* ini merupakan hasil untuk mengetahui apakah program sudah berjalan sesuai dengan *test vektor*, maka nilai *Ciphertext* program akan sama dengan nilai *Ciphertext* pada *test vektor*. *Decryptext* ini hasil dari enkripsi yang dibalik sehingga kita tahu hasil dekripsinya sesuai dengan *plaintext* yang dimasukkan. Hasil output program sebagai berikut:

Tabel 5.1 Hasil Pengujian Pada Raspberry Pi

SPECK 32/64	SPECK 48/72	SPECK 48/96
Plaintext 1: 26956	Plaintext 1: 7102834	Plaintext 1: 1936613733
Plaintext 2: 25972	Plaintext 2: 2128236	Plaintext 2: 1952532000
Key 1: 256	Key 1: 131328	Key 1: 50462976
Key 2: 2312	Key 2: 657672	Key 2: 185207048
Key 3: 4368	Key 3: 1184016	Key 3: 319951120
Key 4: 6424		
Ciphertext 1: 17138	Ciphertext 1: 3693276	Ciphertext 1: 1098224748
Ciphertext 2: 43112	Ciphertext 2: 12601765	Ciphertext 2: 2675528428
Decrypted 1: 26956	Decrypted 1: 7102834	Decrypted 1: 1936613733
Decrypted 2: 25972	Decrypted 2: 2128236	Decrypted 2: 1952532000
Dalam bentuk heksa:	Dalam bentuk heksa:	Dalam bentuk heksa:
Plaintext 1: 0x694c	Plaintext 1: 0x6c6172	Plaintext 1: 0x736e6165
Plaintext 2: 0x6574	Plaintext 2: 0x20796c	Plaintext 2: 0x74614620
Key 1: 0x100	Key 1: 0x20100	Key 1: 0x3020100
Key 2: 0x908	Key 2: 0xa0908	Key 2: 0xb0a0908
Key 3: 0x1110	Key 3: 0x121110	Key 3: 0x13121110
Key 4: 0x1918		
Ciphertext 1: 0x42f2	Ciphertext 1: 0x385adc	Ciphertext 1: 0x4175946cL
Ciphertext 2: 0xa868	Ciphertext 2: 0xc049a5	Ciphertext 2: 0x9f7952ecL
Decrypted 1: 0x694c	Decrypted 1: 0x6c6172	Decrypted 1: 0x736e6165
Decrypted 2: 0x6574	Decrypted 2: 0x20796c	Decrypted 2: 0x74614620

Dari Tabel 5.1 merupakan hasil menjalankan program pada Raspberry Pi yang menghasilkan *output* demikian.



Tabel 5.2 Hasil Pengujian Pada Raspberry Pi

SPECK 64/96	SPECK 64/128	SPECK 96/96
Plaintext 1: 1936613733 Plaintext 2: 1952532000 Key 1: 50462976 Key 2: 185207048 Key 3: 319951120 Ciphertext 1: 1098224748 Ciphertext 2: 2675528428 Decrypted 1: 1936613733 Decrypted 2: 1952532000 Dalam bentuk heksa: Plaintext 1: 0x736e6165 Plaintext 2: 0x74614620 Key 1: 0x3020100 Key 2: 0xb0a0908 Key 3: 0x13121110 Ciphertext 1: 0x4175946cL Ciphertext 2: 0x9f7952ecL Decrypted 1: 0x736e6165 Decrypted 2: 0x74614620	Plaintext 1: 1953841965 Plaintext 2: 997352820 Key 1: 50462976 Key 2: 185207048 Key 3: 319951120 Key 4: 454695192 Ciphertext 1: 1162740363 Ciphertext 2: 2356127048 Decrypted 1: 1953841965 Decrypted 2: 997352820 Dalam bentuk hex: Plaintext 1: 0x7475432d Plaintext 2: 0x3b726574 Key 1: 0x3020100 Key 2: 0xb0a0908 Key 3: 0x13121110 Key 4: 0x1b1a1918 Ciphertext 1: 0x454e028bL Ciphertext 2: 0x8c6fa548L Decrypted 1: 0x7475432d Decrypted 2: 0x3b726574	Plaintext 1: 111494690993440 Plaintext 2: 111563644608556 Key 1: 5514788471040 Key 2: 14345375975688 Ciphertext 1: 108567622285738 Ciphertext 2: 174053711901048 Decrypted 1: 111494690993440 Decrypted 2: 111563644608556 Dalam bentuk heksa: Plaintext 1: 0x656761737520L Plaintext 2: 0x65776f68202cL Key 1: 0x50403020100L Key 2: 0xd0c0b0a0908L Ciphertext 1: 0x62bdde8f79aaL Ciphertext 2: 0x9e4d09ab7178L Decrypted 1: 0x656761737520L Decrypted 2: 0x65776f68202cL

Dari Tabel 5.2 merupakan hasil menjalankan program pada Raspberry Pi yang menghasilkan *output* demikian.

