ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IF184802

METODE PENGAUDITAN INTEGRITAS BERBASIS IDENTITAS DAN BERBAGI DATA DENGAN PENYEMBUNYIAN INFORMASI SENSITIF UNTUK PENYIMPANAN AWAN YANG AMAN

RIFQI ARDIA RAMADHAN
NRP 05111640000005

Dosen Pembimbing I
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing II
Tohari Ahmad, S.Kom., M.IT., Ph.D.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - IF184802

METODE PENGAUDITAN INTEGRITAS BERBASIS IDENTITAS DAN BERBAGI DATA DENGAN PENYEMBUNYIAN INFORMASI SENSITIF UNTUK PENYIMPANAN AWAN YANG AMAN

RIFQI ARDIA RAMADHAN
NRP 05111640000005

Dosen Pembimbing I
Bagus Jati Santoso, S.Kom., Ph.D.

Dosen Pembimbing II
Tohari Ahmad, S.Kom., M.IT., Ph.D.

DEPARTEMENT TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - IF184802

IDENTITY-BASED INTEGRITY AUDITING METHOD AND DATA SHARING WITH SENSITIVE INFORMATION HIDING FOR SECURE CLOUD STORAGE

RIFQI ARDIA RAMADHAN
NRP 05111640000005

Supervisor I
Bagus Jati Santoso, S.Kom., Ph.D.

Supervisor II
Tohari Ahmad, S.Kom., M.IT., Ph.D.

DEPARTEMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

METODE PENGAUDITAN INTEGRITAS BERBASIS IDENTITAS DAN BERBAGI DATA DENGAN PENYEMBUNYIAN INFORMASI SENSITIF UNTUK PENYIMPANAN AWAN YANG AMAN

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S1 Departemen Teknik Informatika Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember

Oleh :

RIFQI ARDIA RAMADHAN
NRP: 05111640000005

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Bagus Jati Santoso, S.Kom., Ph.D. ....
NIP: 198611252018031001 (Pembimbing 1)

Tohari Ahmad, S.Kom., M.IT., Ph.D. .....
NIP: 197505252003121002 (Pembimbing 2)

SURABAYA
Juni 2020

(Halaman ini sengaja dikosongkan)

METODE PENGAUDITAN INTEGRITAS BERBASIS IDENTITAS DAN BERBAGI DATA DENGAN PENYEMBUNYIAN INFORMASI SENSITIF UNTUK PENYIMPANAN AWAN YANG AMAN

Nama	:	**RIFQI ARDIA RAMADHAN**
NRP	:	**05111640000005**
Departemen	:	**Teknik Informatika FTEIC**
Pembimbing I	:	**Bagus Jati Santoso, S.Kom., Ph.D.**
Pembimbing II	:	**Tohari Ahmad, S.Kom., M.IT., Ph.D.**

Abstrak

Dengan adanya layanan penyimpanan awan, pengguna dapat menyimpan datanya secara remote di awan dan membagikannya dengan pengguna lain. Pengauditan integritas data secara remote diusulkan untuk menjamin integritas dari data yang disimpan di awan. Di dalam sistem penyimpanan awan yang umum, file yang disimpan di awan bisa jadi memliki informasi sensitif. Informasi sensitif ini tidak seharusnya terlihat oleh pengguna lain ketika file ini dibagikan. Mengenkripsi file secara keseluruhan bisa digunakan untuk menyembunyikan informasi sensitif, namun membuat file ini tidak bisa digunakan oleh pengguna lain. Untuk menyelesaikan permasalahan ini, digunakan metode pengauditan integritas berbasis identitas dengan menerapkan berbagi data dengan penyembunyian informasi sensitif. File akan disanitasi, lalu signature dimodifikasi sedemikian sehingga menjadi bentuk yang valid dan digunakan untuk pengauditan. Hasil yang diharapkan dari metode ini adalah file yang disimpan di penyimpanan awan dapat dibagikan dan digunakan tanpa dapat melihat informasi sensitif, sementara pengauditan integritas masih dapat dilakukan. Tugas akhir ini akan merancang dan

mengimplementasi cara pengauditan integritas berbasis identitas dan berbagi data dengan penyembunyian informasi sensitif berdasarkan penelitian yang sudah ada.
***Kata-Kunci**: integritas data, informasi sensitif, signature*

IDENTITY-BASED INTEGRITY AUDITING METHOD AND DATA SHARING WITH SENSITIVE INFORMATION HIDING FOR SECURE CLOUD STORAGE

Name	:	**RIFQI ARDIA RAMADHAN**
NRP	:	**05111640000005**
Department	:	**Informatics Engineering FTEIC**
Supervisor I	:	**Bagus Jati Santoso, S.Kom., Ph.D.**
Supervisor II	:	**Tohari Ahmad, S.Kom., M.IT., Ph.D.**

Abstract

With the existence of cloud storage service, user can store their data remotely in cloud and share it with other users. Data integrity auditing was proposed to ensure the integrity of the data saved in cloud. In a common cloud storage system such as Electronic Health Records, file stored in cloud can have sensitive information. This sensitive information should not be exposed to another user when the file is shared. Encrypting the file entirely can be used to redact any sensitive information in the file, but making the file cannot be used by another users. To solve this problem, an identity-based integrity auditing and data sharing with sensitive information hiding method is used. File will be sanitized, and then signature is modified in such a way that it will be valid and used for auditing. The result expected from this method is the file saved in the cloud can be shared without being able to see the sensitive information, while integrity auditing can be done efficiently. This undergraduate thesis will build and implement identity-based integrity auditing and data sharing with sensitive information hiding based on prior research.

