BAB III

METODE DAN PERANCANGAN IMPLEMENTASI PROGRAM

3.1 Metode Penelitian

Penelitian dibagi menjadi beberapa tahap. Uraian dari tahap-tahap tersebut dapat dijelaskan sebagai berikut.

1. Telaah Literatur

Selama penelitian berlangsung, diperlukan pemahaman mengenai topik penelitian khususnya pada tahap implementasi. Pendalaman materi harus dilakukan seiring dengan berjalannya penelitian sehingga hasil dari penelitian merupakan hasil yang terbaik. Adapun materi yang didalami berupa algoritma enkripsi serta arsitektur multiprosesor yang akan digunakan sesuai dengan inti dari penelitian. Telaah literatur dilakukan dengan pencarian dan pembelajaran referensi yang dapat berupa buku, jurnal penelitian, dan artikel.

2. Analisis Kebutuhan

Pada tahap ini, kebutuhan dari implementasi dianalisa. Analisa sistem berupa analisa kebutuhan dari fitur-fitur yang terdapat pada algoritma serta metode implementasi pada arsitektur multiprosesor untuk mendapatkan hasil yang optimal. Fitur-fitur dari implementasi yang akan dianalisa berupa fitur enkripsi algoritma-algoritma enkripsi RC4, RC4-2S, dan RC4itz dalam arsitektur multiprosesor.

3. Perancangan Program

Pada tahap ini, dilakukan perancangan program dari algoritma yang akan diterapkan dalam pemrograman paralel yang digunakan. Pada tahap ini juga dilakukan perancangan desain arsitektur multiprosesor yang akan digunakan.

Perancangan program dan arsitektur multiprosesor akan didasari dengan penelitian-penelitian sebelumnya.

4. Pemrograman dan Implementasi

Pada tahap ini, proses implementasi algoritma enkripsi pada arsitektur multiprosesor dilaksanakan. Pengimplementasian enkripsi pada arsitektur multiprosesor didasari oleh penelitian pada *"Efficient Implementation for MD5-RC4 Encryption Using GPU with CUDA"* (Li dkk, 2009) dan *"Improving Throughput of RC4 Algorithm using Multithreading Techniques in Multicore Processors"* (Weeransinghe, 2012) mengenai pengembangan RC4 pada *multi-threading* prosesor *multicore* CPU dan GPU. Pengimplementasian akan melangkah lebih jauh lagi dengan mengimplementasikan algoritma kriptografi RC4 serta varian baru dari RC4 yaitu RC4-2S dan RC4itz pada arsitektur multiprosesor. Pemrograman dan implementasi dilaksanakan pada Microsoft Visual Studio Ultimate 2013 dengan bahasa pemrograman C++ dengan prosesor Intel i7 untuk CPU dan GeForce GTX-970 untuk GPU dengan menggunakan API CUDA untuk pemrograman GPU.

5.*Testing* dan *Debugging*

Pengujian dan *debugging* program akan dilakukan pada tahap ini. Pengujian dilakukan dengan melakukan proses enkripsi pada file uji coba berbentuk PDF. Pada proses ini, struktur program akan dievaluasi dan akan dilakukan proses *debugging* untuk memperbaiki *bug* pada program.

6 Analisis dan Evaluasi Hasil Implementasi

Analisis dan evaluasi dari hasil implementasi akan dilaksanakan setelah tahap pengimplementasian. Perbandingan hasil algoritma enkripsi RC4, RC4-2S, serta RC4itz dalam arsitektur multiprosesor akan dianalisa pada tahap ini untuk membandingkan performa dari algoritma-algoritma kriptografi tersebut. Langkah-langkah dalam melakukan *statistical analysis* telah dijabarkan dalam bab telaah literatur. Analisa dilakukan dengan melakukan perbandingan perhitungan *throughput* algoritma enkripsi RC4, RC4-2S, serta RC4itz sebagai salah satu perbandingan performa dari algoritma enkripsi tersebut. Hasil dari analisa akan dievaluasi serta dicatat pada laporan penelitian.

7. Konsultasi dan penulisan

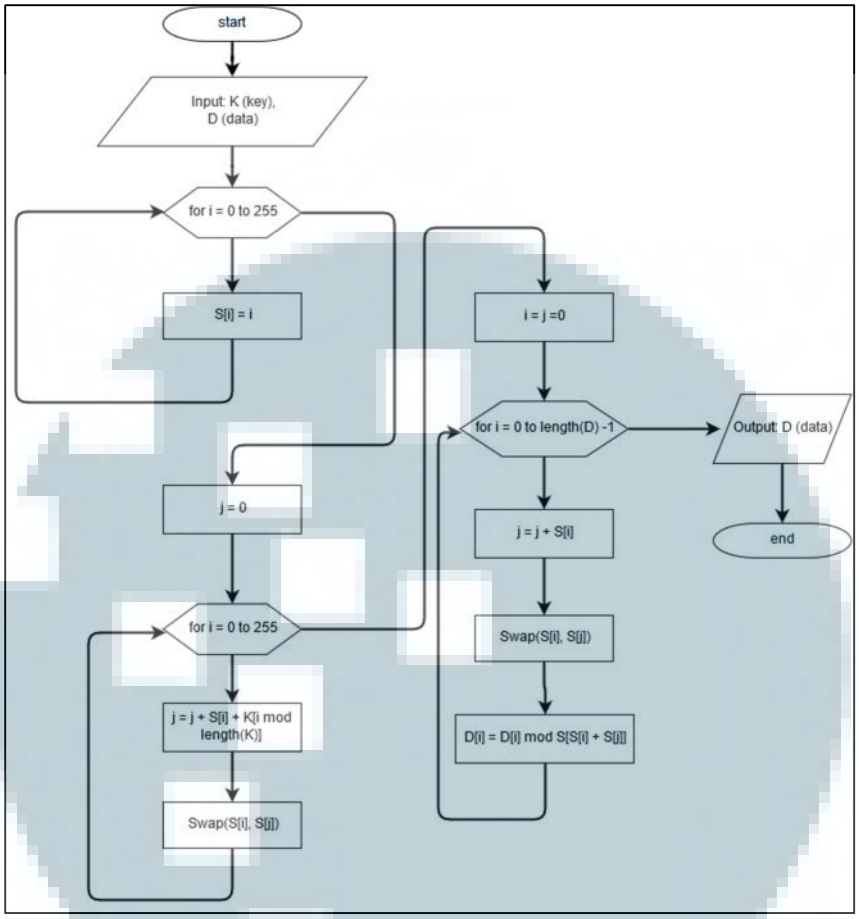
Proses konsultasi penelitian dan penulisan dilakukan selama penelitan berlangsung. Proses konsultasi dilakukan dengan bimbingan dari dosen pembimbing untuk menyempurnakan hasil penelitian. Penulisan laporan dilakukan sebagai dokumentasi dari hasil penelitian yang dilakukan.

3.2 Analisa Program

Dalam perancangan program dalam arsitektur multiprosesor, dibutuhkan analisa proses enkripsi yang akan digunakan dalam pemrograman. Alur dari algoritma enkripsi yang akan digunakan dapat digambarkan dalam sebuah diagram *flowchart* yang akan dijelaskan pada Gambar 3.1 hingga 3.3.

3.2.1 Perancangan Analisa Algoritma RC4

RC4 merupakan salah satu algoritma enkripsi yang akan diimplementasikan ke dalam arsitektur multiprosesor dalam penelitian. Alur algoritma RC4 haruslah dibuat dan dipelajari terlebih dahulu untuk mengimplementasikan algoritma tersebut ke dalam arsitektur multiprosesor. Secara singkat, algoritma RC4 mempunyai data input yang berupa kunci rahasia (K), dan data yang akan dienkripsi (D). Proses pertama yang akan dilakukan dalam algoritma RC4 merupakan proses pembuatan *key-state*. Proses pembuatan *key-state* dibagi menjadi dua proses yaitu proses inisialisasi dan proses *swapping*. *Key-state* yang dihasilkan berukuran sebesar 256 *bytes* yang merupakan jumlah terbesar dari karakter ASCII (*American Standard Code for Information Interchange*). Setelah *key-state* dihasilkan, maka alur algoritma akan berlanjut pada proses enkripsi. RC4 merupakan sebuah algoritma *stream-cipher* dimana setiap bit dari sebuah data yang di-*input* pada algoritma akan dienkripsi bit-per-bit. Oleh karena hal tersebut, proses enkripsi akan diulang sebanyak ukuran data *input*. Alur algoritma yang telah dijabarkan dapat digambarkan dalam sebuah diagram *flowchart*. Gambar 3.1 merupakan sebuah diagram *flowchart* yang menggambarkan alur algoritma RC4.