Tabel 5.3 Hasil Pengujian Pada Raspberry Pi

SPECK 96/144	SPECK 128/128
Plaintext 1: 115586905564534 Plaintext 2: 111520595058798 Key 1: 5514788471040 Key 2: 14345375975688 Key 3: 23175963480336 Ciphertext 1: 135120747310822 Ciphertext 2: 48322952962698 Decrypted 1: 115586905564534 Decrypted 2: 111520595058798 Dalam bentuk heksa: Plaintext 1: 0x69202c726576L Plaintext 2: 0x656d6974206eL Key 1: 0x50403020100L Key 2: 0xd0c0b0a0908L Key 3: 0x151413121110L Ciphertext 1: 0x7ae440252ee6L Ciphertext 2: 0x2bf31072228aL Decrypted 1: 0x69202c726576L Decrypted 2: 0x656d6974206eL	Plaintext 1: 8388271400802151712 Plaintext 2: 7809653424151160096 Key 1: 506097522914230528 Key 2: 1084818905618843912 Ciphertext 1: 8674213117595946264 Ciphertext 2: 11987905258827821669 Decrypted 1: 8388271400802151712 Decrypted 2: 7809653424151160096 Dalam bentuk heksa: Plaintext 1: 0x7469206564616d20L Plaintext 2: 0x6c61766975716520L Key 1: 0x706050403020100L Key 2: 0xf0e0d0c0b0a0908L Ciphertext 1: 0x7860fedf5c570d18L Ciphertext 2: 0xa65d985179783265L Decrypted 1: 0x7469206564616d20L Decrypted 2: 0x6c61766975716520L



Tabel 5.4 Hasil Pengujian Pada Raspberry Pi

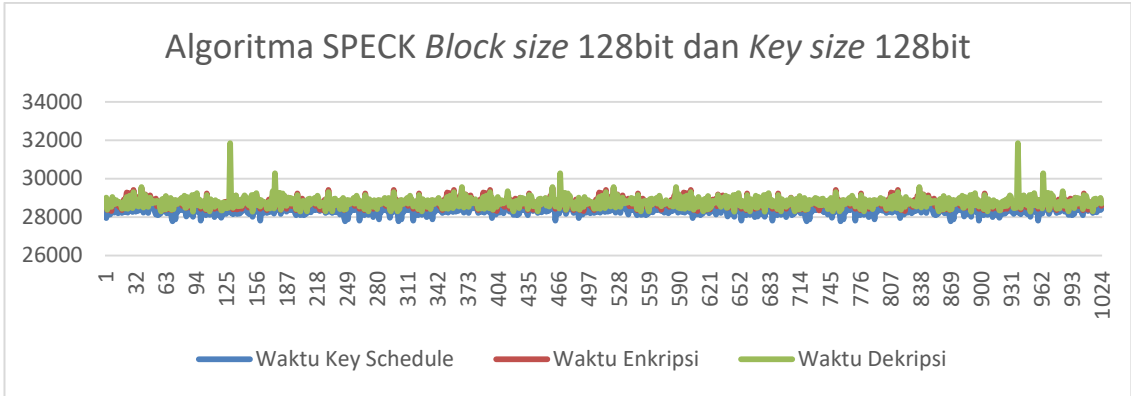
SPECK 128/192	SPECK 128/256
Plaintext 1: 2318916638112444272	Plaintext 1: 4836988544347303525
Plaintext 2: 7310309114568011337	Plaintext 2: 8241948097058793832
Key 1: 506097522914230528	Key 1: 506097522914230528
Key 2: 1084818905618843912	Key 2: 1084818905618843912
Key 3: 1663540288323457296	Key 3: 1663540288323457296
Key 4: 2242261671028070680	Ciphertext 1: 17995285002538719366
Ciphertext 1: 5687681899717758787	Ciphertext 2: 2009959182049170790
Ciphertext 2: 4686278004043740478	Decrypted 1: 4836988544347303525
Decrypted 1: 2318916638112444272	Decrypted 2: 8241948097058793832
Decrypted 2: 7310309114568011337	Dalam bentuk heksa:
Dalam bentuk heksa:	Plaintext 1: 0x43206f7420746e65L
Plaintext 1: 0x202e72656e6f6f70L	Plaintext 2: 0x7261482066656968L
Plaintext 2: 0x65736f6874206e49L	Key 1: 0x706050403020100L
Key 1: 0x706050403020100L	Key 2: 0xf0e0d0c0b0a0908L
Key 2: 0xf0e0d0c0b0a0908L	Key 3: 0x1716151413121110L
Key 3: 0x1716151413121110L	Key 4: 0x1f1e1d1c1b1a1918L
Key 4: 0x1f1e1d1c1b1a1918L	Ciphertext 1: 0xf9bc185de03c1886L
Ciphertext 1: 0x4eeeb48d9c188f43L	Ciphertext 2: 0x1be4cf3a13135566L
Ciphertext 2: 0x4109010405c0f53eL	Decrypted 1: 0x43206f7420746e65L
Decrypted 1: 0x202e72656e6f6f70L	Decrypted 2: 0x7261482066656968L
Decrypted 2: 0x65736f6874206e49L	

Dari Tabel 5.3 dan Tabel 5.4 merupakan hasil menjalankan program pada Raspberry Pi yang menghasilkan output demikian.

