Perancangan dan Implementasi Multiple Audio Files Secret Sharing dengan Penggunaan Pembangkit Bilangan Acak Blum Blum Shub dan Optimasi Multiprocessing dan Asynchronous Programming

Malik Akbar Hashemi Rafsanjani - 13520105
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): pro.malikakbar2357@gmail.com

***Abstract*—Makalah ini mempresentasikan pendekatan untuk implementasi *secret sharing* untuk beberapa berkas suara kepada beberapa *shareholder* dengan optimalisasi berupa penggunaan *multiprocessing* dan *asynchronous programming*. *Secret Sharing* merupakan sebuah teknik kriptografis yang melibatkan pembagian data rahasia menjadi beberapa bagian, yang kemudian didistribusikan di antara *shareholder* yang berbeda yang dapat direkonstruksi menjadi data awal ketika jumlah bagian mencukupi. Makalah ini berfokus pada pembagian beberapa berkas suara secara sekaligus dan dioptimasi dengan *multiprocessing* dan *asynchronous programming* untuk mempercepat proses pembagian dan rekonstruksi. Dalam proses pembangkitan bagian digunakan algoritma Blum Blum Shub sebagai pembangkit bilangan acak.**

Keywords—secret sharing; berkas suara; multiprocessing; asynchronous programming; pembangkit bilangan acak; Blum Blum Shub; kriptografi

I. Introduction

Pada zaman yang semakin modern ini, penggunaan teknologi juga semakin berkembang. Perkembangan ini memungkinkan komunikasi yang semakin kompleks beserta tuntutan terhadap kerahasiaan informasi. Dalam kegiatan yang penting, seperti transaksi bisnis, penyimpanan barang berharga, atau data pribadi rahasia; diperlukan keamanan pertukaran informasi agar pihak yang terlibat tidak mengalami kerugian. Walaupun informasi atau data tersebut tidak ditujukan kepada pihak lain, ada kemungkinan data tersebut dapat tersebar dan dilihat dengan bebas tanpa sepengetahuan pemilik yang kemudian mengakibatkan kebocoran informasi.

Adanya kerentanan terhadap komunikasi mendorong pemastian kerahasiaan dan integritas data menjadi hal yang terpenting. Organisasi, pemerintah, dan individu sama-sama menghadapi tantangan untuk melindungi informasi berharga dari akses tidak sah, pencurian, atau kehilangan yang tidak disengaja. Teknik kriptografi tradisional, seperti enkripsi, memainkan peran penting dalam mengamankan data. Namun, mereka sering mengandalkan penggunaan satu kunci atau kata sandi untuk mengenkripsi dan mendekripsi informasi, membuat mereka rentan terhadap serangan jika kuncinya disusupi, diambil, dirusak, atau kerentanan lainnya.

Kerentanan ini muncul dari risiko bawaan dari penyimpanan dan pengelolaan satu kunci. Jika musuh mendapatkan akses ke kunci, mereka dapat mendekripsi data terenkripsi dan berpotensi mendapatkan akses tidak sah ke informasi sensitif. Selain itu, kehilangan atau kerusakan kunci dapat mengakibatkan hilangnya data terenkripsi secara permanen, membuatnya tidak dapat dipulihkan.

Untuk mengatasi keterbatasan ini, teknik *secret sharing* dikembangkan sebagai sarana untuk mendistribusikan tanggung jawab menjaga informasi sensitif di antara banyak entitas. Ide di balik *secret sharing* adalah membagi rahasia menjadi beberapa bagian, yang kemudian dibagikan kepada sekelompok peserta. Prinsip utamanya adalah tidak ada peserta individu yang memiliki cukup informasi untuk merekonstruksi rahasianya sendiri. Sebagai gantinya, ambang pembagian yang telah ditentukan diperlukan untuk merekonstruksi rahasia asli.

Teknik ini juga berguna pada berkas suara. Banyak berkas suara bersifat rahasia, seperti rekaman percakapan rahasia poltikus, rekaman suara pertemuan transaksi bisnis, dan rekaman suara eksperimen yang dirahasiakan. Hal ini mendorong adanya pengaplikasian *secret sharing* pada berkas suara.

Namun, pada umumnya, tidak hanya ada satu berkas suara yang perlu dilindungi kerahasiaannya. Hal ini dapat dilakukan teknik *secret sharing* pada masing-masing berkas suara. Akan tetapi, hal ini akan membawa kerentanan tambahan karena perlunya pengelolaan banyak bagian dari data rahasia secara sekaligus. Hal ini mendorong dikembangkannya teknik lanjutan yaitu *multi-secret sharing* yang bertujuan agar banyak rahasia dilindungi secara bersamaan, dan peningkatan efisiensi, karena proses distribusi dan rekonstruksi dapat dilakukan bersama.

Pada makalah ini, penulis mencoba untuk merancang dan mengimplementasikan teknik *multi-secret sharing* pada

beberapa berkas suara secara sekaligus. Implementasi ini menggunakan algoritma Blum Blum Shub (BBS) sebagai pembangkit bilangan acak. Selain itu, dilakukan optimasi berupa *multiprocessing* dan *asynchronous programming* untuk meningkatkan kinerja program.

II. LANDASAN TEORI

A. Secret Sharing

Secret Sharing merupakan metode untuk mendistribusikan rahasia di antara kelompok, sedemikian rupa sehingga tidak ada individu yang memiliki informasi yang dapat dipahami tentang rahasia tersebut, tetapi ketika sejumlah individu menggabungkan 'bagian' mereka, rahasia tersebut dapat direkonstruksi [2]. Skema *secret sharing* merupakan skema ideal untuk menyimpan informasi yang sangat sensitif dan sangat penting. Contohnya meliputi: kunci enkripsi, kode peluncuran misil, dan rekening bank bernomor. Masing-masing informasi ini harus dijaga kerahasiaannya, karena pemaparannya dapat menjadi bencana; namun, penting juga agar mereka tidak hilang.

Secret sharing juga memungkinkan pemegang data rahasia awal untuk mempercayai grup secara kesatuan. Secara tradisional, memberikan rahasia kepada suatu grup untuk diamankan akan mengharuskan distributor mempercayai sepenuhnya semua anggota grup. Skema *secret sharing* memungkinkan distributor untuk menyimpan rahasia dengan aman bersama grup meskipun tidak semua anggota dapat dipercaya sepanjang waktu. Selama jumlah pengkhianat tidak pernah lebih dari jumlah kritis yang dibutuhkan untuk merekonstruksi rahasia, rahasianya aman.

Algoritma *secret sharing* yang aman memastikan seseorang dengan bagian berjumlah kurang dari *threshold* tidak memiliki informasi yang lebih banyak daripada seseorang yang tidak memiliki bagian. Sedangkan *secret sharing* yang tidak aman memungkinkan penyerang mendapatkan lebih banyak informasi dengan setiap pembagian.