Gambar 3.1 *Flowchart* Alur Algoritma RC4

Dari Gambar 3.1, dapat terlihat alur kerja dari algoritma RC4. RC4 memiliki tiga *loop* yang memiliki tingkat *loop* berbeda. Dari penjabaran dan alur *flowchart* RC4, sebagai bentuk analisis algoritma lebih lanjut, dapat dilakukan perhitungan kompleksitas algoritma menggunakan perhitungan *Big O*. Kompleksitas dari RC4 dapat dilihat pada Rumus 3.1.

$$f(N) = O(6) + O(6N) \qquad \text{... Rumus 3.1}$$

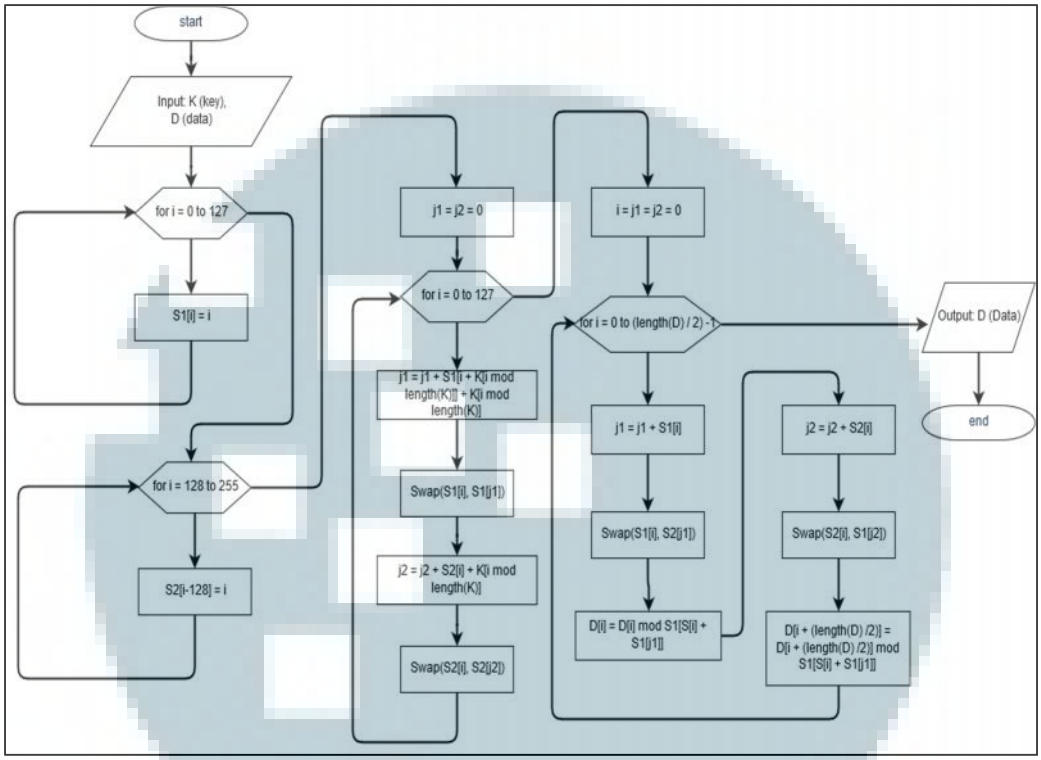
Dari hasil rumus *Big O* tersebut, maka dapat disimpulkan bahwa algoritma RC4 merupakan algoritma dengan pertumbuhan yang linear. Pertumbuhan algoritma RC4 dipengaruhi oleh variabel 'N'. Semakin besar nilai variabel tersebut,

maka semakin tinggi tingkat kompleksitas dari algoritma dan semakin besar waktu yang diperlukan untuk menjalankan algoritma.

3.2.2 Perancangan Analisa Algoritma RC4-2S

RC4-2S merupakan pengembangan dari algoritma RC4. Alur dari RC4-2S memiliki sedikit pengembangan dikarenakan RC4-2S mengimplementasikan dua buah *key-state*. Alur dari RC4-2S dapat dijabarkan secara singkat sebagai berikut. Algoritma RC4-2S menerima *input* yang berupa dokumen yang akan dienkripsi atau 'D' dan kunci rahasia 'K'. Proses awal yang dilakukan algoritma RC4-2S merupakan proses yang sama dengan RC4 yaitu proses pembentukan *key-state*. Pada proses ini dilakukan inisialisasi dari dua buah *key-state* yang akan digunakan. *Key-state* yang akan digunakan berukuran 128 *bytes*. *Key-state* pertama akan menampung nilai dari 0 hingga 127 dan *key-state* kedua akan menampung nilai dari 128 hingga 255. Jumlah ukuran dari dua *key-state* tersebut merupakan jumlah total dari karakter ASCII yaitu 256 *bytes*. Setelah kedua *key-state* terinisialisasi, maka proses selanjutnya yang akan dilakukan adalah *swapping* atau pertukaran nilai antara *key-state*. Setelah *key-state* terbentuk, maka alur akan masuk ke dalam proses enkripsi. Karena algoritma RC4-2S mengimplementasikan dua buah *key-state*, maka proses enkripsi dari algoritma RC4-2S dilakukan sebanyak setengah dari ukuran data yang dienkripsi. *Key-state* pertama akan digunakan dalam mengenkripsi setengah bagian awal dari data, dan *key-state* kedua akan digunakan dalam mengenkripsi setengah bagian akhir dari data. Dari penjabaran alur kerja algoritma RC4-2S, maka dapat dibuat sebuah diagram *flowchart* untuk

menggambarkan alur kerja RC4-2S. Diagram *flowchart* tersebut dapat dilihat pada Gambar 3.2.



Gambar 3.2 *Flowchart* Alur Algoritma RC4-2S

Dari Gambar 3.2, dapat terlihat alur kerja algoritma RC4-2S. RC4-2S memiliki empat *loop* dengan tingkatan yang berbeda. Dengan bantuan penjabaran dan diagram *flowchart* dari alur kerja RC4-2S, maka sebagai bentuk analisa algoritma lebih lanjut, dapat dihitung kompleksitas dari algoritma RC4-2S dengan menggunakan perhitungan *Big O*. Kompleksitas dari algoritma RC4-2S dapat dilihat pada Rumus 3.2.

$$f(N) = O(8) + O(12N) \qquad \text{... Rumus 3.2}$$

Berdasarkan hasil perhitungan *Big O* tersebut, maka dapat disimpulkan bahwa algoritma RC4-2S memiliki pertumbuhan yang linear. Sama dengan

algoritma RC4, pertumbuhan linear kompleksitas algoritma tersebut dipengaruhi oleh variabel ‘N’. Semakin besar nilai variabel tersebut, maka semakin tinggi tingkat kompleksitas dari algoritma dan semakin besar waktu yang diperlukan untuk menjalankan algoritma.