5.1.2 Hasil Analisis Waktu Algoritme SPECK

Pada pengujian waktu ini menggunakan metode perhitungan waktu dilakukan dengan cara menambahkan kode program yang sudah disediakan oleh pemrograman *python* dengan menggunakan *library Datetime* dan kode yang dituliskan menjadi satu dengan kode algoritme SPECK maka program untuk mengukur waktu proses enkripsi, dekripsi dan *key schedule* akan berjalan secara otomatis dan waktu akan ditampilkan pada *output*. Hasil dari pengujian waktu enkripsi, dekripsi dan proses pembuatan *key schedule*. Pengujian dilakukan sebanyak 1024 kali dalam satu tipe algoritme SPECK. Setiap menjalankan program sebanyak 1024 kali menghasilkan waktu yang berbeda – beda dalam satuan waktu *microsecond*. Hal ini disebabkan karena panjang *input* yang dimasukkan, semakin panjang bit dari *input* maka waktu yang dibutuhkan akan bertambah. Sehingga waktu yang dihasilkan pada setiap menjalankan program menjadi berbeda – beda. Sehingga didapatkan hasil dengan digambarkan pada grafik berikut:



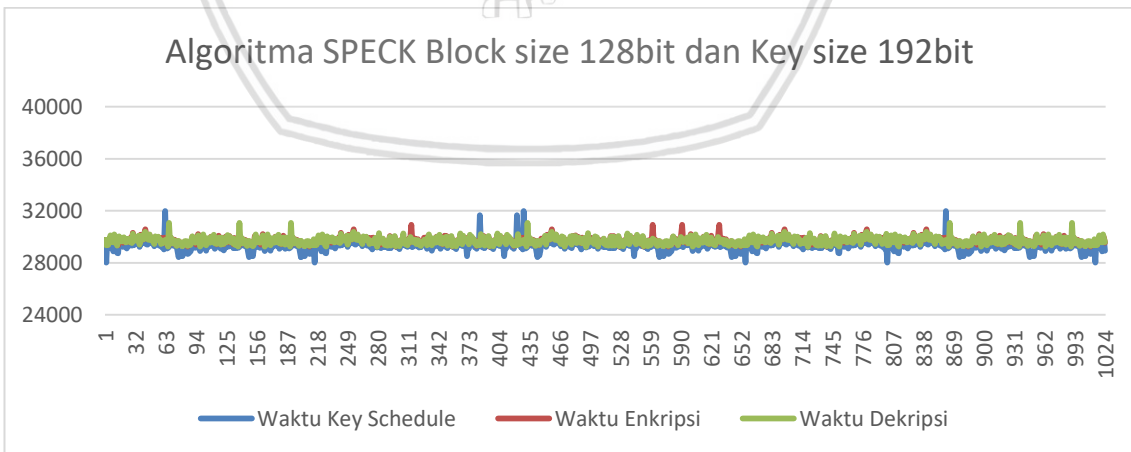


Gambar 5.1 Grafik Waktu Algoritme SPECK

Gambar 5.8 untuk melihat perbedaan waktu pengujian pemrosesan waktu *key schedule*, waktu enkripsi dan waktu dekripsi algoritme SPECK *block size* 128bit *key size* 128bit, maka dilakukan *test* menjalankan program sebanyak 1024 kali pada Raspberry Pi. Disebelah kanan grafik merupakan waktu yang dibutuhkan untuk setiap menjalankan program, dibawah grafik adalah banyaknya pengujian yang dilakukan mencapai 1024 pengujian. Sehingga menghasilkan bentuk grafik sedemikian rupa yang menandakan bahwa waktu untuk setiap proses algoritme SPECK terdapat perbedaan. Hasil dari pengujian ini juga mendapatkan rata – rata waktu yang dapat dilihat pada Tabel 5.12 berikut:

Tabel 5.5 Rata – rata waktu algoritma SPECK

Rata-rata Waktu <i>key schedule</i>	28401 μ s
rata-rata waktu enkripsi	28740 μ s
rata-rata waktu dekripsi	28779 μ s
total waktu rata-rata	85920 μ s