Jumlah *shareholder* dan jumlah batas bagian untuk rekonstruksi pada skema *secret sharing* dapat dideskripsikan dengan skema ambang. Skema ambang (T, N) merupakan metode pembagian data rahasia kepada N *shareholder* sedemikian sehingga rekonstruksi data rahasia hanya dapat dilakukan oleh sembarang himpunan bagian yang terdiri dari T *shareholder* atau lebih.

Multi-secret sharing merupakan teknik kriptografis yang melibatkan pemecahan banyak data rahasia secara sekaligus menjadi bagian-bagian dan didistribusikan ke beberapa *shareholder*. Tidak seperti skema *secret sharing* umum yang berfokus pada satu data rahasia, *multi-secret sharing* memungkinkan distribusi yang aman dan rekonstruksi banyak data rahasia secara bersamaan [3].

Tujuan utama *multi-secret sharing* adalah untuk memastikan bahwa setiap rahasia tetap rahasia dan hanya dapat direkonstruksi ketika jumlah pembagian yang memadai digabungkan. Pendekatan ini memberikan lapisan keamanan tambahan, karena mengompromikan satu bagian tidak mengungkapkan informasi apapun tentang rahasia lainnya. Keuntungan dari *multi-secret sharing* termasuk peningkatan keamanan, karena banyak rahasia dilindungi secara bersamaan, dan peningkatan efisiensi, karena proses distribusi dan rekonstruksi dapat dilakukan bersama.

B. Multiprocessing

Multiprocessing mengacu pada kemampuan sistem komputer untuk mengeksekusi banyak tugas atau proses secara konkuren, memanfaatkan beberapa prosesor. *Multiprocessing* menjalankan beberapa proses secara bersamaan untuk meningkatkan efisiensi dan meningkatkan kinerja sistem secara keseluruhan.

Secara tradisional, sebagian besar komputer memiliki satu unit pemrosesan pusat (CPU) yang menjalankan instruksi secara berurutan. Namun, seiring kemajuan teknologi, komputer mulai menggabungkan banyak CPU atau inti prosesor pada satu chip, memungkinkan eksekusi beberapa proses secara bersamaan. Secara keseluruhan, multiprocessing adalah teknik ampuh yang memanfaatkan kemampuan sistem komputer modern untuk meningkatkan kinerja, skalabilitas, dan kemampuan multitasking.

Manfaat utama dari multiprocessing meliputi:

- Peningkatan *throughput*

  Dengan mengeksekusi beberapa proses secara bersamaan, *multiprocessing* memungkinkan sistem menangani beban kerja yang lebih tinggi dan menyelesaikan lebih banyak tugas dalam jumlah waktu tertentu.

- Peningkatan kinerja

  Ketika sistem memiliki banyak prosesor, sistem dapat mendistribusikan beban kerja di antara mereka, mengurangi waktu yang diperlukan untuk menyelesaikan tugas yang kompleks. Ini mengarah pada peningkatan kinerja sistem.

- Peningkatan *Multitasking*

  Multiprocessing memfasilitasi menjalankan beberapa aplikasi atau tugas secara bersamaan tanpa penurunan kinerja yang signifikan. Setiap proses dapat memiliki sumber daya khusus, memungkinkan multitasking yang lebih lancar.

C. Asynchronous Programming

Asynchronous Programming merupakan suatu teknik pemrograman yang memungkinkan tugas dieksekusi secara konkuren, tanpa menghalangi eksekusi program. Teknik ini sangat berguna ketika berhadapan dengan operasi atau tugas yang memakan waktu yang melibatkan menunggu sumber daya eksternal, seperti *network request* atau operasi *input/output*.

Dalam pemrograman sinkron tradisional, setiap tugas dijalankan secara berurutan, dan program menunggu tugas selesai sebelum melanjutkan ke tugas berikutnya. Hal ini dapat menyebabkan inefisiensi, terutama ketika ada tugas yang melibatkan menunggu operasi *input/output* selesai.

Sebaliknya, pemrograman asinkron memungkinkan untuk memulai tugas dan terus menjalankan tugas lain tanpa menunggu tugas pertama selesai. Ini dicapai dengan menggunakan fungsi atau metode asinkron, yang memungkinkan program untuk menjadwalkan tugas dan menerima pemberitahuan atau hasil saat sudah siap.

Pendekatan umum untuk pemrograman asinkron adalah dengan menggunakan *callback* atau *promise*. *Callback* adalah fungsi yang diteruskan sebagai argumen ke fungsi asinkron, dan dipanggil saat tugas selesai. *Promise*, di sisi lain, mewakili penyelesaian akhir atau kegagalan operasi asinkron, dan mereka memungkinkan untuk menggabungkan beberapa operasi asinkron.

Dengan adanya teknik *async/await* dalam bahasa pemrograman modern, pemrograman asinkron menjadi lebih *developer friendly*. *Async/await* memungkinkan untuk menulis kode asinkron yang menyerupai kode sinkron, membuatnya lebih mudah untuk dipikirkan dan dipelihara. Ini memungkinkan untuk menulis kode yang terlihat berurutan tetapi sebenarnya berjalan secara asinkron.

Dengan menggunakan teknik pemrograman asinkron, kecepatan respon dan efisiensi aplikasi dapat ditingkatkan. Hal ini memungkinkan untuk memanfaatkan sumber daya sistem secara lebih efektif, karena program dapat terus menjalankan tugas lain sambil menunggu operasi yang lebih lambat selesai. Ini sangat bermanfaat dalam skenario di mana banyak tugas dapat dijalankan secara konkuren, yang mengarah pada peningkatan kinerja dan kecepatan respon.

D. Pembangkit Bilangan Acak Blum Blum Shub

Pembangkit bilangan acak merupakan algoritma matematis atau perangkat fisik yang dapat menghasilkan urutan angka yang tampak acak. Ide utama di balik pembangkit bilangan acak adalah menghasilkan angka yang tidak dapat diprediksi dan secara statistik tidak bergantung satu sama lain. Bilangan acak banyak digunakan dalam kriptografi, antara lain digunakan pada hal-hal berikut.

- Pembangkitan nilai-nilai parameter kunci pada algoritma kunci-publik
- Pembangkitan kunci sesi oleh klien pada protokol SSL.
- Pembangkitan nilai acak pada algoritma enkripsi ElGamal.

Prosedur komputasi tidak ada yang dapat menghasilkan deret bilangan acak yang benar-benar sempurna (*truly random*). Bilangan acak dari prosedur komputasi biasa disebut bilangan acak semu (*pseudo-random*) karena hasil dapat diulang jika parameter atau kunci (umpan) yang digunakan sama. Maka dari itu, pembangkit bilangan acak semu dikatakan bersifat deterministik. Namun, hal ini tidak membatasi pembangkit bilangan acak semu untuk bermanfaat secara kriptografi karena beberapa algoritma tersebut merupakan algoritma yang aman secara kriptografi.