3.2.3 Perancangan Analisa Algoritma RC4itz

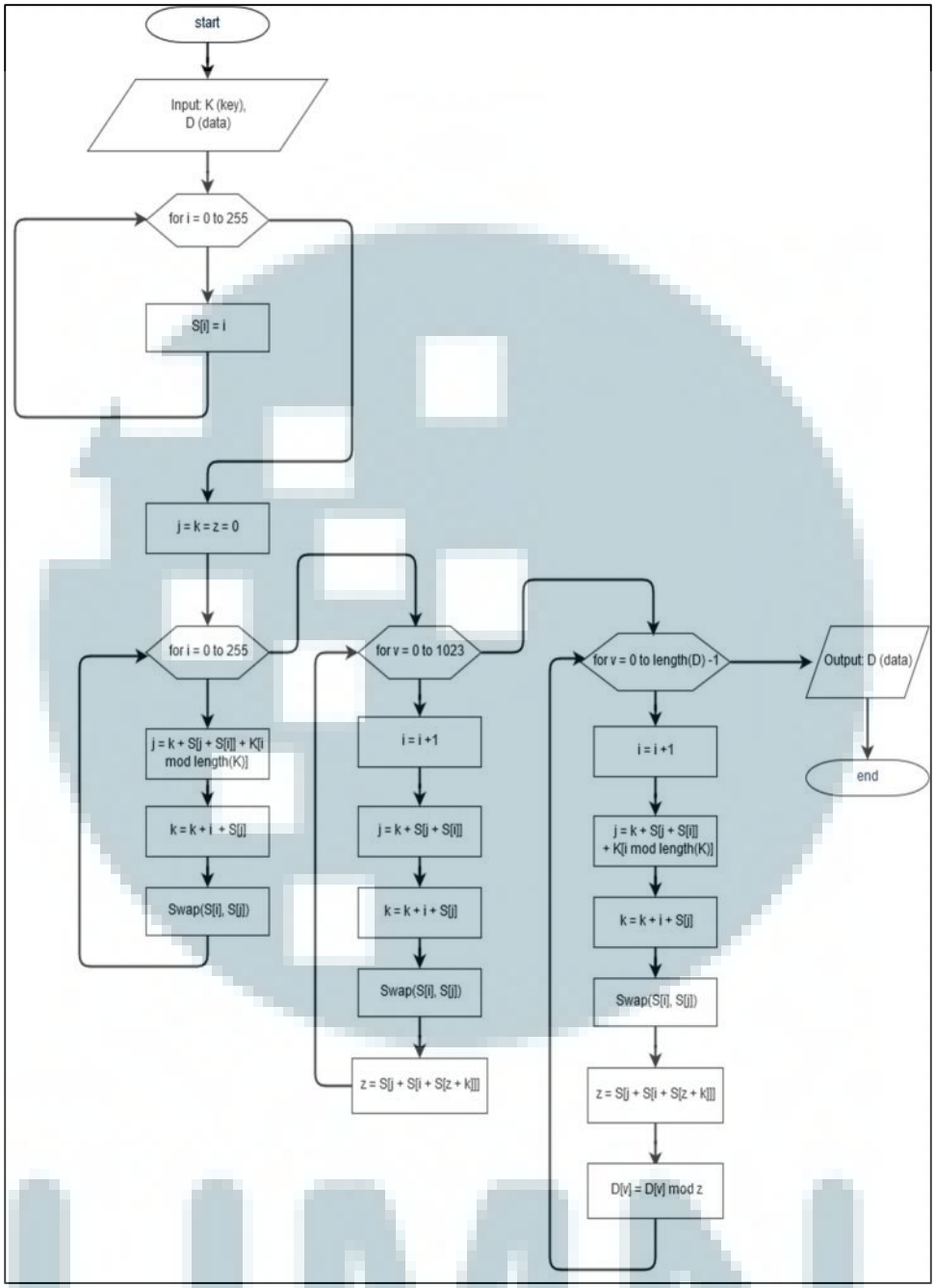
RC4itz merupakan salah satu pengembangan dari algoritma RC4. RC4 merupakan sebuah hibrida dari struktur algoritma RC4 dan struktur algoritma Spritz, sebuah pengembangan algoritma RC4. Menggunakan struktur algoritma yang mirip dengan RC4, RC4itz menambahkan suatu fungsi baru yaitu fungsi *update* (atau yang bernama ‘SkipOutput’ pada RC4itz) yang berguna untuk menambahkan keacakan dari *key-state* yang dihasilkan. Fungsi *update* mengambil hasil *key-state* yang dihasilkan dari fungsi *swapping* sebelumnya, dan melakukan *swapping* nilai dalam *key-state* untuk menambah keacakan dari nilai *key-state*. *Swapping* pada fungsi *update* dilakukan sebanyak 1024 kali atau sebesar satu *kilo-byte*. Selain implementasi fungsi *update*, RC4itz juga menambahkan *internal register* untuk pengacakan nilai *key-state* menjadi empat buah *internal register*.

Secara garis besar, alur kerja dari algoritma RC4itz dapat dijabarkan sebagai berikut. RC4itz menerima *input* data yang berupa dokumen yang akan dienkripsi atau ‘D’, dan kunci rahasia ‘K’. Alur algoritma RC4itz dimulai dengan inisialisasi *key-state* yang sebesar 256 *bytes* sesuai dengan jumlah karakter ASCII. Setelah proses inisialisasi, maka akan dilakukan proses *swapping* nilai *key-state*. Proses *swapping* nilai *key-state* pada RC4itz berlangsung hampir sama dengan proses *swapping* RC4. Namun, RC4itz, menggunakan satu buah *internal register* baru

yaitu k sebagai referensi *swapping*. Setelah proses *swapping* selesai, maka alur akan dilanjutkan dengan proses 'SkipOutput'. Pada proses ini, nilai *key-state* kembali diacak. Namun, hasil dari fungsi 'SkipOutput' sendiri bukan hanya pengacakan dari nilai *key-state.* Pada fungsi 'SkipOutput', nilai dari *internal register* 'z' juga akan didapatkan. *Internal register* 'z' merupakan sebuah nilai *register* baru yang akan digunakan dalam proses enkripsi setelah proses 'SkipOutput'. Setelah proses 'SkipOutput' selesai, maka alur akan berlanjut ke proses enkripsi. Pada proses ini, empat buah *internal register* yang telah diisi nilainya pada proses-proses sebelumnya akan digunakan dalam melakukan proses enkripsi 'D'.

Alur algoritma RC4itz memiliki tingkat kompleksitas yang lebih tinggi dari alur algoritma RC4 namun memiliki tingkat kompleksitas yang lebih rendah dari alur algoritma Spritz. Dengan penambahan *internal register* dan sebuah proses baru, RC4itz dikembangkan dengan tujuan menambah keacakan dari hasil enkripsi RC4.

Berdasarkan uraian dari algoritma RC4itz di atas, maka dapat dibuatlah sebuah diagram *flowchart* untuk menggambarkan alur kerja algoritma RC4itz. Diagram *flowchart* dari alur kerja algoritma RC4itz dapat dilihat pada Gambar 3.3.



Gambar 3.3 *Flowchart* Alur Algoritma RC4itz

Dari Gambar 3.3, dapat dilihat bahwa algoritma RC4itz memiliki empat buah *loop* dalam tingkatan yang berbeda. Dengan penjabaran dan diagram *flowchart* pada Gambar 3.3, maka dapat dilakukan perhitungan kompleksitas algoritma

RC4itz, sebagai analisa algoritma lebih lanjut, dengan perhitungan *Big O*. Kompleksitas dari algoritma RC4itz dapat dilihat pada Rumus 3.3.

$$f(N) = O(6) + O(15N) \qquad \text{… Rumus 3.3}$$

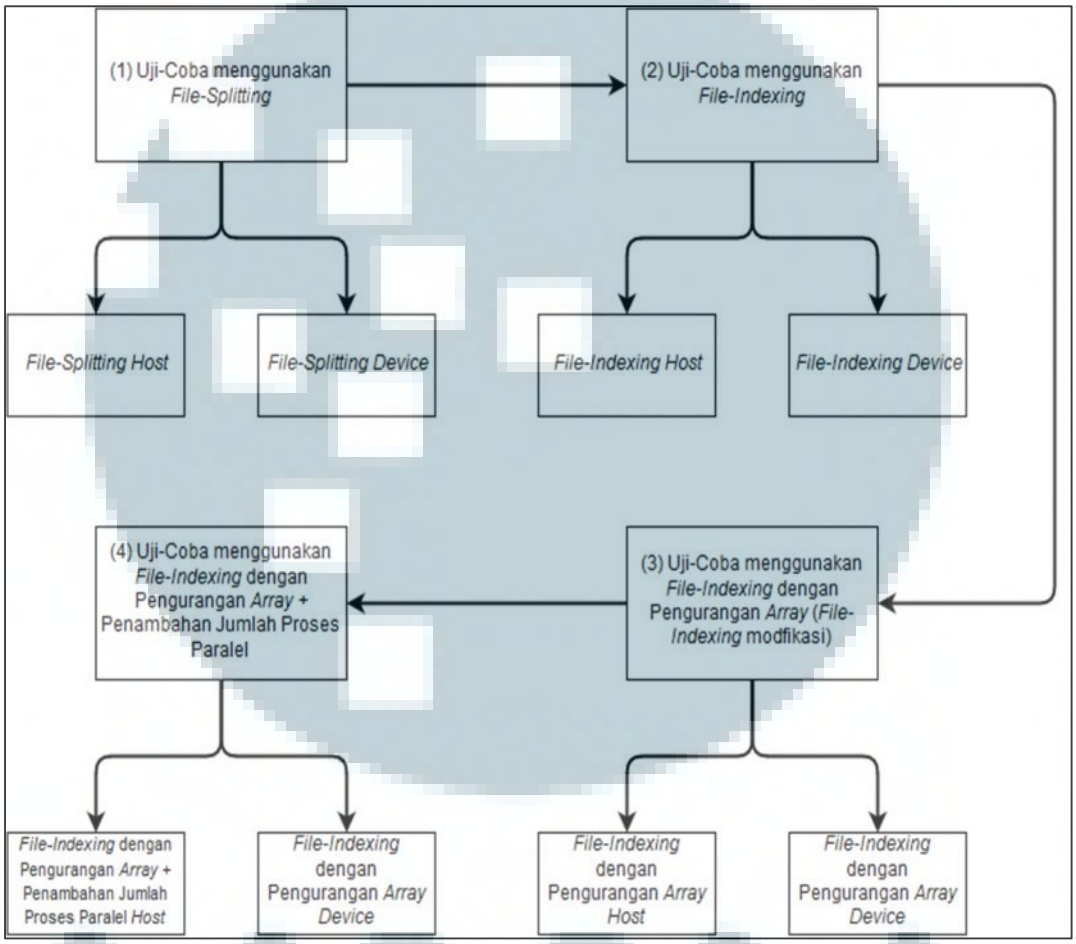
Berdasarkan hasil dari perhitungan kompleksitas algoritma RC4itz, maka dapat disimpulkan bahwa algoritma RC4itz memiliki pertumbuhan kompleksitas yang linear. Pertumbuhan linear kompleksitas RC4itz dipengaruhi oleh variabel 'N'. Pertumbuhan bergerak secara linear. Apabila variabel 'N' bertambah, maka tingkat kompleksitas dari algoritma RC4itz juga akan bertambah.