Gambar 5.2 Grafik Waktu Algoritme SPECK

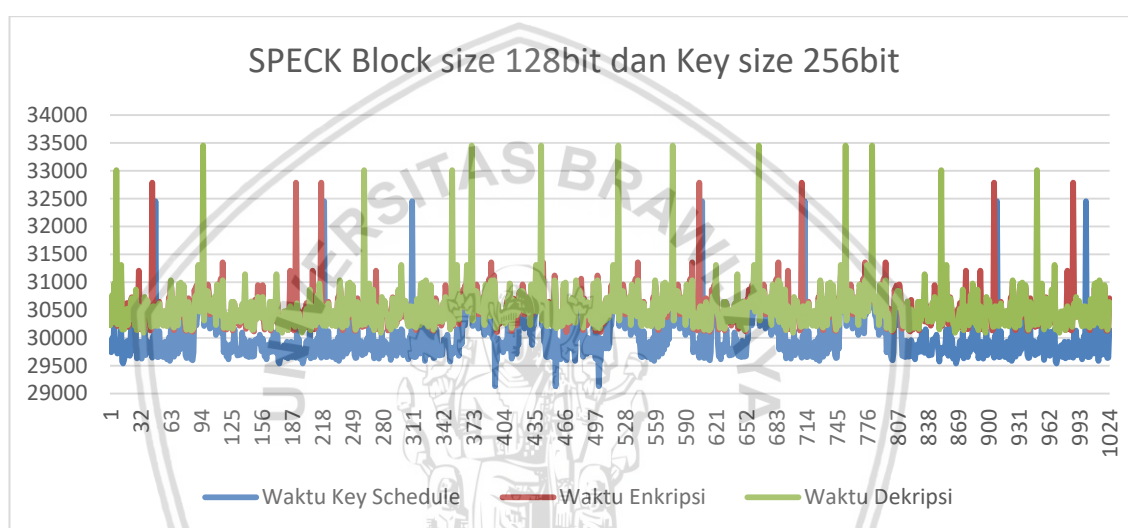
Gambar 5.9 untuk melihat perbedaan waktu pengujian pemrosesan waktu *key schedule*, waktu enkripsi dan waktu dekripsi algoritme SPECK *block size* 128bit *key size* 192bit, maka dilakukan *test* menjalankan program sebanyak 1024 kali pada Raspberry Pi. Disebelah kanan grafik merupakan waktu yang dibutuhkan



untuk setiap menjalankan program, dibawah grafik adalah banyaknya pengujian yang dilakukan mencapai 1024 pengujian. Sehingga menghasilkan bentuk grafik sedemikian rupa yang menandakan bahwa waktu untuk setiap proses algoritme SPECK terdapat perbedaan. Hasil dari pengujian ini juga mendapatkan rata – rata waktu yang dapat dilihat pada Tabel 5.13 berikut:

Tabel 5.6 Rata – rata waktu algoritma SPECK

Rata-rata Waktu <i>key schedule</i>	29360 μ s
rata-rata waktu enkripsi	29706 μ s
rata-rata waktu dekripsi	29695 μ s
total waktu rata-rata	88761 μ s



Gambar 5.3 Grafik Waktu Algoritme SPECK

Gambar 5.10 untuk melihat perbedaan waktu pengujian pemrosesan waktu *key schedule*, waktu enkripsi dan waktu dekripsi algoritme SPECK *block size* 128bit *key size* 256bit, maka dilakukan *test* menjalankan program sebanyak 1024 kali pada Raspberry Pi. Disebelah kanan grafik merupakan waktu yang dibutuhkan untuk setiap menjalankan program, dibawah grafik adalah banyaknya pengujian yang dilakukan mencapai 1024 pengujian. Sehingga menghasilkan bentuk grafik sedemikian rupa yang menandakan bahwa waktu untuk setiap proses algoritme SPECK terdapat perbedaan. Hasil dari pengujian ini juga mendapatkan rata – rata waktu yang dapat dilihat pada Tabel 5.14 berikut:

Tabel 5.7 Rata – rata waktu algoritma SPECK

Rata-rata Waktu <i>key schedule</i>	30012 μ s
rata-rata waktu enkripsi	30526 μ s
rata-rata waktu dekripsi	30535 μ s
total waktu rata-rata	91073 μ s

5.1.3 Uji Kruskal-Wallis

Pada uji Kruskal-Wallis data ini digunakan untuk mengetahui perbedaan antara data yang sudah didapatkan setelah pengujian dilaksanakan. Kruskal-Wallis pengujiannya dikhususkan dalam data non-parametrik. Jika nilai dari sig Kruskal-Wallis menghasilkan $<0,05$ maka harus dilakukan uji selanjutnya yaitu uji *Post Hoc tests*. Dalam pengujian *post hoc tests* dipilih algoritme SPECK dengan varian *plaintext* yang sama dengan panjang *key* yang berbeda. Cara melakukan uji *post hoc* dengan membandingkan antara data 1 dan 2, 1 dan 3, 2 dan 3. Kemudian akan dapat dilihat berapakah nilai sig dan nilai *Chi-Square* yang tertinggi Sehingga data mampu disimpulkan.