***Keywords**: data integrity, sensitive information, signature*

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Metode Pengauditan Integritas Berbasis Identitas dan Berbagi Data dengan Penyembunyian Informasi Sensitif untuk Penyimpanan Awan yang Aman**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis untuk menelusuri lebih dalam terkait dengan topik keamanan informasi. Topik ini jarang untuk dipilih dan didalami, karena lebih menantang dibandingkan dengan topik lain. Menyelami topik ini merupakan tantangan tersendiri bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini, penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu mendukung saya di belakang.
3. Bapak Bagus Jati Santoso, S.Kom., Ph.D. selaku pembimbing I yang telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bapak Tohari Ahmad, S.Kom., M.IT., Ph.D. selaku pembimbing II yang juga telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
5. Teman-teman baik saya yang telah banyak mendukung dan menemani saya selama masa studi saya di Informatika.
6. Dr. Eng. Chastine Fatichah, S.Kom,, M.Kom., selaku

Kepala Departemen Informatika ITS pada masa pengerjaan Tugas Akhir, Bapak Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D, selaku koordinator TA dan segenap dosen Departemen Informatika yang telah memberikan ilmu dan pengalamannya.

7. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan yang berkelanjutan untuk ke depannya.

Surabaya, Juni 2020

Rifqi Ardia Ramadhan

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

DAFTAR PERSAMAAN

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir dan sistematika penulisan.

1.1 Latar Belakang

Dengan peningkatan pesat dari data, penyimpanan data dalam jumlah besar secara lokal menjadi sebuah beban untuk pengguna. Oleh karena itu, semakin banyak organisasi dan individu yang ingin menyimpan data di penyimpanan awan. Namun data yang disimpan di awan mungkin saja rusak atau hilang karena *bug* perangkat lunak, *hardware fault*, dan kesalahan manusia.

Di dalam skema pengauditan integritas data secara *remote*, pemilik data perlu menghasilkan *signature* untuk blok-blok data sebelum diunggah ke awan. *Signature* ini digunakan untuk membuktikan bahwa awan ini benar-benar memiliki blok-blok data yang dikirim dalam fase pengauditan integritas. Lalu pemilik data mengunggah blok-blok data ini bersamaan dengan *signature* yang sesuai ke awan. Data yang disimpan di awan umumnya dibagikan kepada banyak pengguna di banyak aplikasi penyimpanan awan. Berbagi data (*data sharing*) sebagai salah satu fitur yang paling umum dari penyimpanan awan, memungkinkan pengguna untuk berbagi data dengan pengguna lain. Namun, data yang dibagikan yang disimpan di awan ini bisa jadi memiliki informasi sensitif. Sedangkan dalam membagikan data, informasi sensitif tidak boleh terekspos ke pihak luar. Selain itu, integritas dari data ini juga perlu dijamin karena adanya kesalahan manusia dan kegagalan perangkat lunak maupun perangkat keras di awan. Oleh karena itu, sangat penting untuk bisa menyelesaikan pengauditan integritas data secara

remote dengan kondisi informasi sensitif dari data yang dibagikan terlindungi.

Metode yang berpotensi untuk menyelesaikan permasalahan ini adalah mengenkripsi keseluruhan data sebelum mengirimnya ke awan, dan menghasilkan *signature* yang dipakai untuk menverifikasi integritas dari *file* yang dienkripsi ini, lalu mengunggah *file* dan *signature*-nya ke awan. Metode ini bisa mengaplikasikan penyembunyian data sensitif dikarenakan hanya pemilik data yang bisa mendekripsi *file*. Namun hal ini akan membuat data yang dibagikan tidak bisa dipakai oleh pengguna lain. Mendistribusikan kunci dekripsi kepada pengguna terlihat seperti solusi yang tepat untuk permasalahan di atas. Namun, kita tidak mungkin mengaplikasikan metode tersebut ke skenario nyata karena beberapa hal. Pertama, distribusi kunci dekripsi memerlukan kanal yang aman, yang akan sulit dicapai di beberapa kasus tertentu. Terlebih lagi, sangat sulit bagi pengunggah untuk mengetahui pengguna mana yang benar-benar akan memanfaatkan data tersebut dan pengguna mana yang memiliki niat tidak baik. Akibatnya, situasinya tidak memungkinkan untuk menyembunyikan informasi sensitif dengan mengenkripsi keseluruhan *file*. Maka dari itu, bagaimana kita mewujudkan berbagi data dengan penyembunyian informasi sensitif dalam pengauditan integritas data secara *remote* ini sangat penting.