Salah satu algoritma pembangkit bilangan acak semu yang aman secara kriptografi adalah Blum Blum Shub (BBS) [4]. Algoritma ini memenuhi prasyarat sebagai *Cryptographically Secure PseudoRandom Generator* (CSPRNG), yaitu sebagai berikut.

- Secara statistik lolos uji keacakan
- Tahan terhadap serangan yang serius yang bertujuan untuk memprediksi bilangan acak yang dihasilkan dari nilai-nilai sebelumnya.

Dipilihnya algoritma BBS pada makalah ini karena algoritma BBS terbukti aman dan perhitungan yang dilakukan tidak terlalu besar. Keamanan dari BBS didasari pada hal berikut.

- Membedakan bit-bit luaran dengan bit acak merupakan hal yang sukar. Seminimalnya sesukar memecahkan persoalan *quadratic residue problem*.
- Pemfaktoran angka n pada BBS untuk n yang besar merupakan hal yang sukar

Hal ini mengakibatkan algoritma BBS tidak dapat diprediksi dari arah kiri dan dari arah kanan. Hal ini berarti, jika terdapat barisan bit yang dihasilkan BBS, sangat sukar untuk memprediksi barisan bit sebelum maupun sesudah barisan bit yang diberikan [1].

III. Rencana Penyelesaian Masalah

Seperti yang telah dijelaskan sebelumnya, *secret sharing* merupakan skema yang memungkinkan pembagian data rahasia secara aman. Selain itu, skema tersebut dapat dikembangkan menjadi *multi-secret sharing*, untuk membagi beberapa data rahasia secara sekaligus.

Pada makalah ini, penulis memilih menggunakan skema ambang (N, N) yaitu skema yang membagi data rahasia ke N *shareholder* sedemikian sehingga memerlukan N *shareholder* pula untuk merekonstruksi data rahasia awal. Skema ini berguna untuk memastikan data rahasia hanya dapat direkonstruksi jika seluruh *shareholder* berpartisipasi.

Dalam implementasi skema *secret sharing* ini, algoritma yang digunakan adalah algoritma dengan fungsi XOR. Kelebihan algoritma ini adalah algoritma ini memastikan hasil rekonstruksi merupakan data yang tepat sama dengan data rahasia awal. Selain itu, ukuran bagian kurang lebih sama dengan ukuran data semula sehingga ruang yang dipakai tidak jauh lebih besar dibanding ukuran ruang data semula.

Adanya pembagian beberapa data secara sekaligus akan membuat proses semakin lama. Hal ini mendorong adanya optimasi dengan *multiprocessing* dan *asynchronous programming*. Hal ini sesuai karena banyak proses yang independen satu sama lain, sehingga dapat dilakukan secara sekaligus.

Multiprocessing akan digunakan untuk melakukan proses yang bersifat independen dan memerlukan pemrosesan yang banyak membutuhkan proses CPU, dalam kasus ini adalah pembangkitan bilangan acak karena memerlukan komputasi yang cukup besar. Sementara itu, *asynchronous programming* cocok untuk digunakan pada proses yang banyak menunggu *input/output,* dalam kasus ini adalah pembacaan dan penulisan berkas suara.

A. Rancangan Solusi dan Arsitektur

Berikut ini merupakan rancangan solusi yang diterapkan pada makalah ini. Penjelasan lebih mendetail mengenai pemrosesan akan dijelaskan pada bagian implementasi.

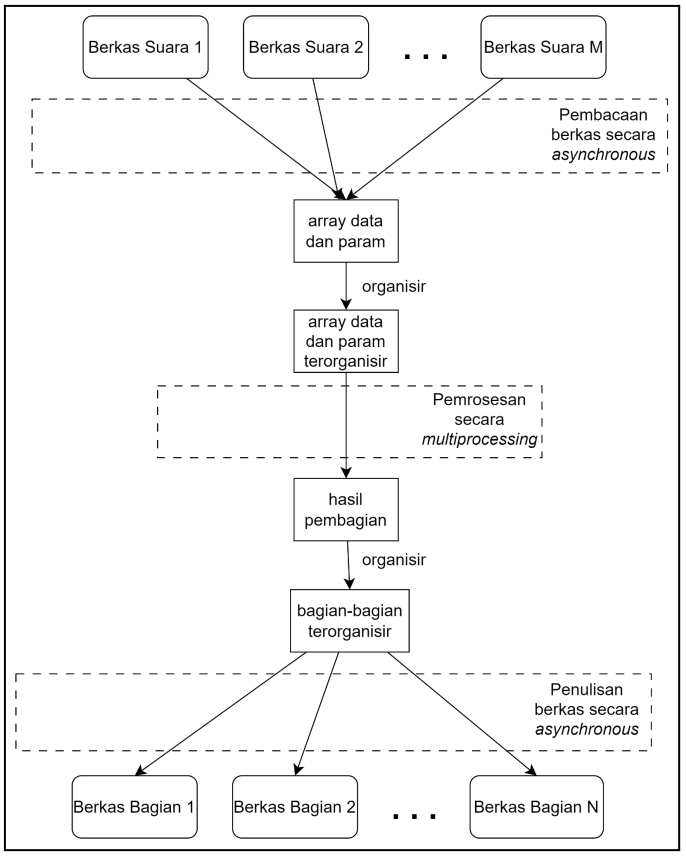

Fig. 1. Visualisasi rancangan solusi program pembangkitan *shares*

Gambar di atas merupakan rancangan solusi untuk proses pembangkitan *shares*. Misalkan terdapat M berkas suara, program akan membaca berkas-berkas tersebut secara asinkron dan menghasilkan suatu array dari data dan parameter dari berkas suara tersebut. Parameter perlu disimpan agar nantinya berkas awal dapat direkonstruksi ulang sesuai dengan parameter awalnya pula. Lalu array data dan parameter tersebut diorganisir ulang agar lebih mudah diproses.

Array yang telah terorganisir ulang tadi diproses secara *multiprocessing* sehingga didapatkan hasil pembangkitan *shares*. Hasil tersebut diorganisir ulang untuk mempermudah penulisan berkas *shares*. Lalu, hasil tersebut dituliskan secara asinkron menjadi berkas-berkas suara baru.