5.1.3.1 Uji Kruskal-Wallis H

Dalam pengujian disini akan diuji untuk seluruh data waktu *Key schedule* dari seluruh algoritme SPECK. Nilai *absdiff* adalah nilai yang dihasilkan dari nilai rank *key schedule* dikurangi dengan nilai mean dari *key schedule*. Nilai tersebut yang digunakan untuk proses pengujian Kruskal-Wallis, karena data yang didapat dari proses pengujian merupakan data yang variannya bermacam – macam (bebas). Sehingga daruh dicari dan mean dari data tersebut. Hasil dari pengujian sebagai berikut:

Tabel 5.8 Deskriptif Kruskal-Wallis waktu *Key schedule*

Descriptive Statistics								
	N	Mean	Std. Deviation	Minimum	Maximum	Percentiles		
						25th	50th (Median)	75th
absdiff	3072	27721,1839	405,09855	25331,24	29305,89	27452,5208	27747,9229	28027,2422
SPECK	3072	2,0000	,81663	1,00	3,00	1,0000	2,0000	3,0000
K								

Tabel 5.9 Kruskal-Wallis test waktu *Key schedule*

Ranks			
	SPECK	N	Mean Rank
absdiff	SPECK128_128	1024	1861,83
	SPECK128_192	1024	1680,59
	SPECK128_256	1024	1067,08
	Total	3072	



Tabel 5.10 Hasil Uji Kruskal-Wallis waktu *Key schedule*

Test Statistics ^{a,b}	
	absdiff
Chi-Square	451,622
df	2
Asymp. Sig.	,000
a. Kruskal Wallis Test	
b. Grouping Variable: SPECK	

Hasil dari pengujian diatas bisa disimpulkan bahwa nilai dari sig < 0,05 yang artinya terdapat perbedaan waktu yang signifikan ketika program dijalankan antara ketiga kelompok algoritme SPECK yang dibandingkan. Untuk mengetahui data kelompok manakah yang membuat data tersebut memiliki perbedaan waktu maka dilakukan uji *post hoc test*.

Tabel 5.11 Deskriptif Kruskal-Wallis waktu enkripsi

Descriptive Statistics								
	N	Mean	Std. Deviation	Minimum	Maximum	Percentiles		
						25th	50th (Median)	75th
absdiff	3072	28123,9430	327,31787	26726,35	28737,02	27872,4609	28123,9609	28390,4609
SPECK	3072	2,0000	,81663	1,00	3,00	1,0000	2,0000	3,0000
K								

Tabel 5.12 Kruskal-Wallis test waktu enkripsi

Ranks			
	SPECK	N	Mean Rank
absdiff	SPECK128_128	1024	1798,14
	SPECK128_192	1024	1643,33
	SPECK128_256	1024	1168,03
	Total	3072	

Tabel 5.13 Hasil uji Kruskal-Wallis waktu enkripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	280,695
df	2
Asymp. Sig.	,000



Hasil dari pengujian diatas bisa disimpulkan bahwa nilai dari sig < 0,05 yang artinya terdapat perbedaan waktu yang signifikan antara ketiga kelompok algoritme SPECK yang dibandingkan. Untuk mengetahui data kelompok manakah yang membuat data tersebut memiliki perbedaan waktu maka dilakukan uji *post hoc test*.

Tabel 5.14 Hasil uji Kruskal-Wallis waktu dekripsi

Descriptive Statistics								
	N	Mean	Std. Deviation	Minimum	Maximum	Percentiles		
						25th	50th (Median)	75th
absdiff	3072	28133,3014	344,58692	25720,07	28775,07	27882,3701	28139,0732	28408,4609
SPECK	3072	2,0000	,81663	1,00	3,00	1,0000	2,0000	3,0000
K								

Tabel 5.15 Hasil uji Kruskal-Wallis waktu dekripsi

Ranks			
	SPECK	N	Mean Rank
absdiff	SPECK128_128	1024	1858,31
	SPECK128_192	1024	1591,51
	SPECK128_256	1024	1159,68
	Total	3072	

Tabel 5.16 Hasil uji Kruskal-Wallis waktu dekripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	323,567
df	2
Asymp. Sig.	,000

Hasil dari pengujian diatas bisa disimpulkan bahwa nilai dari sig < 0,05 yang artinya terdapat perbedaan waktu yang signifikan antara ketiga kelompok Algoritme SPECK yang dibandingkan. Untuk mengetahui data kelompok manakah yang membuat data tersebut memiliki perbedaan waktu maka dilakukan uji *post hoc test*.

5.1.4 Uji Post Hoc Test

Dalam pengujian ini diambil dari Algoritme SPECK *block size* 128bit *key size* 128bit, Algoritme SPECK *block size* 128 *key size* 192, Algoritme SPECK *block size* 128 *key size* 256. Pengujian dilakukan pada plaintext yang sama dengan perbedaan panjang key. Dengan hasil sebagai berikut:



5.1.4.1 Waktu Key schedule

Tabel 5.17 Hasil uji Kruskal-Wallis waktu *key schedule*

Test Statistics ^{a,b}	
	absdiff
Chi-Square	451,622
df	2
Asymp. Sig.	,000

Hasil pada Tabel 5.24 merupakan hasil dari uji Kruskal-Wallis dengan membandingkan 3 algoritma SPECK dengan tipe *block size* 128bit dan varian *key* 128bit, 192bit dan 256bit. Dapat disimpulkan dari nilai sig terlihat $< 0,05$ yang artinya terdapat perbedaan waktu yang signifikan. Untuk melihat perbandingan manakah yang memiliki perbedaan waktu yang besar maka diujikan pada pengujian *post hoc tests* sebagai berikut:

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192.