Tugas akhir ini ingin menyelesaikan permasalahan tersebut dengan melakukan pengauditan integritas berbasis identitas dan berbagi data dengan penyembunyian informasi sensitif untuk penyimpanan awan yang aman. Dalam solusi ini pengauditan dilakukan dengan identitas pengunggah, sedemikian sehingga walaupun data telah disanitasi, pengauditan integritas data bisa dilakukan. Tugas akhir ini merupakan implementasi berdasarkan jurnal yang sudah ada. [1]

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana mengimplementasi sanitasi informasi sensitif dari sebuah data?
2. Bagaimana mengimplementasi pengauditan integritas data berbasis identitas untuk data yang akan disanitasi dari informasi sensitif?
3. Bagaimana cara mengimplementasikan pengauditan integritas data berbasis identitas untuk data yang akan dibagikan di penyimpanan awan?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. Bahasa pemrograman yang digunakan untuk implementasi Tugas Akhir ini adalah Python 3.x.
2. Implementasi dilakukan dalam satu sistem, yang berarti simulasi dilakukan dalam *scope* tertutup.
3. Implementasi *pairing-based cryptography* diimplementasi dengan *library PBC Library*.
4. Format *file* data yang disanitasi adalah *file* PDF.
5. Alat bantu yang digunakan untuk melakukan sanitasi *file* PDF tersebut adalah *pdf-redactor* yang berbasis Python.
6. Antarmuka diimplementasi dengan *flask*, kerangka kerja web mikro yang berbasis Python.
7. Basis data yang digunakan dalam implementasi ini adalah MongoDB.

1.4 Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

1. Membuat sistem sanitasi informasi sensitif dari sebuah data.
2. Membuat perangkat pengauditan integritas data berbasis identitas berbasis identitas untuk data yang akan disanitasi dari informasi sensitif.
3. Membuat perangkat pengauditan integritas data berbasis identitas untuk data yang dibagikan di penyimpanan awan.

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini adalah memungkinkan adanya pengecekan integritas data tanpa perlu mengenkripsi seluruh bagian data. Data kemudian dapat dibagikan di awan dengan aman dan dapat diaudit sewaktu-waktu.

1.6 Metodologi

Metodologi yang digunakan dalam pembuatan Tugas Akhir ini adalah sebagai berikut.

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan tugas akhir ini terdiri dari hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah pada tugas akhir, tujuan dari pembuatan tugas akhir dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan referensi mengenai berbagai perangkat dan metode yang digunakan dalam mengimplementasi Tugas Akhir ini, seperti Python, *PBC Library*, integritas data, *Flask*, dan sebagainya, untuk mendukung dan memastikan setiap tahap pembuatan tugas akhir sesuai dengan prosedur yang berlaku serta dapat diimplementasikan. Sumber informasi dan referensi bisa didapatkan melalui buku, jurnal, dan internet.

1.6.3 Analisis dan Desain Perangkat Lunak

Pada tahap ini dilakukan analisis dan perancangan terhadap arsitektur tugas akhir yang akan dibuat. Tahap ini merupakan tahap yang paling penting dimana segala bentuk implementasi dapat berjalan dengan baik ketika arsitektur sistem juga baik.

1.6.4 Implementasi Perangkat Lunak

Pada tahap ini dilakukan implementasi atau realisasi dari analisis dan perancangan arsitektur sistem yang sudah dibuat sebelumnya, sehingga menjadi infrastruktur yang sesuai dengan apa yang direncanakan.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian untuk mengukur performa dari sistem penyimpanan konfigurasi perangkat jaringan menggunakan arsitektur sistem yang telah dibuat. Beberapa performa yang diukur antara lain, kecepatan sistem dan ketepatan dalam melakukan pengauditan integritas data. Setelah ujicoba dilakukan, maka dilaksanakan evaluasi terhadap kinerja sistem yang telah diimplementasi dengan tujuan dikembangkan untuk iterasi selanjutnya.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku tugas akhir yang berisi dokumentasi yang mencakup teori, konsep, implementasi dan hasil pengerjaan tugas akhir.

1.7 Sistematika Penulisan

Sistematika penulisan laporan tugas akhir secara garis besar adalah sebagai berikut :

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan metode, algoritma, pustaka dan alat yang digunakan dalam pembuatan tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai kriptografi, *pairing-based cryptography*, Python, Flask, integritas data, *private key*, MongoDB, dan *PBC Library*, dan pdf-redactor.
3. Bab III. Desain dan Perancangan
   Bab ini berisi mengenai analisis dan perancangan arsitektur sistem yang akan diimplementasikan dalam pembuatan tugas akhir.
4. Bab IV. Implementasi
   Bab ini berisi mengenai implementasi dari arsitektur sistem yang dibuat sebelumnya. Penjelasan berupa kode program dan pengaturan yang digunakan untuk implementasi arsitektur sistem.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisi tentang tahapan ujicoba terhadap performa arsitektur sistem dan evaluasi terhadap sistem yang dibuat.