Dengan membandingkan hasil perhitungan kompleksitas algoritma, didapatkan sebuah kesimpulan bahwa algoritma RC4itz dengan jumlah variabel yang mempengaruhi pertumbuhan kompleksitas terbanyak, merupakan algoritma yang memiliki kompleksitas tertinggi dibandingkan dengan algoritma RC4 dan RC4-2S. Dengan tingkat kompleksitas yang tinggi, RC4itz diharapkan memiliki tingkat keamanan yang lebih tinggi. Namun sebagai gantinya, tingkat *throughput* dari RC4itz lebih rendah dibandingkan dengan kedua algoritma sebelumnya.

3.3 Perancangan Program pada Arsitektur Multiprosesor

Setelah dilakukan analisa terhadap algoritma-algoritma enkripsi yang akan digunakan, maka tahap berikut dalam penelitian adalah merancang sistem dasar yang akan dibuat. Sistem yang akan dirancang meliputi alur kerja program secara keseluruhan dan perancangan implementasi algoritma enkripsi pada proses paralel. Perancangan program dilakukan dengan empat uji coba yang berbeda. Uji coba menggunakan *file-splitting, file-indexing,* dan *file-indexing* dengan pengurangan jumlah *array* menggunakan 64 jumlah proses paralel dengan pembagian 8 *block*

dan 8 *thread* sementara uji coba penambahan jumlah proses paralel menggunakan jumlah proses paralel yang beragam untuk meneliti pengaruh peningkatan jumlah proses paralel pada arsitektur multi-prosesor. Alur dari pelaksanaan uji coba dapat dilihat pada Gambar 3.4.



Gambar 3.4 Alur Pelaksanaan Uji Coba

Dari Gambar 3.4, dapat dilihat bahwa uji coba pertama yang dilakukan adalah 'Uji Coba dengan *File-Splitting'* dengan pemrograman untuk *host* dan *device* untuk proses *file-splitting.* Uji coba kedua yang dilakukan adalah 'Uji Coba dengan *File-Indexing'*. Pemrograman juga dilakukan untuk *host* dan *device* untuk proses *file-splitting*. Uji coba ketiga yang dilakukan adalah 'Uji Coba *File-Indexing* dengan

Pengurangan Jumlah *Array'* atau dapat disebut juga sebagai 'Uji Coba *File-Indexing* Termodifikasi'. Pemrograman dilakukan pada bagian *host* dan *device* untuk proses *file-indexing* dengan pengurangan jumlah *array*. Uji coba keempat yang dilakukan adalah 'Uji Coba *File-Indexing* dengan Pengurangan Jumlah *Array'* ditambah dengan 'Penambahan Jumlah Proses Paralel'. Uji coba keempat juga dapat disebut sebagai 'Uji Coba Penambahan Jumlah Proses Paralel'. Pemrograman pada uji coba penambahan jumlah proses paralel menggunakan pemrograman pada uji coba *file-indexing* dengan pengurangan jumlah *array* sebagai dasar. Modifikasi yang dilakukan dalam uji coba penambahan jumlah proses paralel hanya dilakukan pada bagian *host*. Detil-detil dari perancangan uji coba akan dijelaskan pada subbab-subbab berikutnya.

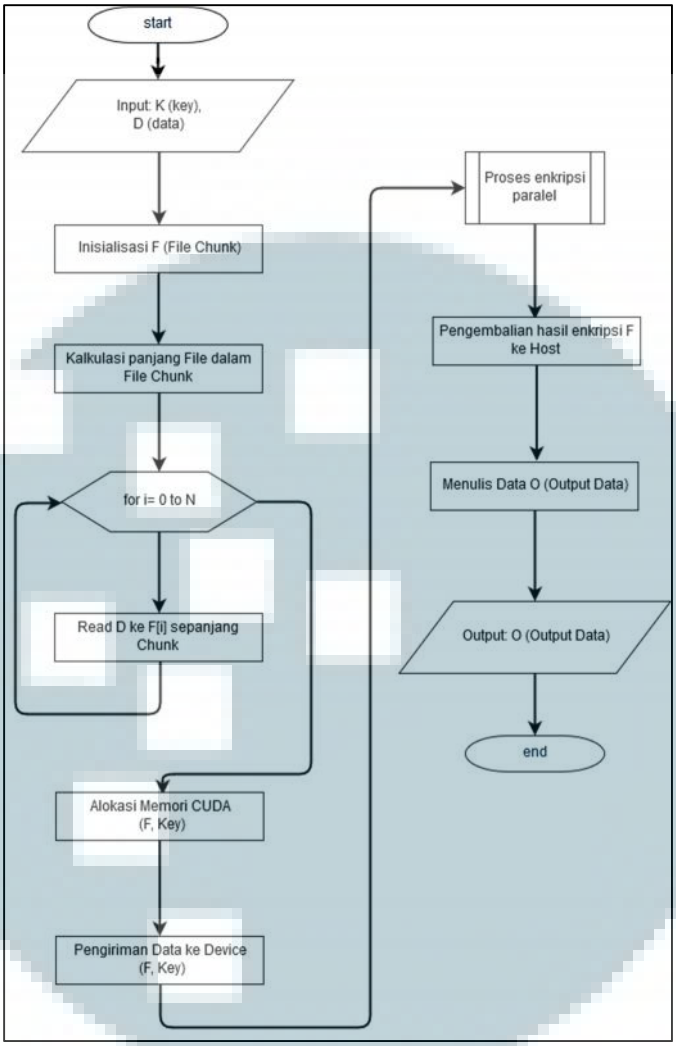
3.3.1 Perancangan Uji Coba Menggunakan File-Splitting

Pada uji coba menggunakan *file-splitting*, program terbagi menjadi dua yaitu program pada *host* dan program pada *device.* Uji coba dilakukan dengan membagi *file input* yang akan dienkripsi. Tujuan dari uji coba adalah untuk mengetahui apakah metode *file-splitting* secara pembagian *file-input* ke dalam beberapa variabel penampung dapat dilakukan untuk enkripsi secara paralel. Penjabaran dari program *host* dan *device* akan dijelaskan pada subbab-subbab berikut ini.

A. Perancangan Program pada Host

Garis besar alur program yang akan dirancang adalah sebagai berikut, program menerima *input* berupa data *input* 'D' atau dokumen *input* yang akan dienkripsi beserta sebuah kunci rahasia yang akan digunakan pada proses enkripsi.

Setelah mendapatkan *input* 'D', program akan melakukan inisialisasi variabel 'F' yang akan digunakan sebagai penampung *index* pada proses *file indexing*. Pada proses *file-splitting*, pertama dilakukan pencarian ukuran pecahan 'D' setelah dokumen dibagi menjadi 'N'-1 pecahan dimana 'N' merupakan jumlah paralel proses yang akan digunakan, serta sisa hasil bagi ukuran 'D' pada 'N'-1. Kemudian 'D' yang telah dipecah dimasukkan ke dalam sebuah variabel penampung 'F' sehingga 'F' menampung pecahan-pecahan data *input*. Setelah semua data lengkap, maka dilakukan alokasi dan pengiriman data pada GPU untuk 'F', dan kunci rahasia. Setelah alokasi selesai, maka proses enkripsi akan dilakukan pada GPU secara paralel sebanyak jumlah 'N'. Setelah proses enkripsi selesai dilakukan, maka hasil dari enkripsi tersebut dikembalikan ke CPU. Setelah data dikembalikan, maka data yang telah dienkripsi akan ditulis menjadi 'O' atau *output* data yang merupakan hasil dari enkripsi. Seluruh alur proses program dapat digambarkan dengan sebuah diagram *flowchart*. Diagram *flowchart* yang mengambarkan proses program pada *host* tersebut dapat dilihat pada Gambar 3.5 sebagai berikut.