Tabel 5.18 Hasil uji *post hoc tests* waktu *key schedule*

Test Statistics ^{a,b}	
	absdiff
Chi-Square	16,926
df	1
Asymp. Sig.	,000

2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256.

Tabel 5.19 Hasil uji *post hoc tests* waktu *key schedule*

Test Statistics ^{a,b}	
	absdiff
Chi-Square	431,921
df	1
Asymp. Sig.	,000

3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256.

Tabel 5.20 Hasil uji *post hoc tests* waktu *key schedule*

Test Statistics ^{a,b}	
	absdiff
Chi-Square	229,260
df	1
Asymp. Sig.	,000

Dari uji post hoc didapatkan nilai sebagai berikut :

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 bernilai 16,926.
2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256 bernilai 431,921.
3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256 bernilai 229,260.

Nilai *Chi-Square* dari algoritme SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 adalah 431,921. Dari Tabel diatas juga diketahui nilai sig $0,01 < 0,05$, maka H_0 ditolak. Maka terdapat perbedaan waktu yang signifikan. Hasil analisis dari pengujian diatas dapat disimpulkan bahwa algoritme SPECK *block size* 128bit *key size* 128bit dengan algoritme SPECK *block size* 128bit *key size* 256bit memiliki perbedaan waktu saat dijalankan. Hal ini disebabkan proses untuk melakukan enkripsi dan dekripsi untuk satu kali pengujian menggunakan ukuran bit yang besar untuk jumlah *plaintext* saja. Jika ditambah dengan memasukkan *key* yang memiliki varian bit yang berbeda-beda maka dipastikan proses akan mengalami penambahan waktu proses. karena semakin panjang ukuran *key* yang dimasukkan kedalam program maka waktu proses yang dijalankan juga semakin bertambah.

5.1.4.2 Waktu Enkripsi

Tabel 5.21 Hasil uji Kruskal-Wallis tests waktu enkripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	280,695
df	2
Asymp. Sig.	,000

Hasil pada Tabel 5.28 merupakan hasil dari uji Kruskal-Wallis dengan membandingkan 3 algoritma SPECK dengan tipe *block size* 128bit dan varian *key* 128bit, 192bit dan 256bit. Dapat disimpulkan dari nilai sig terlihat $< 0,05$ yang artinya terdapat perbedaan waktu yang signifikan. Untuk melihat perbandingan manakah yang memiliki perbedaan waktu yang besar maka diujikan pada pengujian post hoc tests sebagai berikut:

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192.

Tabel 5.22 Hasil uji *post hoc tests* waktu enkripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	16,709
df	1
Asymp. Sig.	,000

2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256.

Tabel 5.23 Hasil uji *post hoc tests* waktu enkripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	253,938
df	1
Asymp. Sig.	,000

3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256.

Tabel 5.24 Hasil uji *post hoc tests* waktu enkripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	150,391
df	1
Asymp. Sig.	,000

Dari uji *post hoc* didapatkan nilai sebagai berikut :

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 bernilai 16,709.
2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256 bernilai 253,938.
3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256 bernilai 150,391.

Nilai *Chi-Square* dari algoritme SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 adalah 253,938. Dari Tabel diatas juga diketahui nilai sig $0,01 < 0,05$, maka H_0 ditolak. Maka terdapat perbedaan waktu yang signifikan. Hasil analisis dari pengujian diatas dapat disimpulkan bahwa algoritme



SPECK *block size* 128bit *key size* 128bit dengan algoritme SPECK *block size* 128bit *key size* 256bit memiliki perbedaan waktu saat dijalankan. Hal ini disebabkan proses untuk melakukan enkripsi dan dekripsi untuk satu kali pengujian menggunakan ukuran bit yang besar untuk jumlah *plaintext* saja. Jika ditambah dengan masukkan *key* yang memiliki varian bit yang berbeda-beda maka dipastikan proses akan mengalami penambahan waktu proses. karena semakin panjang ukuran *key* yang dimasukkan kedalam program maka waktu proses yang dijalankan juga semakin bertambah.

5.1.4.3 Waktu Dekripsi

Tabel 5.25 Hasil uji Kruskal-Wallis tests waktu dekripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	323,567
df	2
Asymp. Sig.	,000

Hasil pada Tabel 5.32 merupakan hasil dari uji Kruskal-Wallis dengan membandingkan 3 algoritma SPECK dengan tipe *block size* 128bit dan varian *key* 128bit, 192bit dan 256bit. Nilai sig < 0,05 yang artinya H0 ditolak dan H1 diterima dapat ditarik kesimpulan terdapat perbedaan waktu yang signifikan. Untuk melihat perbandingan manakah yang memiliki perbedaan waktu yang besar maka diujikan pada pengujian *post hoc tests* sebagai berikut:

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192.

Tabel 5.26 Hasil uji *post hoc tests* waktu dekripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	49,858
df	1
Asymp. Sig.	,000
a. Kruskal-Wallis Test	
b. Grouping Variable:	SPECK



2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256.