6. Bab VI. Penutup
   Bab ini merupakan bab terakhir yang memaparkan kesimpulan dari hasil pengujian dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran yang ditujukan bagi pembaca yang berminat untuk melakukan pengembangan terhadap tugas akhir ini.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

(Halaman ini sengaja dikosongkan)

BAB II

TINJAUAN PUSTAKA

2.1 Kriptografi

Kriptografi adalah ilmu melakukan enkripsi dan dekripsi dari data. Didasarkan dari matematika yang kompleks, kriptografi menyediakan beberapa layanan keamanan informasi yang penting seperti autentikasi, kerahasiaan, integritas, dan *non-repudiation*. Protokol dan aplikasi kriptografi membuat kriptografi ramah pengguna dan memungkinkan pengguna untuk mengamankan data tanpa perlu melakukan perhitungan matematika kompleks sendiri. Kriptografi modern bergantung pada kunci kriptografi, umumnya berupa teks pendek, untuk melakukan enkode dan dekode pesan bersamaan dengan algoritma kriptografi. Kriptografi *Symmetric* dan *Assymmetric key* memberikan kerahasiaan data. *Digital signatures*, salah satu produk dari *public key cryptography*, memungkinkan verifikasi dari autentikasi, integritas, dan *non-repudiation*. [2]

Di dalam konsep data dan telekomunikasi, kriptografi penting ketika berkomunikasi melalui perantara apapun yang tidak dipercaya. Perantara ini mencakup hampir seluruh jaringan, termasuk internet. Terdapat lima fungsi utama dari kriptografi:

- *Privacy/confidentiality* (Privasi/kerahasiaan): Memastikan tidak ada yang dapat membaca pesan yang dikirim kecuali penerima yang dimaksud.
- *Authentication* (Autentikasi): Proses membuktikan identitas seseorang.
- *Integrity* (Integritas): Memastikan kepada penerima bahwa pesan yang diterima tidak diubah dengan cara apapun dari aslinya.
- *Non-repudiation*: Mekanisme untuk membuktikan bahwa pengirim aslinya yang memang mengirimkan pesannya.
- *Key exchange*: Metode di mana kunci kripto dibagikan antara pengirim dan penerima.

Terdapat beberapa fungsi lain yang bisa didukung oleh kriptografi dan istilah lain yang mungkin muncul [3]:

- *Forward Secrecy* (Kerahasiaan yang berkelanjutan)
- *Perfect Security* (Keamanan yang sempurna)
- *Deniable Authentication* (Autentikasi yang dapat ditolak)

2.2 *Pairing-based Cryptography*

Pairing-based Cryptography adalah penggunaan *pairing* dalam membangun atau menganalisis sistem kriptografi.

Pairing merupakan komputasi matematis yang memiliki berbagai sifat yang berguna dalam kriptografi. *Pairing* yang saat ini lumrah digunakan adalah *bilinear pairing*. Dalam sebuah buku, Menezes mendeskripsikan cara kerja *bilinear pairing* memiliki cara kerja sebagai berikut [4]:

- Misalkan n adalah sebuah bilangan prima, $G_1 = \langle P \rangle$ adalah sebuah *additively-written group* berorde n dengan identitas ∞, dan G_T adalah sebuah *multiplicatively-writted group* berorde n dengan identitas 1.
- *Bilinear pairing* dalam (G_1, G_T) adalah sebuah *map* dimana:

$$\hat{e} \colon G_1 \times G_1 \to G_T \tag{II.1}$$

  yang memenuhi syarat berikut:
  - *(bilinearity)* Untuk semua $R, S, T \in G_1, \hat{e}(R + S, T) = \hat{e}(R, T)\hat{e}(S, T)$ dan $\hat{e}(R, S + T) = \hat{e}(R, S)\hat{e}(R, T)$.
  - *(non-degeneracy)* $\hat{e}(P, P) \neq 1$.
  - *(computability)* $\hat{e}$ dapat secara efisien dikomputasi.
- *Bilinear pairing* memiliki beberapa properti:
  - $\hat{e}(S, \infty) = 1$ dan $\hat{e}(\infty, S) = 1$.
  - $\hat{e}(S, -T) = \hat{e}(-S, T) = \hat{e}(S, T)^{-1}$

  - $\hat{e}(aS, bT) = \hat{e}(S, T)^{ab}$ untuk semua $a, b \in \mathbb{Z}$
  - $\hat{e}(S, T) = \hat{e}(T, S)$
  - Jika $\hat{e}(S, R) = 1$ untuk semua $R \in G_1$, maka $S = \infty$
- Keamanan dari banyak *pairing* bergantung pada kekuatan dari *bilinear Diffie-Hellman problem*.

2.3 *PBC Library*

PBC (Pairing-based Cryptography) Library adalah pustaka C gratis yang dirilis dengan lisensi *GNU Lesser GPL* yang dibangun di atas *GMP library* (*GNU Multiple Precision Arithmetic Library* [5]) yang menjalankan operasi matematis dari *pairing-based cryptosystems*. Pustaka ini dibuat oleh Ben Lynn sebagai implementasi tesisnya tentang *pairing* dalam kriptosistem. [6] *PBC Library* didesain sebagai implementasi mendasar *pairing-based cryptosystems*, yang mana kecepatan dan praktis adalah tujuan yang penting. *PBC Library* menerapkan beragam modul seperti pembuatan kurva eliptikal, operasi kurva eliptikal, dan komputasi *pairing*. Ditulis dalam bahasa C, kecepatan pustaka ini terbilang cepat. *PBC Library* dalam Tugas Akhir ini digunakan sebagai alat bantu dalam mengimplementasi pembuatan *signature* dan verifikasinya. Selain itu terdapat *wrapper* Python yang sangat membantu untuk membangun implementasi ini dengan lingkungan yang sama.