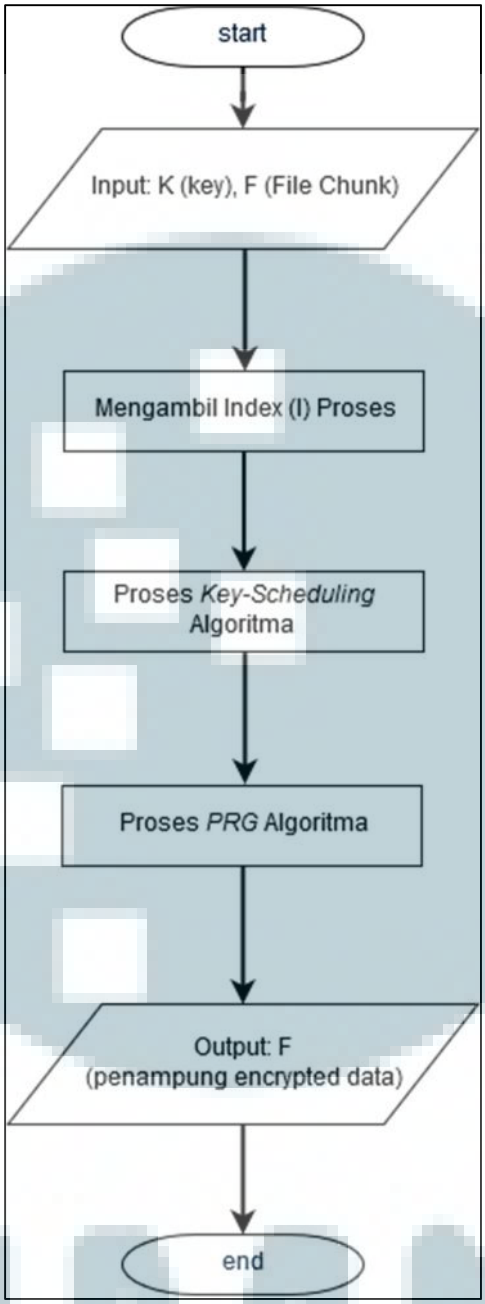
Gambar 3.5 *Flowchart* Proses Program pada *Host* dalam Uji Coba menggunakan *File-Splitting*

Seperti yang dapat dilihat pada Gambar 3.5, alur proses program pada *host* meliputi insialisasi variabel-variabel yang dibutuhkan, membaca data *input* serta memanggil *kernel* GPU. Program diawali oleh inisialisasi variabel 'F' yang merupakan penampung dari pecahan data *input* 'D' yang akan dienkripsi. Setelah inisialisasi, dilakukanlah perhitungan panjang dari setiap pecahan data 'D' sebanyak 'N' buah. Proses berlanjut untuk pembacaan data ke dalam penampung 'F'. Data-data tersebut dibaca sepanjang panjang pecahan data yang telah dihitung

sebelumnya. Setelah proses baca data *input* selesai, maka dilakukanlah alokasi memori dan pengiriman data-data ke pada *device*. Proses enkripsi secara paralel dapat dilakukan setelah pengiriman data ke *device*. Hasil dari proses paralel adalah sebuah variabel 'F' yang menampung data-data yang telah terenkripsi. Variabel 'F' tersebut akan dikembalikan ke *host* untuk dilakukan proses penulisan *output* data 'O'. *Output* data 'O' merupakan hasil akhir dari program.

B. Perancangan Implementasi Algoritma pada Device

Pada proses enkripsi *stream cipher*, sebuah data terenkripsi secara bit-per-bit. Proses enkripsi tersebut mudah dilakukan dalam sebuah program sekuensial. Akan tetapi, dalam sebuah paralel proses GPU mengharuskan sejumlah proses dapat dikerjakan dalam waktu yang bersamaan. Oleh karena hal tersebut, untuk mengimplementasikan algoritma *stream cipher* RC4, RC4-2S dan RC4itz ke dalam sebuah paralel proses, diterapkanlah proses *file-splitting*. Inti dari *file-splitting* adalah memecah data yang akan dienkripsi menjadi sejumlah data-data kecil yang akan dienkripsi secara paralel oleh GPU. Dalam uji coba menggunakan *file-splitting*, proses *file-splitting* yang dilakukan pada *host* atau CPU berupa pemecahan pembacaan *file* ke dalam sebuah penampung. Setelah didapatkan garis besar alur implentasi, maka proses implementasi dapat digambarkan dalam sebuah *flowchart*. *Flowchart* yang menggambarkan alur implementasi algoritma dalam paralel proses dapat dilihat pada Gambar 3.6.



Gambar 3.6 *Flowchart* Algoritma pada *Device* dalam Uji Coba menggunakan *File-Splitting*

Seperti yang dapat dilihat pada Gambar 3.6, garis besar dari implementasi algoritma enkripsi RC4, RC4-2S, dan RC4itz dalam sebuah proses paralel adalah sebagai berikut. Proses menerima *input* berupa kunci 'K', data 'D' dan *file-chunk* 'F' yang akan digunakan untuk proses enkripsi. Setelah mendapatkan *input*, alur

berlanjut dengan pengambilan *index* proses pada CUDA. *Index* proses merupakan sebuah *identifier* unik sebuah proses paralel yang didapat dengan rumus Rumus 3.4.

$$I_{[illegible]} = Thr_{[illegible]} \quad I_{[illegible]} + B \quad I_{[illegible]} + B \quad D_{[illegible]} \quad \ldots \text{Rumus 3.4}$$

Thread Index merupakan *index* dari *thread* dimana proses berada, *Block Index* merupakan *index* dari *block* dimana *thread* berada dan *Block Dim* merupakan jumlah dari *thread* yang terdapat dalam sebuah *block.* Setelah mendapatkan *index* proses, maka proses akan dilanjutkan dengan pembentukan *key-state* sesuai dengan metode *key-scheduling* algoritma enkripsi. Setelah *key-state* berhasil dibentuk, maka alur akan berlanjut dengan proses PRG (*Pseudo-Random Generation*) atau proses enkripsi bit data. Data yang akan dienkripsi pada proses PRG didapat dengan mengambil data yang terdapat dalam penampung 'F' pada alamat *index* proses yang telah dihitung sebelumnya. Proses PRG akan dilakukan sebanyak jumlah bit yang terdapat pada data 'F' atau sebanyak panjang data dalam 'F'. Setelah proses PRG selesai dilakukan, maka hasil yang didapat adalah data 'F' pada alamat *index* proses yang telah terenkripsi. Hasil data tersebut akan dikembalikan ke *host* untuk ditulis sebagai *output* data.