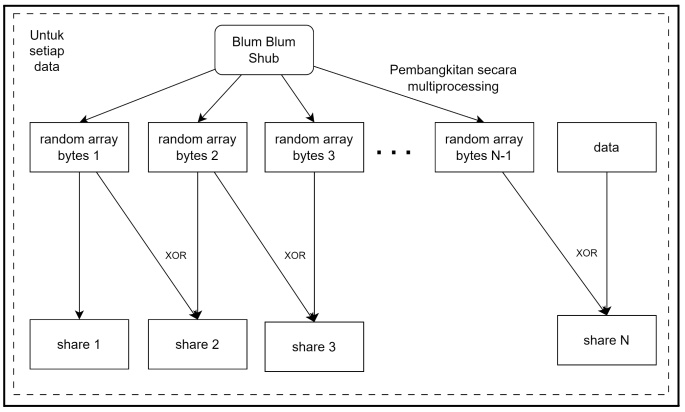

Fig. 2. Visualisasi rancangan solusi proses pembangkitan *shares*, terkhusus bagian multiprocessing

Gambar di atas merupakan rancangan solusi untuk proses pembangkitan *shares*, khususnya pada bagian *multiprocessing*. Untuk setiap data yang ada, algoritma BBS membangkitkan data acak secara multiprocessing karena pembangkitan data acak bersifat independen satu sama lain. Data acak yang didapat bersama data awal lalu dilakukan proses pembangkitan share sejumlah yang diminta.

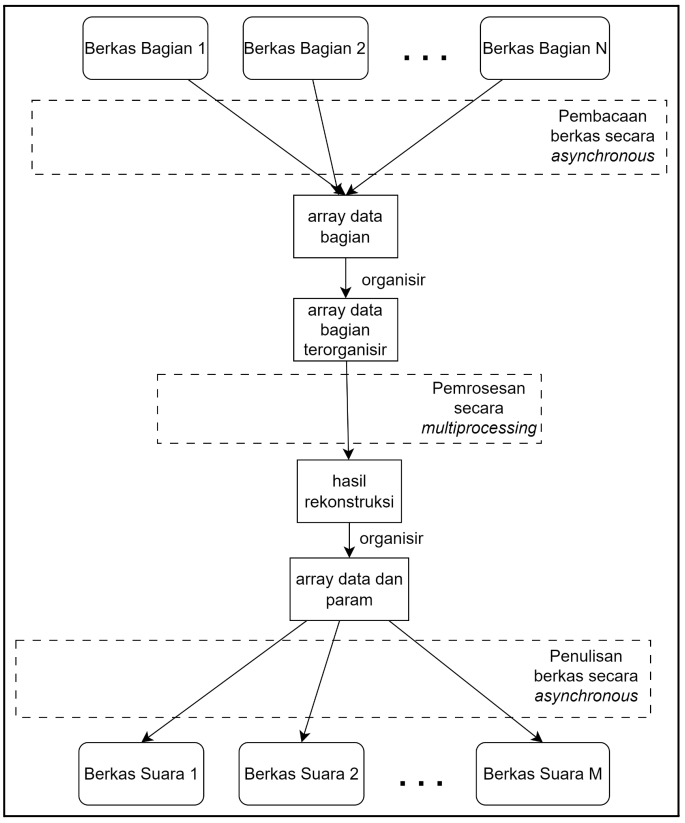

Fig. 3. Visualisasi rancangan solusi program rekonstruksi

Gambar di atas merupakan rancangan solusi untuk proses rekonstruksi data awal. Misalkan terdapat N berkas *shares*, program akan membaca berkas-berkas tersebut secara asinkron dan menghasilkan suatu array dari data dan parameter dari berkas suara tersebut. Parameter dari berkas

suara dapat diabaikan karena bukan merupakan parameter asli dan tidak menyimpan data yang diperlukan. Lalu array data tersebut diorganisir ulang agar lebih mudah diproses.

Array yang telah terorganisir ulang tadi diproses secara *multiprocessing* sehingga didapatkan hasil rekonstruksi data awal. Hasil tersebut diorganisir ulang untuk mempermudah penulisan berkas data awal. Lalu, hasil tersebut dituliskan secara asinkron menjadi berkas-berkas suara awal.

B. Batasan dan Asumsi

Berdasarkan rancangan yang telah diajukan, penulis membuat implementasi untuk mendemonstrasikan cara kerja dari skema multiple audio files secret sharing yang telah disebutkan. Berikut merupakan batasan dan asumsi yang digunakan dalam proses implementasi.

- Bahasa pemrograman yang digunakan adalah Python.
- Algoritma Blum Blum Shub diimplementasikan sendiri dengan bilangan P dan Q yang digunakan adalah 30000000091 dan 40000000003, serta *seed* yang digunakan adalah 396676160873959238473. Bilangan P dan Q tersebut dipilih karena memenuhi kondisi yang diperlukan, berukuran cukup besar, dan banyak digunakan untuk pembangkitan bilangan acak BBS. Sedangkan *seed* yang dipakai dibangkitkan secara acak dengan program lain dan divalidasi jika memang memenuhi kondisi yang diperlukan untuk *seed* algoritma BBS.
- Berkas suara yang digunakan adalah berkas suara berformat .wav dengan digunakan pustaka wave untuk mengakses dan menulis berkas suara.
- Digunakan pustaka asyncio sebagai pustaka dalam implementasi *asynchronous programming.*
- Digunakan pustaka multiprocessing sebagai pustaka dalam implementasi *multiprocessing*.
- Digunakan pembatas b'\xff\xff\x00\x00\xff\xff' dalam organisasi data frame suara dan parameter berkas dan diasumsikan tidak ada bytes dengan pola sama pada data frame suara atau parameter berkas.
- Fokus utama implementasi adalah untuk menunjukkan cara kerja dan validasi hasil skema pembagian dan rekonstruksi beberapa berkas suara sekaligus.
- Dilakukan pembandingan kinerja terhadap proses secara *synchronous* dengan analisis dan visualisasi dibantu dengan pustaka cProfile, pstats, dan skaneviz.
- Digunakan kasus dengan ukuran berkas dan jumlah kecil karena berdasarkan percobaan yang dibatalkan dengan ukuran berkas besar dan jumlah banyak mengakibatkan proses berjalan dengan waktu lama.

C. Implementasi Pembangkit Bilangan Acak Blum Blum Shub

Seperti yang telah dijelaskan sebelumnya, digunakan bahasa Python, termasuk dalam implementasi pembangkit bilangan acak Blum Blum Shub (BBS). BBS digunakan pada proses pembangkitan *shares* yang membutuhkan bytes acak sejumlah ukuran data. Maka dari itu, diperlukan pembangkit bilangan acak yang dapat membangkitkan bytes acak atau bilangan bulat di antara 0 sampai 255. Algoritma BBS yang diimplmentasikan dilakukan *post processing* agar dapat memenuhi kebutuhan tersebut.

Algoritma BBS yang diimplementasikan dijelaskan sebagai berikut [1].