2.4 Integritas data

Dalam sebuah publikasi oleh NIST, dijelaskan bahwa integritas adalah istilah untuk menjaga dari modifikasi atau penghancuran informasi yang salah dan memastikan informasi yang mutlak dan asli. Integritas adalah salah satu dari tiga konsep *confidentiality, integrity, availability*. Dari istilah di atas, integritas data adalah sebuah sifat keaslian data yang

muncul seketika data itu dibuat. Integritas data mencakup data di dalam penyimpanan, ketika diproses, dan ketika dikirim. [7]

2.5 *Private key*

Dalam sebuah publikasi oleh NIST, dijelaskan bahwa *private key* adalah *key* ”kriptografi yang digunakan dengan *asymmetric-key* dalam algoritma kriptografi yang tidak dipublikasikan dan secara unik diasosiasikan dengan entitas yang berhak menggunakannya.” Dalam kriptosistem *asymmetric-key*, *private key* sering dikaitkan dengan *public-key*. [8]

Dalam Tugas Akhir ini *private key* digunakan untuk membuat *signature* yang unik dan tidak terkait dengan *file* yang bersangkutan.

2.6 Python

Python adalah bahasa pemrograman *interpreter*, tingkat tinggi, dan serba-guna. Python mengutamakan keterbacaan kode dan ditujukan agar programmer bisa menulis kode yang jelas untuk proyek skala kecil maupun besar. Python memiliki kemampuan ekstensibilitas yang besar untuk pemrograman *back-end* dengan abstraksi yang tinggi. Dalam sebuah wawancara, dijelaskan bahwa Python menggabungkan modul, *exception*, penulisan secara dinamis, tipe data dinamis yang sangat tinggi, dan kelas. Python memiliki antarmuka ke banyak *system call* dan pustaka di berbagai sistem dan dapat diperluas ke bahasa pemrograman C atau C++. Python dapat berjalan pada berbagai sistem operasi seperti Unix, Linux, Mac Os dan Windows. [9]

Python adalah bahasa pemrograman tingkat tinggi yang dapat diterapkan pada berbagai masalah. Bahasa ini dilengkapi pustaka

yang besar untuk melakukan pemrosesan *string*, protokol internet, rekayasa perangkat lunak, dan antarmuka sistem operasi [10].

Dilihat dari kelebihannya, bahasa pemrograman Python dapat digunakan dalam pengembangan aplikasi yang kompleks. Pada tugas akhir ini, bahasa pemrograman Python dimanfaatkan untuk pembuatan *node* untuk tiap-tiap langkah pengauditan integritas berbasis identitas. Selain itu, Python juga digunakan sebagai antarmuka dengan pustaka *PBC Library*.

2.7 Flask

Flask adalah kerangka kerja aplikasi web yang ringan. Flask didesain agar bisa dibuat dengan cepat dan mudah, dengan kemampuan pengembangan menjadi aplikasi yang lebih kompleks. Di dalam dokumentasinya, pada awalnya Flask dibuat sebagai *wrapper* dari Werkzeug dan Jinja, dan sekarang menjadi salah satu kerangka kerja aplikasi web yang paling populer.

Flask didesain tidak memiliki dependensi dan tata letak kerangka aplikasi. Dengan demikian pengembang memiliki kebebasan untuk mengatur kerangka aplikasinya sendiri serta menambahkan modul yang diperlukan sesuai kebutuhan. Flask memiliki berbagai ekstensi yang dikembangkan oleh komunitas sehingga dapat menambahkan berbagai fungsi dengan mudah[11].

Flask memiliki kelebihan yaitu sangat ringan dan sangat sederhana dalam proses pengembangan. Sehingga Flask sangat cocok digunakan untuk pembuatan *HTTP API* dan antarmuka dinamis yang sederhana. Pada Tugas Akhir ini, Flask akan digunakan untuk pembuatan *HTTP API* dan antarmuka sederhana untuk pengguna.

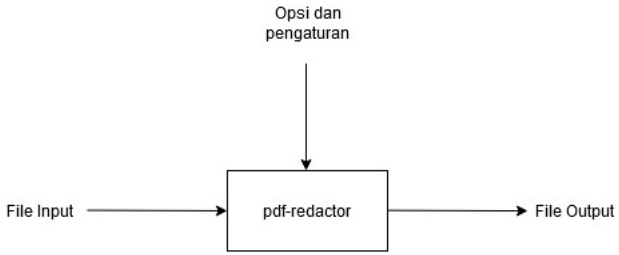
2.8 MongoDB

MongoDB adalah basis data serbaguna, berbasis dokumen, dan terdistribusi, dibangun untuk pengembang aplikasi modern dan era awan. MongoDB berlisensi SSPL, yang mana singkatnya jika fungsi utama MongoDB digunakan sebagai servis komersil, pengguna harus memilih antara mempublikasikan kodenya atau membayar biaya lisensi. Tapi jika MongoDB tidak digunakan sebagai servis utamanya, maka bisa digunakan secara gratis. [12] MongoDB menerapkan tiga prinsip desain utama dalam pengembangannya, yakni:

1. Model data *document*
2. Desain sistem terdistribusi dalam strukturnya
3. MongoDB dapat dijalankan dalam lingkungan manapun

Dalam Tugas Akhir ini, MongoDB digunakan untuk menyimpan blok data yang diunggah oleh pengguna beserta *signature*-nya dalam awan.