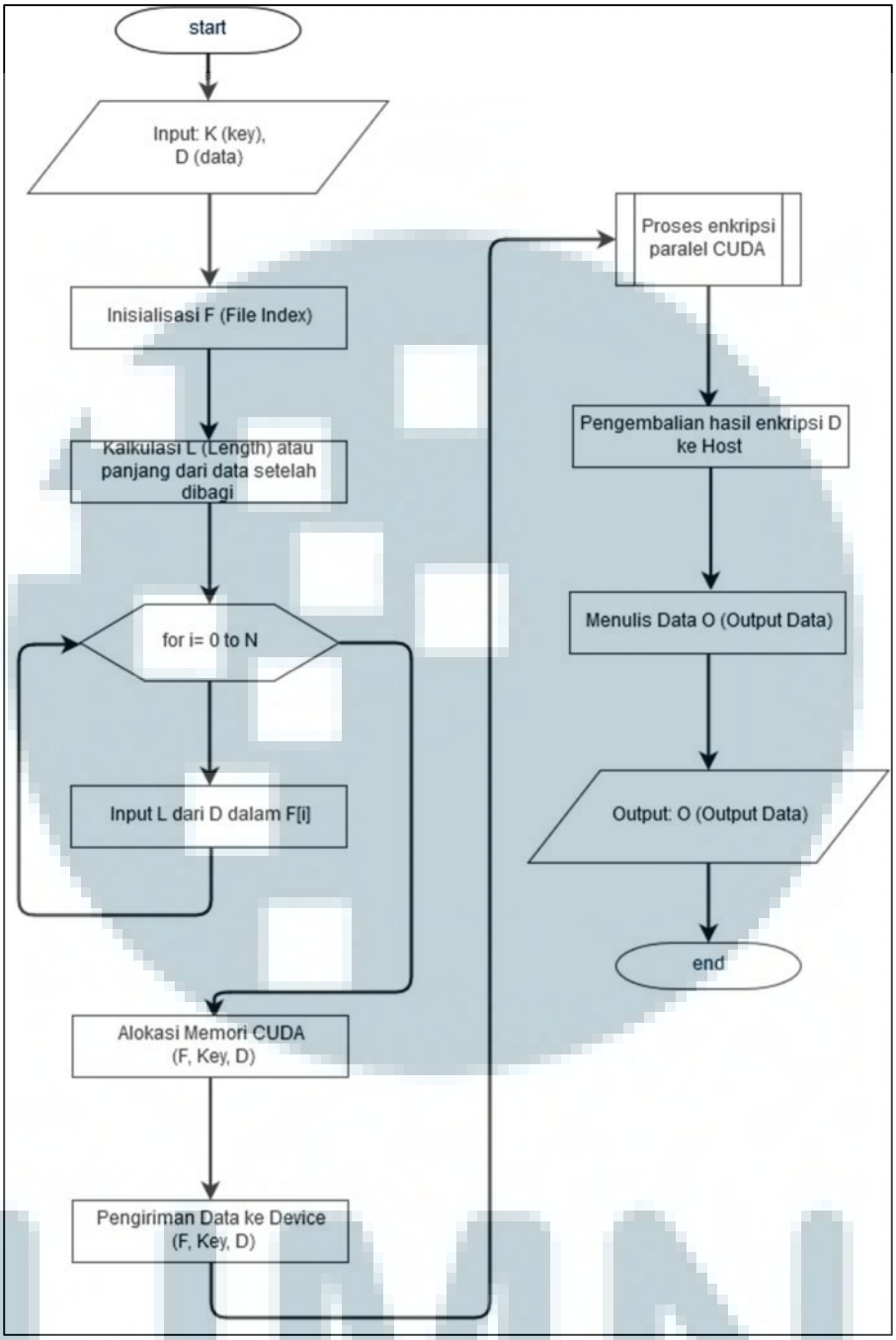
3.3.2 Perancangan Uji Coba Menggunakan File-Indexing

Pada uji coba menggunakan *file-indexing,* program kembali terbagi menjadi dua bagian yaitu program pada *host* dan program pada *device.* Uji coba menggunakan *file-indexing* dilakukan dengan pembagian *file input* dan menyimpan hasil dari pembagian tersebut sebagai *index* acuan untuk proses enkripsi. Tujuan dari uji coba menggunakan *file-indexing* adalah sebagai metode alternatif dari

proses *file-splitting* yang dilakukan pada uji coba menggunakan *file-splitting*. Penjabaran dari program *host* dan *device* akan dijelaskan pada subbab-subbab berikut ini.

A. Perancangan Program pada Host

Garis besar alur program pada uji coba yang akan dirancang adalah sebagai berikut, program menerima *input* berupa data *input* 'D' atau dokumen *input* yang akan dienkripsi beserta sebuah kunci rahasia yang akan digunakan pada proses enkripsi. Setelah mendapatkan *input* 'D', program akan melakukan inisialisasi variabel 'F' yang akan digunakan sebagai penampung *index* pada proses *file indexing*. Pada proses *file indexing*, dilakukan pencarian ukuran 'D' setelah dokumen dibagi menjadi 'N'-1 pecahan dimana 'N' merupakan jumlah paralel proses yang akan digunakan, serta sisa hasil bagi ukuran 'D' pada 'N'-1. Data-data tersebut kemudian dimasukkan ke dalam variabel 'F'. Setelah semua data lengkap, maka dilakukan alokasi data pada GPU untuk 'F', kunci rahasia dan 'D'. Setelah alokasi selesai, maka proses enkripsi akan dilakukan pada GPU secara paralel sebanyak jumlah 'N'. Setelah proses enkripsi selesai dilakukan, maka hasil dari enkripsi tersebut dikembalikan ke CPU. Setelah data dikembalikan, maka data yang telah dienkripsi akan ditulis menjadi 'O' atau *output* data yang merupakan hasil dari enkripsi. Seluruh alur proses program dapat digambarkan dengan sebuah diagram *flowchart*. Diagram *flowchart* yang mengambarkan proses program tersebut dapat dilihat pada Gambar 3.7 sebagai berikut.



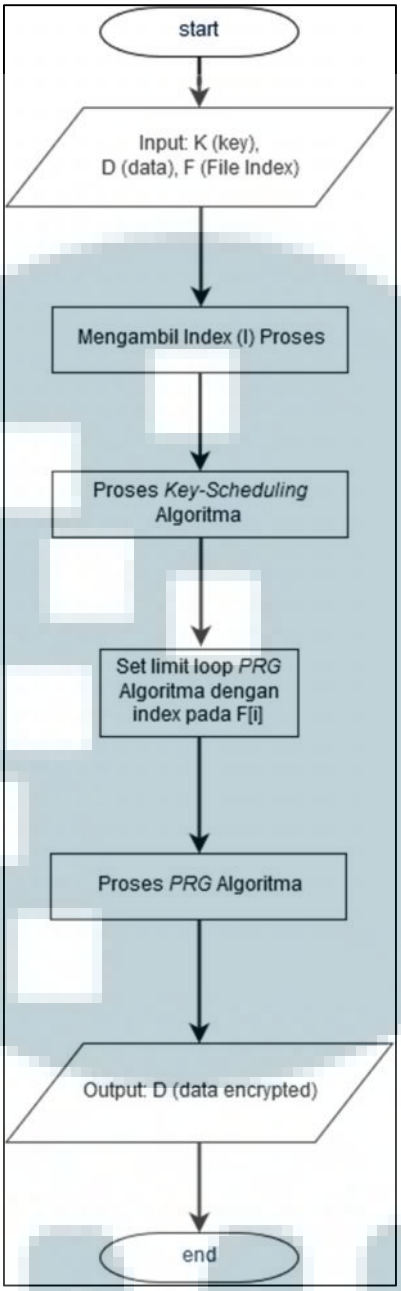
Gambar 3.7 *Flowchart* Proses Program pada *Host* pada Uji Coba menggunakan *File-Indexing*

Seperti yang dapat dilihat pada Gambar 3.7, alur proses program terbagi menjadi dua yaitu proses pada *host* atau CPU, dan proses pada *device* atau GPU. Data yang diterima di *host* harus dialokasi dan dikirimkan ke *device* sehingga

device dapat memproses data tersebut. Proses pengiriman data dari dan menuju *device* dapat memakan waktu yang cukup besar bila tidak dioptimalkan. Oleh karena hal tersebut, setelah dilakukan analisa pada kebutuhan algoritma, maka dapat ditentukan bahwa hanya tiga data penting yang dibutuhkan untuk proses enkripsi. Tiga data tersebut adalah dokumen PDF yang akan dienkripsi atau 'D', kunci rahasia, dan data *index* dari ukuran 'D' yang dimasukkan ke dalam variabel 'F'. Setelah data tersebut terkirim ke *device*, maka proses enkripsi secara paralel dapat dilakukan. Setelah proses paralel selesai, data 'D' yang telah terenkripsi akan dikembalikan ke *host* untuk dilakukan proses penulisan *output* data atau 'O'. *Output* data 'O' merupakan hasil akhir dari program.

B. Perancangan Implementasi Algoritma pada Device

Pada uji coba, untuk mengimplementasi enkripsi *stream cipher* dalam proses paralel, digunakanlah *file-indexing*. Tujuan dari *file-indexing* adalah untuk mengatur jumlah bit data yang akan dienkripsi pada paralel proses. Setelah dilakukan pengujian pada *file-splitting* dalam uji coba sebelumnya, maka uji coba akan menggunakan *file-indexing*. Setelah didapatkan garis besar alur implementasi, maka proses implementasi dapat digambarkan ke dalam sebuah *flowchart*. *Flowchart* yang menggambarkan alur implementasi algoritma dalam paralel proses dapat dilihat pada Gambar 3.8.