Tabel 5.27 Hasil uji *post hoc tests* waktu dekripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	308,586
df	1
Asymp. Sig.	,000
a. Kruskal-Wallis Test	
b. Grouping Variable: SPECK	

3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256.

Tabel 5.28 Hasil uji *post hoc tests* waktu dekripsi

Test Statistics ^{a,b}	
	absdiff
Chi-Square	127,030
df	1
Asymp. Sig.	,000
a. Kruskal-Wallis Test	
b. Grouping Variable: SPECK	

Dari uji *post hoc* didapatkan nilai sebagai berikut :

1. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 bernilai 49,858.
2. SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 256 bernilai 308,586.
3. SPECK *block size* 128 *key size* 192 dengan SPECK *block size* 128 *key size* 256 bernilai 127,030.

Nilai *Chi-Square* dari algoritme SPECK *block size* 128bit *key size* 128bit dengan SPECK *block size* 128 *key size* 192 adalah 308,586. Dari Tabel 5.35 diketahui nilai sig $0,01 < 0,05$, maka H_0 ditolak. Maka terdapat perbedaan waktu yang signifikan. Hasil analisis dari pengujian diatas dapat disimpulkan bahwa algoritme SPECK *block size* 128bit *key size* 128bit dengan algoritme SPECK *block size* 128bit *key size* 256bit memiliki perbedaan waktu saat dijalankan. Hal ini disebabkan proses untuk melakukan enkripsi dan dekripsi untuk satu kali pengujian menggunakan ukuran bit yang besar untuk jumlah *plaintext* saja. Jika ditambah dengan masukkan *key* yang memiliki varian bit yang berbeda-beda maka dipastikan proses akan



mengalami penambahan waktu proses. karena semakin panjang ukuran *key* yang dimasukkan kedalam program maka waktu proses yang dijalankan juga semakin bertambah.

Kesimpulan dari analisis tiga algoritme yaitu algoritme SPECK *block size* 128bit *key size* 128bit, algoritme SPECK *block size* 128bit *key size* 192bit, Algoritme SPECK *block size* 128bit *key size* 256bit yang mempengaruhi nilai sig untuk uji Kruskal-Wallis adalah algoritme SPECK *block size* 128bit *key size* 256bit, karena *key size* yang dipakai pada Algoritme tersebut sangat besar mencapai 256bit, dimana membutuhkan waktu untuk memproses enkripsi dan dekripsi data yang cukup lama.

5.1.5 Hasil pengujian CPU dan RAM

Dalam penelitian ini pengujian RAM dan CPU menggunakan aplikasi monitoring TOP. Metode yang digunakan dalam pengujian dengan cara menjalankan setiap program algoritme SPECK satu-persatu dan saat menjalankan program TOP juga harus sudah dijalankan terlebih dahulu sehingga ketika program berjalan dapat dilihat proses CPU dan RAM yang digunakan. Dalam proses monitoring sudah didapatkan hasil dari kapasitas memori yang terpakai untuk seluruh Algoritme SPECK sebesar 0,6%. Sehingga didapatkan nilai RAM 0,6% dari 947732KiB Memori yang merupakan jumlah memori total pada Raspberry Pi.

$$0,6\% \times 947732 = 568639,2 \text{ KiB memori}$$

Untuk CPU yang digunakan adalah 2,6% untuk seluruh Algoritme SPECK namun itu hanya pada saat masing-masing program dijalankan. Setelah program berjalan maka proses CPU bernilai 0,0%. Proses CPU akan dilemparkan kedalam Memori yang artinya CPU hanya dibutuhkan saat program akan dijalankan ketika program berjalan maka CPU tidak sepenuhnya digunakan.

```

pi@raspberrypi ~
└─$ top
top - 14:50:19 up 3 min, 4 users, load average: 0.14, 0.12, 0.05
tasks: 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.0 us, 0.4 sy, 0.0 ni, 98.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 947732 total, 238052 used, 709680 free, 15200 buffers
KiB Swap: 102396 total, 0 used, 102396 free. 134204 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 1017 root        20   0   9252   5344   3476  S   2.6   0.6   0:00.08 python
  
```

Gambar 5.4 RAM dan CPU saat dijalankan

```

KiB Mem: 947732 total, 237492 used, 710240 free, 15492 buffers
KiB Swap: 102396 total, 0 used, 102396 free. 134264 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 1083 root        20   0   9252   5452   3588  S   0.0   0.6   0:00.08 python
  
```

Gambar 5.5 RAM dan CPU saat program sudah berjalan



BAB 6 PENUTUP

6.1 Kesimpulan

Sesuai dengan rumusan masalah yang diajukan diawal penelitian serta berdasarkan hasil analisis dari pengujian maka dapat ditarik beberapa kesimpulan sebagai berikut :