2.9 pdf-redactor



Gambar 2.1: Diagram Alur pdf-redactor

pdf-redactor adalah sebuah modul Python yang digunakan untuk menghapus data PDF di tingkat teks. pdf-redactor dibuat dengan pdfrw untuk memproses dan menulis data PDF. [13]

pdf-redactor berguna untuk menghapus atau mengganti teks di dalam PDF, yang dalam tugas akhir ini akan digunakan untuk menghapus informasi sensitif dari dokumen yang disebutkan. Untuk mempermudah dan mengamankan penggunaan, digunakan *fork* dari pengguna bernama dkloving, yang telah mengimplementasikan fungsi yang berguna untuk memanggil pdf-redactor tanpa *subprocess*. [14] Gambar 2.1 menunjukkan metode pdf-redactor memproses *file* PDF. *File* masukan didefinisikan dalam opsi atau di-*pipe* melalui stdout, lalu beberapa opsi didefinisikan juga untuk membuat pdf-redactor tahu apa yang harus dilakukan terhadap PDF yang dimasukkan, kemudian pdf-redactor menggunakan pdfrw untuk menerapkan opsi yang didefinisikan.

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

3.1 Deskripsi Umum Sistem

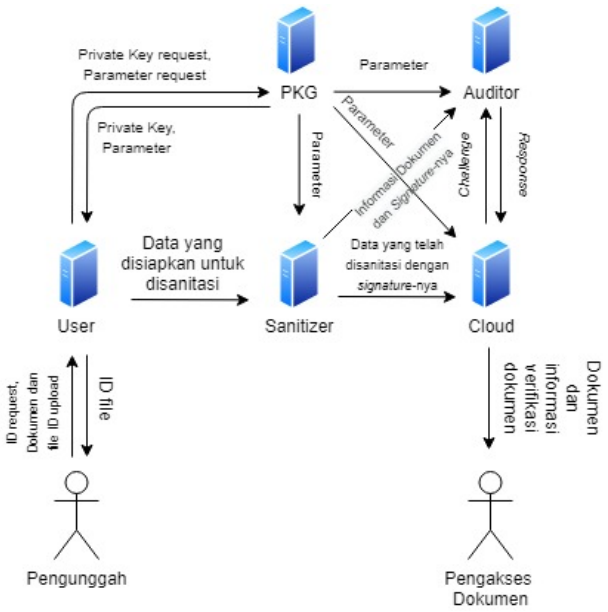
Sistem yang akan dibuat dalam tugas akhir ini dapat melakukan fungsi pembuatan kunci utama, melakukan pembuatan dan verifikasi *signature*, sanitasi informasi sensitif yang ingin dibersihkan dari dokumen aslinya, dan penyimpanan awan sederhana yang digunakan untuk berbagi data. Oleh karena itu dalam tugas akhir ini sistem dibagi dalam lima subsistem: *Private Key Generator* (PKG), *User*, *sanitizer*, penyimpanan awan, dan pengaudit (Auditor). Tujuan akhir dari pembuatan sistem ini adalah agar data dapat dibagikan dan diverifikasi tanpa perlu mempertahankan seluruh informasi sensitif yang terdapat pada data yang dibagikan. Audit dapat dilakukan sewaktu-waktu oleh pihak ketiga. Sistem dibangun dengan bahasa pemrograman Python versi 3.

Sistem berkomunikasi satu sama lain dengan API yang disediakan untuk komunikasi internal sistem menggunakan Flask. Komunikasi yang pasti terjadi adalah antara PKG dengan subsistem lain untuk meminta parameter yang digunakan untuk melakukan verifikasi *signature*. Sedangkan komunikasi lain akan dijelaskan dalam poin selanjutnya.

File yang didukung untuk dilakukan sanitasi adalah PDF, *file* dokumen formal yang umum digunakan. Karena spesifikasi *file* PDF yang terbilang tidak biasa, digunakan sebuah pustaka yang berlisensi Public Domain bernama pdf-redactor. Pustaka ini berguna untuk mengganti kata-kata dalam sebuah dokumen. Perubahan yang diterapkan dalam sistem ini adalah dari informasi sensitif menjadi ” ” atau *whitespace*, sehingga informasi sensitif tidak dapat dibaca.

Sistem menggunakan basis data untuk penyimpanan awan untuk berbagi data. Basis data yang digunakan adalah basis data MongoDB.