Gambar 3.8 *Flowchart* Algoritma pada *Device* dalam Uji Coba menggunakan *File-Indexing*

Seperti yang dapat dilihat pada Gambar 3.8, garis besar dari implementasi algoritma enkripsi RC4, RC4-2S, dan RC4itz dalam sebuah proses paralel adalah sebagai berikut. Proses menerima *input* berupa kunci 'K', data 'D' dan *file-index* 'F' yang akan digunakan untuk proses enkripsi. Setelah mendapatkan *input*, alur

berlanjut dengan pengambilan *index* proses pada CUDA. *Index* proses merupakan sebuah *identifier* unik sebuah proses paralel yang didapat dengan Rumus 3.4. *Index* proses kemudian akan digunakan dalam proses PRG.

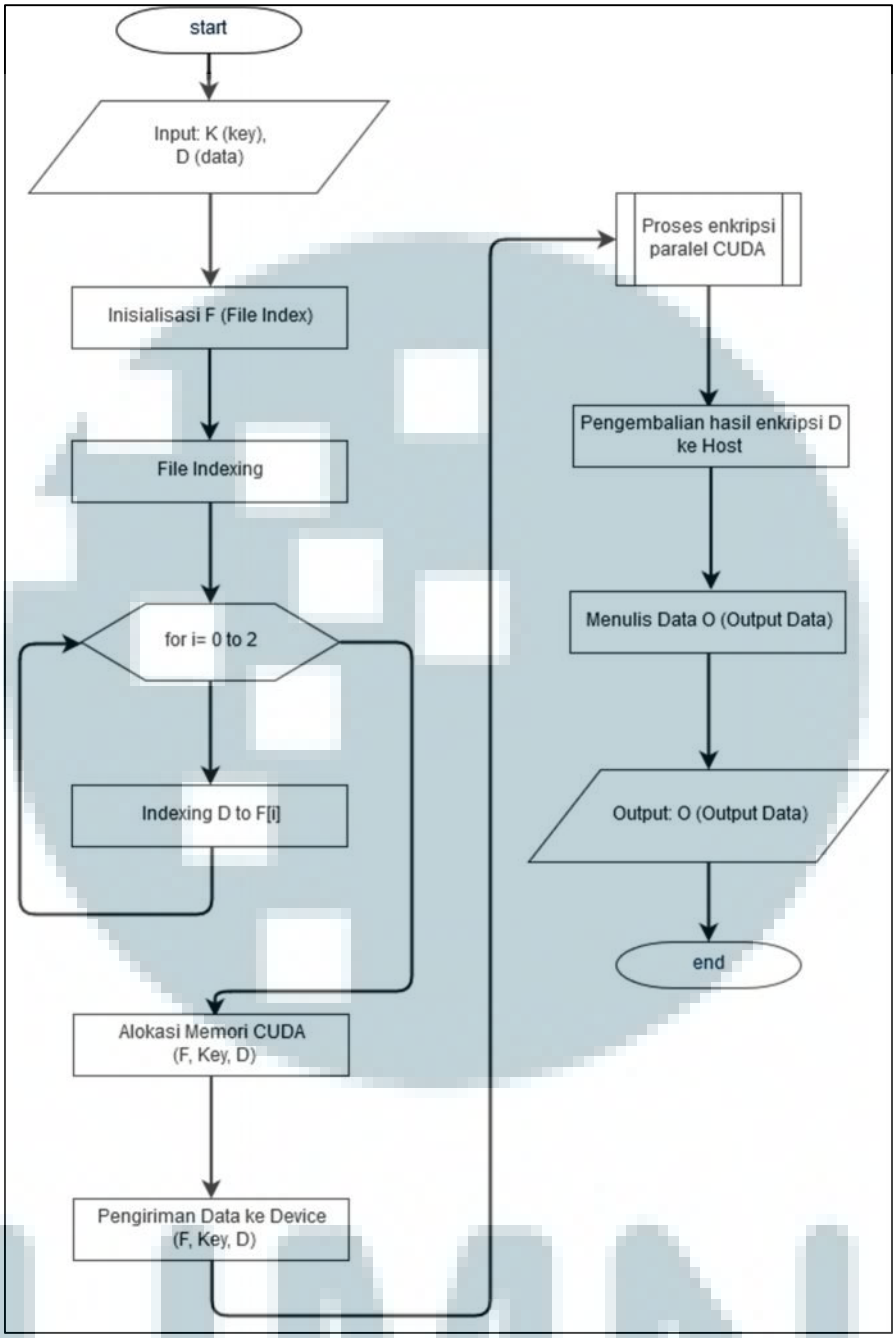
Setelah mendapatkan *index* proses, maka proses akan dilanjutkan dengan pembentukan *key-state* sesuai dengan metode *key-scheduling* algoritma enkripsi. Setelah *key-state* berhasil dibentuk, maka alur akan berlanjut dengan proses PRG atau proses enkripsi bit data. Pada proses ini, dilakukan proses enkripsi sebanyak jumlah data bit. Jumlah data bit dan posisi data yang akan terenkripsi dalam 'D' ditentukan dengan nilai dalam penampung 'F' pada alamat *index* proses. Setelah mendapatkan batas dari *loop* dan posisi awal *stream* bit data yang akan dienkripsi, maka proses PRG akan dimulai. Setelah proses PRG selesai dilakukan, maka hasil yang didapat adalah data 'D' yang telah terenkripsi. Hasil data tersebut akan dikembalikan ke *host* untuk ditulis sebagai *output* data.

3.3.3 Perancangan Uji Coba File-Indexing dengan Pengurangan Jumlah Array

Pada uji coba *file-indexing* dengan pengurangan jumlah *array*, program kembali terbagi menjadi dua bagian yaitu program pada *host* dan program pada *device*. Uji coba menggunakan metode *file-indexing* untuk melakukan pembagian *file input* dengan pendekatan yang berbeda dengan uji coba sebelumnya. Penjabaran dari program *host* dan *device* akan dijelaskan pada subbab-subbab berikut ini.

A. Perancangan Program pada Host

Garis besar alur program yang akan dirancang adalah sebagai berikut, awal dari program berlangsung sama dengan dua uji coba sebelumnya. Program menerima *input* berupa data *input* 'D' atau dokumen *input* yang akan dienkripsi beserta sebuah kunci rahasia yang akan digunakan pada proses enkripsi. Setelah mendapatkan *input* 'D', program akan melakukan inisialisasi variabel 'F' yang akan digunakan sebagai penampung *index* pada proses *file-indexing*. Proses *file-indexing* yang akan digunakan memiliki pendekatan berbeda dengan Proses *file-indexing* pada uji coba sebelumnya. Modifikasi proses *file-indexing* berlangsung sebagai berikut. Pertama, dilakukan pencarian ukuran 'D' setelah dokumen dibagi menjadi 'N'-1 pecahan. Ukuran tersebut kemudian dimasukkan ke dalam variabel penampung F pada alamat pertama atau 'F[0]'. Setelah didapat nilai pertama, maka 'F' pada alamat kedua atau 'F[1]' diisi dengan sisa hasil bagi ukuran panjang 'D' dengan nilai pada 'F[0]'. Data-data tersebut kemudian dimasukkan ke dalam variabel 'F'. Setelah semua data lengkap, maka dilakukan alokasi data pada GPU untuk 'F', kunci rahasia dan 'D'. Setelah alokasi selesai, maka proses enkripsi akan dilakukan pada GPU secara paralel sebanyak jumlah 'N'. Setelah proses enkripsi selesai dilakukan, maka hasil dari enkripsi tersebut dikembalikan ke CPU. Setelah data dikembalikan, maka data yang telah dienkripsi akan ditulis menjadi 'O' atau *output* data yang merupakan hasil dari enkripsi. Seluruh alur proses program dapat digambarkan dengan sebuah diagram *flowchart*. Diagram *flowchart* yang mengambarkan proses program tersebut dapat dilihat pada Gambar 3.9 sebagai berikut.