- Pilih dua bilangan p dan q, yang merupakan bilangan prima dan diharasiakan. Keduanya harus kongruen dengan 3 dalam modulus 4
- Hitung bilangan n = p * q
- Pilih bilangan bulat acak lain s, sebagai *seed* dari pembangkitan, yang memenuhi properti berikut.
    - $2 \leq s < n$
    - s dan n relatif prima
- Hitung X_0 = s^2 mod n
- Barisan bit acak dapat diambil secara terus menerus dari proses iterasi berikut.
    - Hitung X_i+1 = X_i ^ 2 mod n
    - Z_i = bit LSB (Least Significant Bit) dari X_i
- Barisan bit acak yang dihasilkan diambil dari Z_1, Z_2, Z_3, …

Implementasi algoritma ini dilakukan dalam suatu kelas Python yang menerima parameter p dan q, serta parameter opsional seed. Pada instantiasi dilakukan validasi terhadap properti p dan q. Jika memenuhi persyaratan, akan dilakukan instantiasi dan kalkulasi bilangan n. Dilakukan pula validasi seed untuk mengecek apakah properti seed valid.

Dalam implementasinya, diperlukan beberapa fungsi bantuan sehingga diimplementasikan fungsi bantuan yaitu fungsi coprime untuk menentukan apakah dua bilangan merupakan bilangan yang relatif prima, dan fungsi is_prime untuk menentukan apakah suatu bilangan merupakan bilangan prima.

Pembangkitan bilangan acak dilakukan dengan mengikuti algoritma yang telah dijelaskan. Disediakan tiga fungsi sebagai *post processing* dan digunakan sebagai berikut.

- gen_bits untuk membangkitkan bit sejumlah yang dispesifikkan pada parameter.
- gen_bytes untuk membangkitkan bytes sejumlah yang dispesifikkan pada parameter.
- randrange untuk membangkitkan bilangan bulat diantara batas yang dispesifikkan pada parameter.

D. Implementasi Multi-Secret Sharing

Diimplementasikan program utama yang dapat melakukan interaksi melalui *command line* sebagai *interface* dari pengguna dalam menentukan tipe program yang dilakukan dan data masukan yang dipakai. Seperti yang telah disebutkan sebelumnya, akan digunakan *multiprocessing* pada pemrosesan data. Sedangkan *asynchronous programming* digunakan pada pembacaan dan penulisan berkas.

Program pembangkitan bagian *secret sharing*, dimulai meminta masukan dari pengguna mengenai jumlah berkas yang akan digunakan sebagai data rahasia dan lokasi berkas-berkas tersebut beserta jumlah *shares* yang akan

dibangkitkan. Lalu dilakukan inisialisasi pool dari *multiprocessing* yang dapat dipakai kembali. Program lalu membaca berkas secara asinkron sehingga didapatkan data frame berkas suara dan parameter dari berkas. Program lalu mengorganisir data tersebut sedemikian sehingga semua data dan parameter sebagai satu array dengan data terurut diikuti dengan parameter terurut.

Program lalu melakukan pemrosesan pembangkitan bagian dengan *multiprocessing*. Algoritma yang dipakai dijelaskan sebagai berikut [1].

- Misalkan data adalah D, dengan *shares* yang dihasilkan adalah S1, S2, …, SN dan array bilangan acak yang membantu adalah R1, R2, …, R(N-1) dengan ukuran seperti ukuran D
- Skema (n,n) dapat dihasilkan dengan urutan:
  - S1 = R1
  - S2 = R1 ⊕ R2
  - …
  - S(N-1) = R(N-2) ⊕ R(N-1)
  - SN = R(N-1) ⊕ D
- Dihasilkan *shares* dengan skema ambang (N, N) yaitu S1, S2, …, SN

Pada implementasinya, *multiprocessing* dilakukan pada pembangkitan array bilangan acak dengan BBS. Hal ini dapat dilakukan karena proses pembangkitan bersifat independen satu sama lain.

Shares yang dihasilkan lalu di-*join* menggunakan separator yaitu b'\xff\xff\x00\x00\xff\xff' dengan alasan yang telah disebutkan pada batasan dan asumsi. Lalu digunakan parameter *dummy* untuk berkas bagian dari parameter berkas suara pertama. Selanjutnya, *shares* yang dihasilkan dan parameter *dummy* ditulis ke berkas-berkas sebanyak jumlah *shares* dengan *asynchronous programming.*

Program rekonstruksi data awal *secret sharing*, dimulai meminta masukan dari pengguna mengenai jumlah berkas *shares* yang akan digunakan dan lokasi berkas-berkas tersebut. Lalu dilakukan inisialisasi pool dari *multiprocessing* yang dapat dipakai kembali. Program lalu membaca berkas secara asinkron sehingga didapatkan data frame berkas suara dan parameter dari berkas. Program lalu hanya mengambil data berkas karena parameter asli berada pada data frame.

Program lalu mengorganisir dan memecah data menggunakan separator yang sama, sedemikian sehingga semua data dan parameter terpisah-pisah. Lalu dilakukan proses rekonstruksi dengan algoritma sebagai berikut [1].

- Misalkan *shares* yang dihasilkan adalah S1, S2, …, SN
- Dilakukan operasi XOR terhadap semua share untuk mendapatkan data awal:
  - S1 ⊕ S2 ⊕ … ⊕ SN = R1 ⊕ (R1 ⊕ R2) ⊕ … ⊕ (R(N-1) ⊕ D)
  - S1 ⊕ S2 ⊕ … ⊕ SN = (R1 ⊕ R1) ⊕ (R2 ⊕ R2) ⊕ … ⊕ (R(N-1) ⊕ R(N-1)) ⊕ D
  - S1 ⊕ S2 ⊕ … ⊕ SN = (0 ⊕ 0 ⊕ … ⊕ 0) ⊕ D
  - S1 ⊕ S2 ⊕ … ⊕ SN = 0 ⊕ D
  - S1 ⊕ S2 ⊕ … ⊕ SN = D

Untuk setiap data dan parameter dari *shares* dilakukan algoritma yang sama tersebut untuk mendapatkan data dan parameter awal. Data frame dan parameter awal yang didapatkan lalu diorganisir kembali sesuai berkas awal dan ditulis pada berkas hasil secara asinkron.

Selain itu, diimplementasikan proses yang tidak menggunakan *multiprocessing* dan *asynchronous programming* sebagai *benchmark*. Data kinerja dari proses pembangkitan *shares* dan rekonstruksi baik teroptimasi maupun *benchmark* dikalkulasi, disimpan, dan divisualisasikan dengan pustaka yang telah disebutkan pada batasan dan asumsi.

IV. Hasil Dan Pembahasan

Pada bagian ini, dilakukan pengujian terhadap implementasi beserta analisis terhadap hasil pengujian yang didapatkan.

A. Inisialisasi Pengujian

Pada tahap pengujian, disimpan beberapa berkas suara yang akan digunakan sebagai data rahasia. Berkas-berkas suara tersebut adalah sebagai berikut.