1. Implementasi Algoritme SPECK dilakukan dengan cara menghubungkan Raspberry Pi pada laptop dengan menggunakan kabel LAN, selanjutnya melakukan seting konfigurasi ip pada laptop dan Raspberry Pi. Jika sudah diakses melalui *software Putty* maka Raspberry Pi sudah terkoneksi dengan laptop. Kemudian memasukkan program algoritme SPECK kedalam Raspberry Pi. Menjalankan perintah pada Raspberry Pi untuk mengeksekusi program Algoritme SPECK. Pemasukkan *plaintext* dan *key* agar program dapat menampilkan hasil *output*.
2. Performansi dari Algoritme SPECK berdasarkan *block size* dan *key size* dikatakan cukup baik, ditunjukkan dari total rata-rata waktu pemrosesan algoritme SPECK block size 32bit key size 64bit sebesar 18851 μ s, Algoritme SPECK block size 48bit key size 72bit sebesar 25373 μ s, algoritme SPECK block size 48bit key size 96bit sebesar 26275 μ s, algoritme SPECK block size 64bit key size 96bit sebesar 36488 μ s, algoritme SPECK block size 64bit key size 128bit sebesar 39782 μ s, algoritme SPECK block size 96bit key size 96bit sebesar 58478 μ s, algoritme SPECK block size 96bit key size 144bit sebesar 60532 μ s, algoritme SPECK block size 128bit key size 128bit sebesar 85920 μ s, algoritme SPECK block size 128bit key size 192bit sebesar 88761 μ s, algoritme SPECK block size 128bit key size 256bit sebesar 91073 μ s. Algoritme SPECK memiliki konsumsi RAM sebesar 0,6% dari memori Raspberry Pi dan CPU yang terpakai 2.6% pada saat program dijalankan, setelah program berjalan maka CPU bernilai 0,0%, karena proses akan dipindahkan pada RAM.
3. Hasil yang dikeluarkan dari Raspberry Pi valid *plaintext* dan *Ciphertext* setelah proses enkripsi menghasilkan nilai yang sama dengan *test vektor*, pada proses dekripsi juga menghasilkan nilai yang sama dengan masukkan awal dari *plaintext*.

6.2 Saran

Saran dari penelitian ini sebagai berikut:

- 1 Hasil spesifikasi kebutuhan dan perancangan sistem dapat dilanjutkan sebagai dasar tahap pengembangan selanjutnya yaitu tahap implementasi sistem.
- 2 Penelitian selanjutnya dapat dikembangkan dengan menambahkan sebuah sensor untuk menjadikan *input*.

DAFTAR PUSTAKA

- Adrian-Vasile Duka, dkk, 2017. Implementation of SIMON and SPECK Lightweight Block Ciphers on Programmable Logic Controllers.
- Alex Biryukov, dkk, 2014. Differential Analysis of Block Ciphers SIMON and SPECK.
- Ariyus, D., 2008. *Pengantar Ilmu Kriptografi : Teori analisis & implementasi.* Yogyakarta: CV ANDI OFFSET.
- Boris Ryabko, 2018. The distinguishing attack on SPECK, SIMON, Simeck, HIGHT and LEA.
- Drs. Ario Suryo Kusumo, 2004. In: *Visual Basic.NET versi 2002 dan 2003.* Jakarta: Pt Elex Media Komputindo.
- Eko Hari Rachmawanto, dkk, 2015. Keamanan File Menggunakan Teknik Kriptografi Shift Cipher.
- Gudono, 2011. *Analisis Data Multivariat.* Yogyakarta: BPFE-YOGYAKARTA.
- Herlina Budiono, J., 2014. *Statistik Terapan: Aplikasi untuk Riset Skripsi, Tesis dan Disertasi.* s.l.:Elex Media Komputindo.
- Junaidi, 2015. Statistik Uji Kruskal-Wallis.
- Komputer, W., 2013. *The Best Encryption Tools.* s.l.:Elex Media Komputindo.
- Kusbandono, A., 2004. Implementasi Algoritme AES-128 pada Mikrokontroler 8051.
- Lubis, M. S., 2018. *Metodologi penelitian.* Yogyakarta: Deepublish.
- Matt Richardson, Shawn Wallace, 2012. Getting Started with Raspberry Pi. In: United States of America: Christopher Hearse.
- Permana, T. D., 2014. SISTEM MONITORING MENGGUNAKAN MINI PC RASPBERRY PI.
- Ray Beaulieu dkk, 2013. THE SIMON AND SPECK FAMILIES OF LIGHTWEIGHT BLOCK CIPHERS.
- Ray Beaulieu dkk, 2015. The SIMON and SPECK lightweight block ciphers.
- Rizky Oktora Prihadini Putri dkk, 2014. Keefektifan Pembelajaran Matematika dengan Pendekatan CTL dan Problem Posing Ditinjau dari Ketercapaian SK/KD dan Kemampuan Koneksi Matematik.
- Salemba, P., 2014. *Statistik untuk Kedokteran dan Kesehatan.* Jakarta: Salemba Medika.
- Thomas Eisenbarth, Erdinç Öztürk, 2014. *Lightweight Cryptography for Security and Privacy.* Turkey: Springer.
- Tim EMS, 2015. *Kamus Komputer Lengkap,* Jakarta: PT Elex Media Komputindo.