Gambar 3.9 *Flowchart* Proses Program pada *Host* pada Uji Coba menggunakan *File-Indexing* dengan Pengurangan *Array*

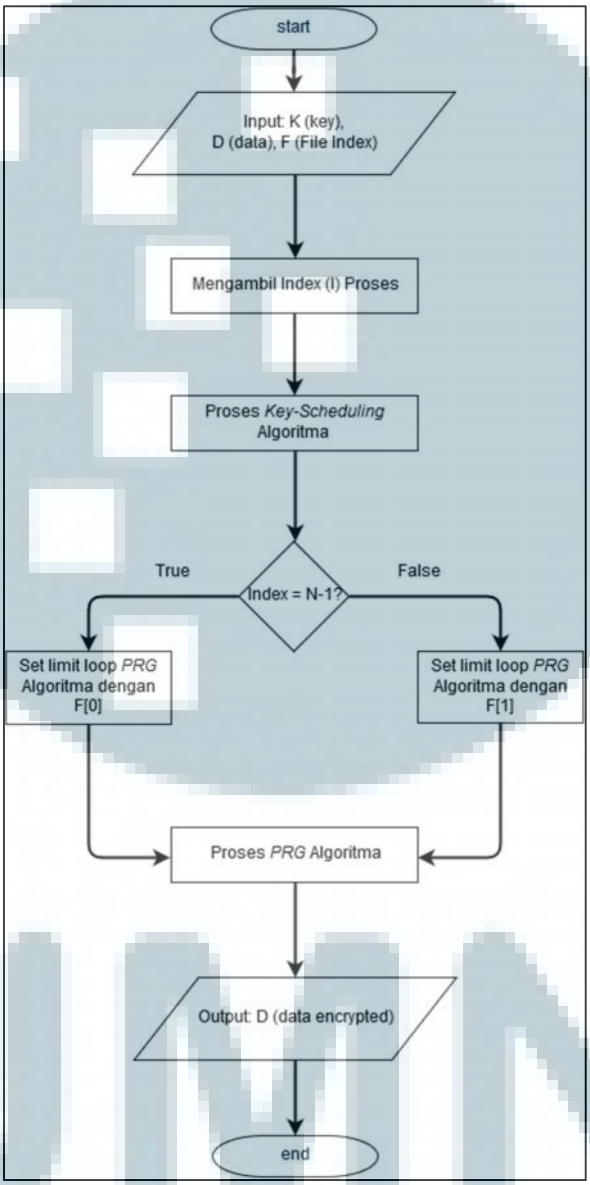
Seperti yang dapat dilihat pada Gambar 3.9, alur proses program terbagi menjadi dua yaitu proses pada *host* atau CPU, dan proses pada *device* atau GPU. Data yang diterima di *host* harus dialokasi dan dikirimkan ke *device* sehingga *device* dapat memproses data tersebut. Proses pengiriman data dari dan menuju

device dapat memakan waktu yang cukup besar bila tidak dioptimalkan. Setelah dilakukan analisa lebih lanjut mengenai data yang diperlukan dalam proses enkripsi, maka variabel 'F' yang menampung *index* data pada uji coba sebelumnya dapat disederhanakan dari sebuah variabel penampung dengan panjang sebesar 'N' menjadi variabel penampung dengan panjang dua. Variabel 'F' dapat disederhanakan dengan hanya menyimpan ukuran data 'D' setelah dipecah sebesar 'N'-1, dan sisa hasil bagi ukuran panjang 'D' dengan nilai sebelumnya. Inti dari penyederhanaan variabel 'F' adalah untuk mengurangi waktu alokasi, pengiriman data, serta pengambilan data dalam proses enkripsi. Tiga data yang akan dikirim adalah data *input* 'D', kunci rahasia 'K', dan data *index* dalam variabel penampung 'F'. Setelah data tersebut terkirim ke *device*, maka proses enkripsi secara paralel dapat dilakukan. Hasil dari proses paralel berupa data 'D' yang telah terenkripsi. Data tersebut dikembalikan ke *host* untuk dilakukan proses penulisan *output* data atau 'O'. *Output* data 'O' merupakan hasil akhir dari program.

B. Perancangan Implementasi Algoritma pada Device

Untuk mengimplementasikan algoritma *stream cipher* RC4, RC4-2S dan RC4itz ke dalam sebuah paralel proses pada uji coba *file-indexing* dengan pengurangan *array*, diterapkanlah proses *file-indexing* dengan pendekatan yang berbeda dengan proses *file-indexing* uji coba sebelumnya. Pada proses *file-indexing* uji coba dengan pengurangan jumlah *array*, data yang dimasukkan ke dalam variabel penampung 'F' hanya berupa ukuran dari data 'D' setelah dibagi 'N'-1 dimana 'N' merupakan jumlah dari proses paralel, serta sisa hasil bagi panjang dari 'D' dengan nilai sebelumnya. *Index* tersebut akan digunakan sebagai batas *loop*

serta penentu posisi data ‘D’ yang akan dienkripsi pada setiap proses paralel. Setelah didapatkan garis besar alur implementasi, maka proses implementasi dapat digambarkan dalam sebuah *flowchart. Flowchart* yang menggambarkan alur implementasi algoritma dalam paralel proses dapat dilihat pada Gambar 3.10.



Gambar 3.10 *Flowchart* Algoritma pada *Device* dalam Uji Coba *File-Indexing* dengan Pengurangan Jumlah *Array*

Seperti yang dapat dilihat pada Gambar 3.10, garis besar dari implementasi algoritma enkripsi RC4, RC4-2S, dan RC4itz dalam sebuah proses paralel adalah

sebagai berikut. Proses menerima *input* berupa kunci ‘K’, data ‘D’ dan *file-index* ‘F’ yang akan digunakan untuk proses enkripsi. Setelah mendapatkan *input*, alur berlanjut dengan pengambilan *index* proses pada *device*. *Index* proses didapatkan dengan Rumus 3.4 dan akan digunakan dalam proses PRG dan untuk menentukan nilai dari variabel ‘F’ yang akan digunakan.

Setelah mendapatkan *index* proses, maka proses akan dilanjutkan dengan pembentukan *key-state* sesuai dengan metode *key-scheduling* algoritma enkripsi. Setelah *key-state* berhasil dibentuk, maka alur akan berlanjut dengan sebuah *conditional statement* sebelum proses PRG atau proses enkripsi bit data. Pada proses ini, dilakukan sebuah pengecekan *index* proses dengan jumlah proses paralel yang berjalan adalah sebanyak ‘N’. Bila proses bukan merupakan proses terakhir atau *index* tidak sama dengan ‘N’-1, maka proses akan memilih ‘F[0]’ sebagai jumlah dari *loop* yang akan dilakukan pada proses PRG. Bila proses merupakan proses terakhir atau *index* sama dengan ‘N’-1, maka proses akan memilih ‘F[1]’ sebagai jumlah dari *loop* yang akan dilakukan pada proses PRG.

Setelah mendapatkan batas dari *loop*, proses akan berlanjut dengan proses PRG dimana data akan dienkripsi. Jumlah data yang akan dienkripsi dalam setiap proses bergantung terhadap batas *loop* yang telah ditentukan pada *conditional statement* sebelum proses PRG. Posisi data D yang akan dienkripsi didapatkan dengan Rumus 3.5.

$$P = I \quad P * N \quad F[0] \qquad \text{... Rumus 3.5}$$

Nilai ‘F[0]’ digunakan dalam menentukan posisi awal setiap proses paralel PRG karena nilai ‘F[0]’ merupakan panjang dari pecahan data ‘D’ yang dibagi

dengan nilai ‘N’-1 sedangkan nilai ‘F[1]’ menampung sisa hasil bagi panjang data ‘D’ dengan nilai ‘F[0]’ yang akan menjadi batas *loop* dari proses paralel terakhir. Setelah proses PRG selesai dilakukan, maka hasil yang didapat adalah data ‘D’ yang telah terenkripsi. Hasil data tersebut akan dikembalikan ke *host* untuk ditulis sebagai *output* data.

3.3.4 Perancangan Uji Coba dengan Perubahan Jumlah Proses Paralel

Pada uji coba perubahan jumlah proses paralel, dilakukan peningkatan jumlah proses paralel program pada *device* untuk mempercepat proses enkripsi. Dengan menggunakan uji coba sebelumnya sebagai dasar dari uji coba perubahan jumlah proses paralel, jumlah proses paralel ditingkatkan secara bertahap dengan peningkatan sebagai berikut; 16, 64, 256, 1024, dan 4096 buah proses paralel. Tujuan dari uji coba adalah untuk meneliti pengaruh jumlah proses paralel terhadap *throughput* dari proses enkripsi.

3.3.5 Perancangan Analisa Hasil Implementasi

Setelah uji coba berhasil dilaksanakan, hasil dari uji coba yang telah dilakukan dianalisa. Analisa dilakukan dengan melakukan perbandingan dari hasil yang didapatkan pada uji coba dalam bentuk tabel dan grafik batang serta grafik garis. Selain membandingkan hasil uji coba, hasil-hasil tersebut juga dibandingkan dengan nilai yang didapat pada saat algoritma dijalankan secara sekuensial. Nilai tersebut dapat berupa rata-rata waktu enkripsi atau nilai *throughput* dari algoritma yang dijalankan secara sekuensial.