TABLE I. Berkas Suara Rahasia

Nama	Ukuran	Hash (SHA-256)
CantinaBand3.wav	130 KB	2E5F73001B324E413BDCF658FCA5485057C333F4198E51E7E86BB2E772CD0973
keyboard3s.wav	551 KB	B3726EAC5C9612EA20E245314812575BF9DF5FB6B8024B80C7CFE9033452BB2B
StarWars3.wav	130 KB	2BF6074771B8BF5C556DF66BD373825AFA395CD4F3355B18B236723BD0960A89
taunt4.wav	90 KB	2EA9369B8E7605F954266BB88C55A8FE4E1BBBB260BA89170E15E2B1A03FDE84

Verifikasi apakah data rahasia dan data hasil rekonstruksi identik, dilakukan dengan memeriksa nilai hash dari kedua berkas tersebut. Jika keduanya memiliki nilai hash yang sama, maka keduanya identik. Selain itu, kedua berkas audio dapat dibandingkan suaranya dengan cara dijalankan.

B. Pengujian dan Analisis

Setelah berkas data telah disiapkan dan program untuk validasi, maka proses pembangkitan bagian dan rekonstruksi dapat dilakukan. Terdapat tiga kasus uji yang akan dilakukan pada implementasi yang telah dibuat.

1. Kasus 1: Pembangkitan *shares* dan rekonstruksi dengan jumlah berkas kurang dari jumlah *shares*

   Kasus ini merupakan kasus yang menguji apakah program dapat melakukan *multi-secret sharing* terhadap berkas-berkas dengan jumlah kurang dari jumlah share yang akan dihasilkan. Pada kasus ini

digunakan berkas CantinaBand3.wav dan keyboard3s.wav sebagai data rahasia. Ditetapkan jumlah *shares* yang dihasilkan sejumlah empat *shares*.

Berikut merupakan hasil berkas suara rekonstruksi.

TABLE II. HASIL BERKAS SUARA REKONSTRUKSI KASUS 1

Nama	Ukuran	Hash (SHA-256)
combine1.wav	130 KB	2E5F73001B324E413BDCF658FCA5485057C333F4198E51E7E86BB2E772CD0973
combine2.wav	551 KB	B3726EAC5C9612EA20E245314812575BF9DF5FB6B8024B80C7CFE9033452BB2B

Pada data hash hasil berkas suara, didapatkan nilai hash yang sama dengan nilai hash dari berkas awal. Hal ini menandakan berkas hasil rekonstruksi identik dengan berkas suara awal.

Berikut merupakan hasil visualisasi dari kinerja program untuk pembangkitan *shares*.

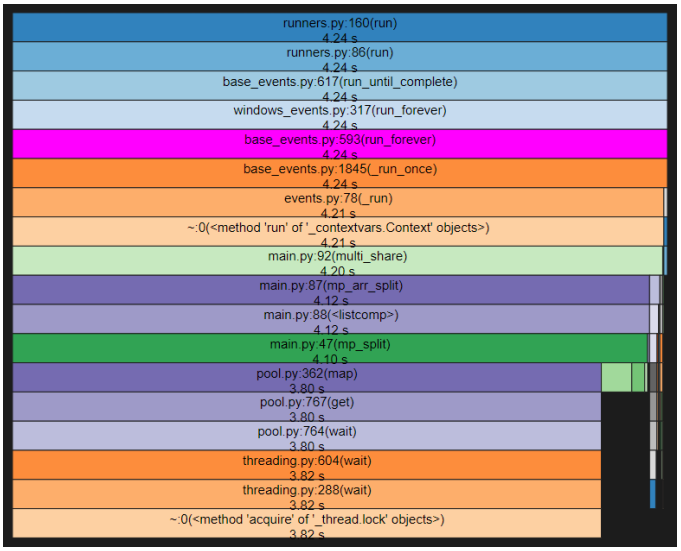

Fig. 4. Visualisasi kinerja program pembangkitan *shares* teroptimasi pada kasus satu

Berikut merupakan hasil visualisai dari kinerja program untuk rekonstruksi.

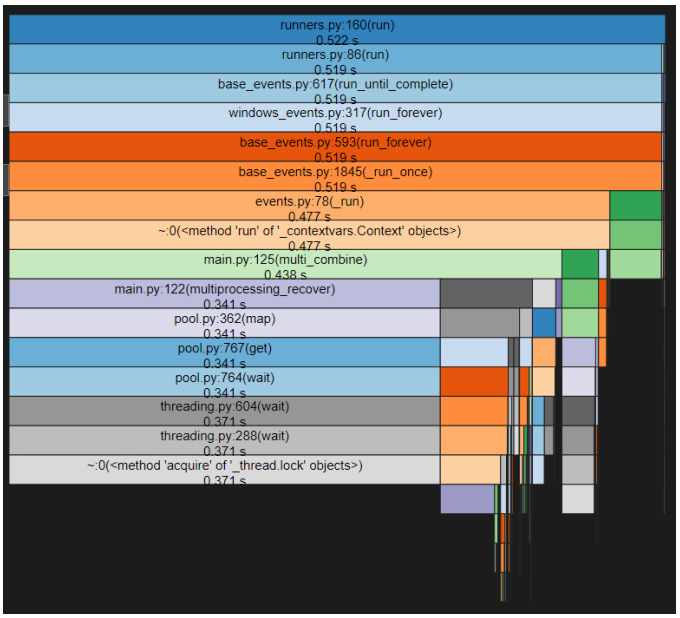

Fig. 5. Visualisasi kinerja program rekonstruksi teroptimasi pada kasus satu

Selain itu dilakukan perbandingan dengan program tanpa optimasi dan didapatkan visualisasi kinerja program sebagai berikut.

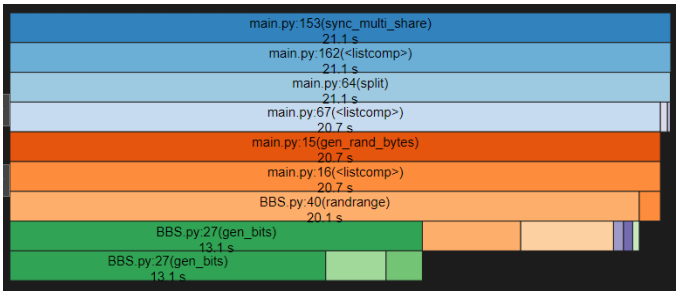

Fig. 6. Visualisasi kinerja program pembangkitan *shares* tanpa optimasi pada kasus satu

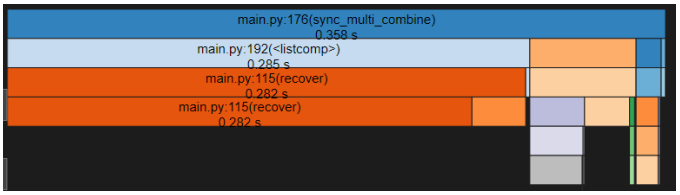

Fig. 7. Visualisasi kinerja program rekonstruksi tanpa optimasi pada kasus satu

Dapat terlihat bahwa kinerja pembangkitan *shares* dengan optimasi jauh lebih baik daripada program tanpa optimasi. Didapatkan waktu total untuk program teroptimasi selama yaitu 4.24 detik, sementara program tanpa optimasi memiliki waktu 21.1 detik. Hal ini menandakan optimasi cukup sukses dalam meningkatkan kinerja program. Proses penyumbang waktu terlama pada program tanpa optimasi adalah program BBS untuk membangkitkan bilangan acak, yaitu selama 20.1 detik.