3.2 Alur Kerja Sistem



Gambar 3.1: Diagram Alur Komunikasi Sistem

Tabel 3.1: Daftar Notasi

Notasi	Arti Notasi
p	Bilangan prima besar
G_1, G_2	*Multiplicative cyclic group* dengan orde p
g	*Generator* dari group G_1
e	*Bilinear pairing map* $e\colon G_1 \times G_1 \rightarrow G_2$
Z_p^*	*Prime field* dengan elemen-elemen tidak nol
H	Fungsi *hash* kriptografik $H\colon \{0,1\}^* \rightarrow G_1$
x	Sebuah elemen di Z_p^*
$u', \mu_1, \mu_2, \ldots, \mu_l,\ u,\ g_2$	Elemen di dalam G_1
g_1	Nilai publik
n	Jumlah blok data pada *file* F
$F = \{m_1, m_2, \ldots, m_n\}$	*File* asli F
$F' = \{m_1', m_2', \ldots, m_n'\}$	*File* yang tersanitasi F'yang disimpan di awan.
ID	Identitas *user*
K	Himpunan yang berisikan teks yang mengandung informasi sensitif.
msk	*Master secret key*
sk_{ID}	*Private key* dari *user* ID
$\Phi = \{\sigma_i\}_{1 \leq i \leq n}$	Himpunan *signature* dari *file* asli F
$\Phi' = \{\sigma_i'\}_{1 \leq i \leq n}$	Himpunan *signature* dari *file* yang tersanitasi F'

Tujuan akhir dari implementasi pengauditan integritas data berbasis identitas ini adalah agar data dapat dibagikan dan diverifikasi tanpa perlu mempertahankan seluruh informasi sensitif yang terdapat pada data yang dibagikan. Oleh karena itu, audit dapat dilakukan sewaktu-waktu tanpa perlu khawatir bocornya informasi sensitif ke pihak ketiga.

Tugas Akhir ini akan mengimplementasikan pengauditan berbasis identitas yang dibagi menjadi beberapa tahap pengerjaan.

Gambar 3.1 menunjukkan alur komunikasi sistem. PKG berkomunikasi dengan seluruh sistem untuk memberikan parameter-parameter yang dibutuhkan untuk komputasi.

User berkomunikasi kepada PKG untuk meminta *private key* dan *Sanitizer* untuk mengirimkan blok data beserta *signature*-nya. *User* juga berkomunikasi dengan pengguna yang ingin mengungggah dokumen PDF sebagai antarmuka terhadap sistem.

Sanitizer berkomunikasi dengan Awan untuk mengirimkan blok data yang telah tersanitasi dan *signature* yang telah ditransformasi ke bentuk yang sesuai dan dengan Auditor untuk mengirimkan informasi terkait *file* dan *signature* yang berguna untuk kepentingan verifikasi kebenaran.

Awan berkomunikasi dengan Auditor untuk menjawab permintaan bukti yang diminta Auditor dan kepada pengguna yang ingin mengunduh *file* yang berada di Awan.

Auditor berkomunikasi hanya kepada Awan untuk meminta bukti verifikasi dan mengembalikan hasil verifikasinya ke Awan untuk ditunjukkan kepada pengguna.

Tahapan yang dilakukan dari sistem ini adalah sebagai berikut.

1. Persiapan

   Private Key Generator adalah entitas yang bertanggung jawab untuk membuat parameter sistem dan *private key* untuk user ID berdasarkan identitasnya ID.

(a) *Private Key Generator* (PKG) memilih dua multiplicative cyclic group G_1 dan G_2 dengan orde prima p, generator g dari G_1, sebuah map bilinear $e: G_1 \times G_1 \rightarrow G_2$.
(b) PKG memilih secara acak sebuah elemen $x \in Z_p^*$, elemen $\mu', \mu_1, \mu_2, \ldots, \mu_l, u, g_2 \in G_1$ dimana l adalah panjang bit identitas *user*, dan sebuah fungsi hash kriptografi $H: 0,1^* \rightarrow G_1$
(c) PKG lalu menghitung *public value* $g_1 = g^x$ dan sebuah secret key master $msk = g_2$ PKG lalu memberitahukan parameter sistem $pp = (G_1, G_2, p, e, g, \mu', \mu_1, \mu_2, \ldots \mu_l, u, g_1, g_2, H, f)$ dan menyimpan secret key master msk.

2. Ekstraksi
Proses ini bertujuan mengekstraksi *private key* dari ID

(a) Setelah menerima identitas *user* $ID = (ID_1, ID_2, ID_3, \ldots, ID_l) \in \{0,1\}^l$, PKG secara acak memilih sebuah nilai $r_{ID} \in Z_p^*$ dan menghitung

$$sk_{ID} = \left(sk'_{ID}, sk''_{ID}\right) = \left({g_2}^x \cdot \left(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j} \right), g^{r_{ID}} \right) \quad \text{(III.1)}$$

sebagai *private key* untuk *user* ID. Kemudian PKG mengirimkan *private key* dan r_{ID} tersebut ke *user* ID.
(b) *User* ID lalu menverifikasi kebenaran dari *private key* sk_{ID} yang telah diterima dengan mengecek persamaan

berikut pada persamaan III.2 berikut:

$$e(sk'_{ID}, g) = e(g_1, g_2) \cdot e(\mu' \prod_{j=1}^{l} \cdot \mu_j^{ID_j}, sk''_{ID}) \quad \text{(III.2)}$$

(c) Jika persamaan III.2 di atas tidak benar, maka *user* ID akan menolak sk_{ID} dan r_{ID}; selain itu terima.