Namun, untuk rekonstruksi, program dengan optimasi sedikit tertinggal dengan waktu 0.522 detik, sementara program tanpa optimasi hanya 0.358 detik. Dari visualisasi kinerja, didapatkan penyumbang waktu terlama pada program teroptimasi adalah untuk mendapatkan *thread lock,* selama 0.371 detik.

Dari hasil tersebut, didapatkan optimasi pada pembangkitan bilangan acak dengan *multiprocessing* sangat berguna dalam meningkatkan kinerja program. Sementara itu, adanya multiprocessing pada program rekonstruksi membawa *overhead* yang cukup besar pada kasus ini.

2. Kasus 2: Pembangkitan shares dan rekonstruksi dengan jumlah berkas lebih dari jumlah shares

   Kasus ini merupakan kasus yang menguji apakah program dapat melakukan *multi-secret sharing* terhadap berkas-berkas dengan jumlah lebih dari jumlah share yang akan dihasilkan. Pada kasus ini digunakan keempat berkas yang disiapkan yaitu, CantinaBand3.wav, keyboard3s.wav, StarWars3.wav, dan taunt4.wav, sebagai data rahasia. Ditetapkan jumlah *shares* yang dihasilkan sejumlah dua *shares*.

   Berikut merupakan hasil berkas suara rekonstruksi.

TABLE III. HASIL BERKAS SUARA REKONSTRUKSI KASUS 2

Nama	Ukuran	Hash (SHA-256)
combine1.wav	130 KB	2E5F73001B324E413BDCF658FCA5485057C333F4198E51E7E86BB2E772CD0973
combine2.wav	551 KB	B3726EAC5C9612EA20E245314812575BF9DF5FB6B8024B80C7CFE9033452BB2B
combine3.wav	130 KB	2BF6074771B8BF5C556DF66BD373825AFA395CD4F3355B18B236723BD0960A89
combine4.wav	90 KB	2EA9369B8E7605F954266BB88C55A8FE4E1BBBB260BA89170E15E2B1A03FDE84

Pada kasus dua, juga didapatkan data nilai hash hasil berkas suara sama dengan nilai hash dari berkas awal. Hal ini menandakan berkas hasil rekonstruksi identik dengan berkas suara awal.

Berikut merupakan hasil visualisasi dari kinerja program untuk pembangkitan *shares*.

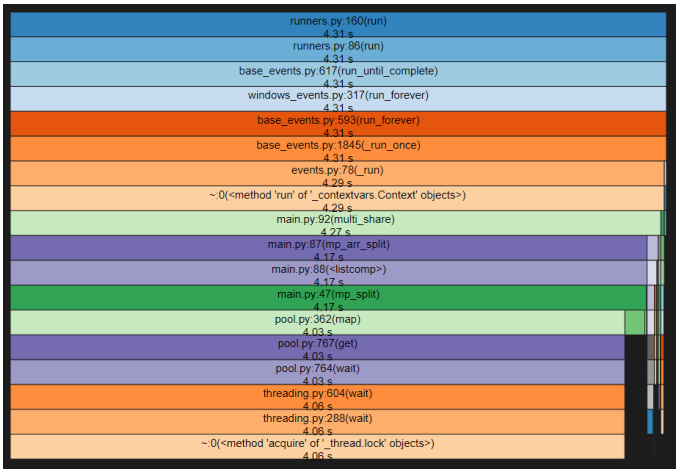

Fig. 8. Visualisasi kinerja program pembangkitan *shares* teroptimasi pada kasus dua

Berikut merupakan hasil visualisai dari kinerja program untuk rekonstruksi.

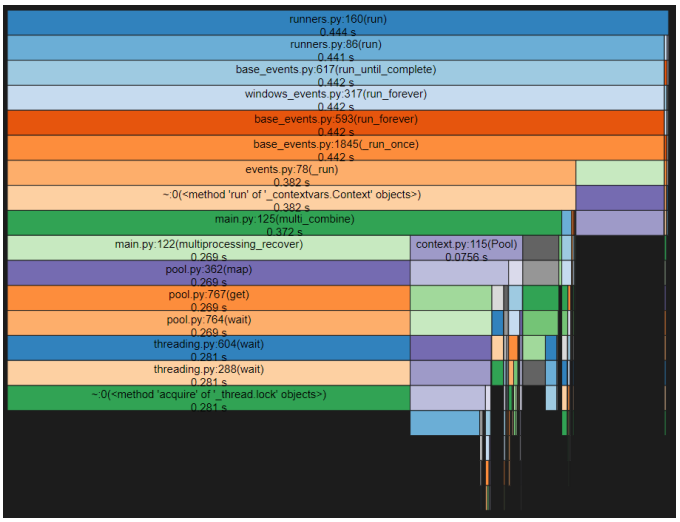

Fig. 9. Visualisasi kinerja program rekonstruksi teroptimasi pada kasus dua

Selain itu dilakukan perbandingan dengan program tanpa optimasi dan didapatkan visualisasi kinerja program sebagai berikut.

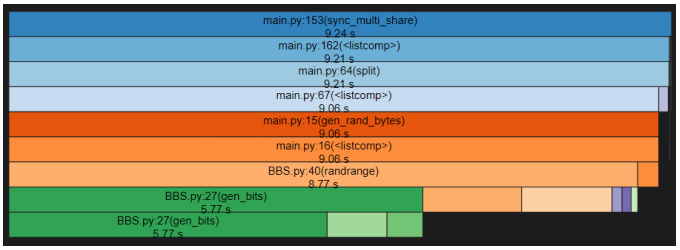

Fig. 10. Visualisasi kinerja program pembangkitan *shares* tanpa optimasi pada kasus dua

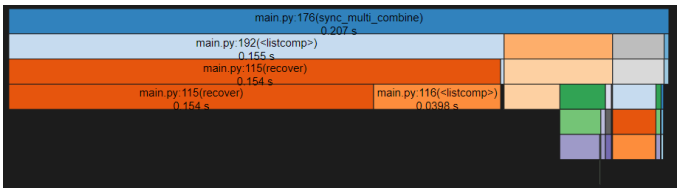

Fig. 11. Visualisasi kinerja program rekonstruksi tanpa optimasi pada kasus dua

Pada kasus dua, didapatkan hasil yang cukup mirip dengan kasus satu. Dapat terlihat bahwa kinerja pembangkitan *shares* dengan optimasi jauh lebih baik daripada program tanpa optimasi. Didapatkan waktu total untuk program teroptimasi selama yaitu 4.31 detik, sementara program tanpa optimasi memiliki waktu 9.24 detik. Walaupun perbedaan tidak semencolok pada kasus satu, hal ini tetap menandakan optimasi cukup sukses dalam meningkatkan kinerja program. Proses penyumbang waktu terlama pada program tanpa optimasi adalah program BBS untuk membangkitkan bilangan acak, yaitu selama 8.77 detik.

Seperti kasus satu, untuk rekonstruksi, program dengan optimasi sedikit tertinggal dengan waktu 0.444 detik, sementara program tanpa optimasi hanya 0.207 detik. Dari visualisasi kinerja, didapatkan penyumbang waktu terlama pada program teroptimasi adalah untuk mendapatkan *thread lock,* selama 0.281 detik, sama seperti kasus satu.

Dari hasil tersebut, didapatkan optimasi pada pembangkitan bilangan acak dengan *multiprocessing* sangat berguna dalam meningkatkan kinerja program. Sementara itu, adanya multiprocessing pada program rekonstruksi membawa *overhead* yang cukup besar pada kasus ini.

C. *Analisis Secara Keseluruhan*

Berdasarkan hasil pengujian yang telah dilakukan, dapat ditarik beberapa poin penting, yaitu sebagai berikut.

- Optimasi *multiprocessing* terbukti efektif meningkatkan kinerja program pembangkitan *shares*

  Dari kedua kasus yang dilakukan, didapatkan perbedaan kinerja cukup besar pada program teroptimasi dibanding tanpa optimasi. Hal ini terjadi bahkan pada kasus dengan data yang sedikit dengan jumlah *shares* terbanyak adalah empat.

- Optimasi *asynchronous programming* tidak terlalu terlihat manfaatnya, setidaknya pada kasus dengan ukuran berkas kecil dan sedikit.

  Penulis menduga, hal ini dikarenakan ukuran berkas kecil dan sedikit. Dari kedua kasus yang dilakukan, digunakan berkas dengan ukuran kecil dan sedikit, sesuai dengan penjelasan pada batasan dan asumsi. Hal ini mengakibatkan optimasi *asynchronous programming* tidak terlalu terlihat pengaruh peningkatan kinerjanya.

- Optimasi *multiprocessing* pada program rekonstruksi lebih banyak membawa *overhead* dibanding peningkatan kinerja, setidaknya pada jumlah *shares* yang sedikit dan ukuran kecil.

  Pada visualisasi kinerja program rekonstruksi, didapatkan *cost* untuk melakukan rekonstruksi sangatlah kecil. Hal ini mengakibatkan biaya *overhead* dari *multiprocessing*, yaitu *locking* (pada kasus ini adalah ketika melakukan update pada array hasil) dan inisialisasi *pool multiprocessing* lebih besar daripada biaya kinerja rekonstruksi. Pada kasus *shares* dengan jumlah banyak dan ukuran besar, penulis menduga optimasi *multiprocessing* akan lebih terlihat manfaatnya.

- Skema *Multi-Secret Sharing* yang digunakan terbukti aman dan efektif untuk menjaga dan membagi kerahasiaan.

  Dari pengujian yang dilakukan, didapatkan hasil hash dari data awal dan data hasil rekonstruksi bernilai sama. Hal ini menandakan isi kedua berkas sama. Selain itu, dilakukan pengujian dengan cara membandingkan suara kedua berkas tersebut, dan didapatkan bahwa suara yang dihasilkan identik. Selain itu, suara yang dihasilkan dari pembangkitan *shares* sangat berbeda dengan suara data awal dengan suara menyerupai *noise* acak. Ditambah dengan penggunaan algoritma pembangkit bilangan acak yang aman, yaitu Blum Blum Shub, dapat ditarik bahwa skema *multi-secret sharing* ini bersifat aman dan efektif. Untuk hasil berkas suara yang dihasilkan dari kasus uji dapat diakses pada *section* selanjutnya.

V. KESIMPULAN

Dari makalah ini, dapat disimpulkan bahwa *multi-secret sharing* untuk beberapa data rahasia, khususnya data berkas suara berformat wav, secara sekaligus terbukti dapat dilakukan dan aman. Selain itu, berdasarkan hasil dan analisis yang dilakukan, terbukti optimasi dengan *multiprocessing* dan *asynchronous programming* dapat mempercepat kinerja program. Dengan demikian, pembagian beberapa data rahasia berkas suara secara sekaligus kepada suatu kelompok *shareholder* dapat dilakukan dengan aman dan cepat.

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur yang sebesar-besarnya kepada rahmat Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga makalah berjudul "Perancangan dan Implementasi *Multiple Audio Files Secret Sharing* dengan Penggunaan Pembangkit Bilangan Acak Blum Blum Shub dan Optimasi *Multiprocessing* dan *Asynchronous Programming*" dapat diselesaikan dengan baik. Penulis mengucapkan terima kasih kepada dosen pengajar IF4020 Kriptografi, Dr. Rinaldi Munir, S.T, M.T. yang telah memberikan bimbingan dan ilmu terkait materi Kriptografi ini, khusunya pada materi *secret sharing* dan pembangkitan bilangan acak. Penulis juga mengucapkan terima kasih banyak kepada para penulis sumber referensi digunakan pada makalah ini yang telah memberikan ilmu yang dibutuhkan penulis untuk menyelesaikan makalah ini.

Pranala Berkas Hasil Pengujian

Berkas hasil pengujian yang dilakukan, meliputi berkas suara *shares* dan rekonstruksi, beserta data statistik kinerja program dapat diakses pada pranala berikut.

https://drive.google.com/drive/folders/1KtQFLxMKPLQqi2-dFdrg_sXmRKT_W94E?usp=sharing

Pranala Kode Program Implementasi

Kode program implementasi dapat diakses pada pranala berikut.

https://github.com/malikrafsan/MultiProcessing-Multi-Audio-Secret-Sharing

Pernyataan

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

Daftar Pustaka

[1] Munir, Rinaldi. 2021. Slide Kuliah Kriptografi (Bandung: Institut Teknologi Bandung)

[2] Chattopadhyay, Arup & Sadhu, Shayak & Nag, Amitava & Singh, Jyoti. (2017). An audio secret sharing using XOR operations with scalable shares. 2743-2746. 10.1109/TENCON.2017.8228328.

[3] Bisht, K., & Deshmukh, M. (2021). A novel approach for multilevel multi-secret Image sharing scheme. The Journal of Supercomputing, 77(10), 12157–12191. https://doi.org/10.1007/s11227-021-03747-y

[4] Buchanan OBE, P. B. (2019, April 19). Go be rAnd0m with Blum Blum Shub. Medium. Retrieved May 18, 2023, from https://medium.com/asecuritysite-when-bob-met-alice/doo-wop-diddi-diddi-blum-blum-shub-96eebf44868d

Bandung, 22 Mei 2023

Malik Akbar Hashemi Rafsanjani
13520105