3. Pembuatan *signature*
Proses ini *signature* dibuat oleh *user* untuk verifikasi data ke *sanitizer*.

(a) Setiap blok data $m_i \in Z_p^* \, (i \in [1, n])$ dari *file* asli F, *user* ID mengkalkulasi *signature* σ_i pada blok data m_i dengan kalkulasi III.3berikut.

$$\sigma_i = {g_2}^x \left(\mu' \prod_{j=1}^{l} {\mu_j}^{ID_j} \right)^{r_{ID}} \quad (H(name||i) \cdot u^{m_i})^r \quad \text{(III.3)}$$

Lalu buat $\Phi = \{\sigma_i\}_{1 \leq i \leq n}$ menjadi sebuah himpunan *signature* untuk *file* asli F.

(b) *User* ID mendefinisikan $\tau = name \,||g^{r_{ID}}|| \, g^r$.

(c) *User* ID mengkalkulasi nilai transformasi $\beta = u^r$ yang akan digunakan untuk menverifikasi kalkulasi *signature* pada tahap 4. Lalu *user* ID mengirim $\{F, \Phi, \tau, K, ID, \beta, r_{ID}, r\}$ ke *sanitizer*, lalu menghapus pesan tersebut dari penyimpanan lokal.

4. Sanitasi
Dalam proses ini, semua informasi sensitif dibersihkan dari *file*. *Sanitizer* adalah entitas yang berfungsi untuk mensanitasi blok data yang berisikan informasi sensitif, baik pribadi maupun organisasi. *Sanitizer* mengubah

signature dari blok data ini menjadi nilai yang valid, dan mengunggah *file* yang telah tersanitasi dan *signature*-nya ke awan. [1]

(a) Sanitizer memproses τ untuk mendapatkan nama identitas *file* name dan nilai verifikasi $g^{r_{ID}}$ dan g^r, dan melanjutkan proses berikutnya.
(b) Sanitizer kemudian menverifikasi kebenaran signature $\sigma_i \quad (i \in [1, n])$ dengan cara yang tertera pada persamaan III.4.

$$e(\sigma_i, g) = e(g_1, g_2) \cdot e(\mu\prime \prod_{j=1}^{l} {\mu_j}^{ID_j}, g^{r_{ID}}) \cdot e(H(name||i) \cdot u^{m_i}, g^r) \quad \text{(III.4)}$$

Jika persamaan III.4 di atas tidak valid, maka *sanitizer* akan menganggap *signature* nya tidak valid. Jika tidak, lanjut ke langkah berikutnya.

(c) *Sanitizer* menverifikasi kebenaran dari nilai transformasi β dengan mengecek apakah $e\,(u,\ g^r) \ = \ e\,(\beta, g)$ benar atau tidak. Jika benar, *sanitizer* akan mensanitasi blok data berdasarkan K. Informasi sensitif ini didefinisikan pada himpunan K. Untuk menyamakan formatnya, *sanitizer* menggunakan *wildcard* untuk menggantikan barisan data ini. Misalnya di dalam EHR, Rifqi adalah nama *user*. *Sanitizer* mengganti barisan data yang disebutkan dalam K dengan, misalkan "$* * * * *$", atau cukup dengan merubahnya menjadi *whitespace* atau " ". Lalu sanitizer mentransformasi signature dari *file* F yang telah disanitasi ke dalam bentuk yang valid untuk *file* yang tersanitasi F' dengan cara

berikut:

$$\sigma'_i = g_2^x \cdot (\mu' \cdot \prod_{j=1}^{l} \mu_j^{ID_j})^{r_I D} \cdot (H(name\|i) \cdot u^{m'_i})^r \tag{III.5}$$

Definisikan $\Phi' = \sigma'_{i1 \le i \le n}$ menjadi himpunan *signature* dari *file* yang telah tersanitasi.

(d) Sanitizer kemudian mengirimkan $\{F', \Phi', name\}$ ke awan, dan mengirimkan *file* tag τ ke Auditor Pihak Ketiga. Lalu *file* tersebut dihapus dari penyimpanan lokal.

5. Pembuatan bukti
   Bukti dibuat saat Auditor meminta Awan untuk memberikan verifikasinya.
   (a) Auditor menverifikasi *file* tag τ. Auditor tidak akan mengaudit apabila τ tidak valid; selain itu Auditor mem-parsing τ untuk mendapatkan identitas *file* $name$ dan nilai verifikasi $g^{r_{ID}}$ dan g^r lalu membuat *challenge* audit $chal$ dengan cara berikut:
      i. Secara acak mendefinisikan himpunan I dengan elemen sebanyak c di mana $I \subseteq [1, n]$.
      ii. Membuat nilai acak $v_i \in Z_p^*$ untuk tiap $i \in I$
      iii. Mengirimkan challenge audit $chal = \{i, v_i\}_{i \in I}$ ke awan.
   (b) Setelah menerima $chal$ dari Auditor, awan membuat bukti kepemilikan data dengan cara sebagai berikut:
      i. Melakukan komputasi kombinasi linear dari blok data $\lambda = \sum_{i \in I} m'_i v_i$.
      ii. Mengkalkulasi *signature* agregasi $\prod_{i \in I} \sigma_i'^{v_i}$.
      iii. Mengembalikan bukti audit $P = \{\lambda, \sigma\}$ kepada Auditor.
6. Verifikasi bukti

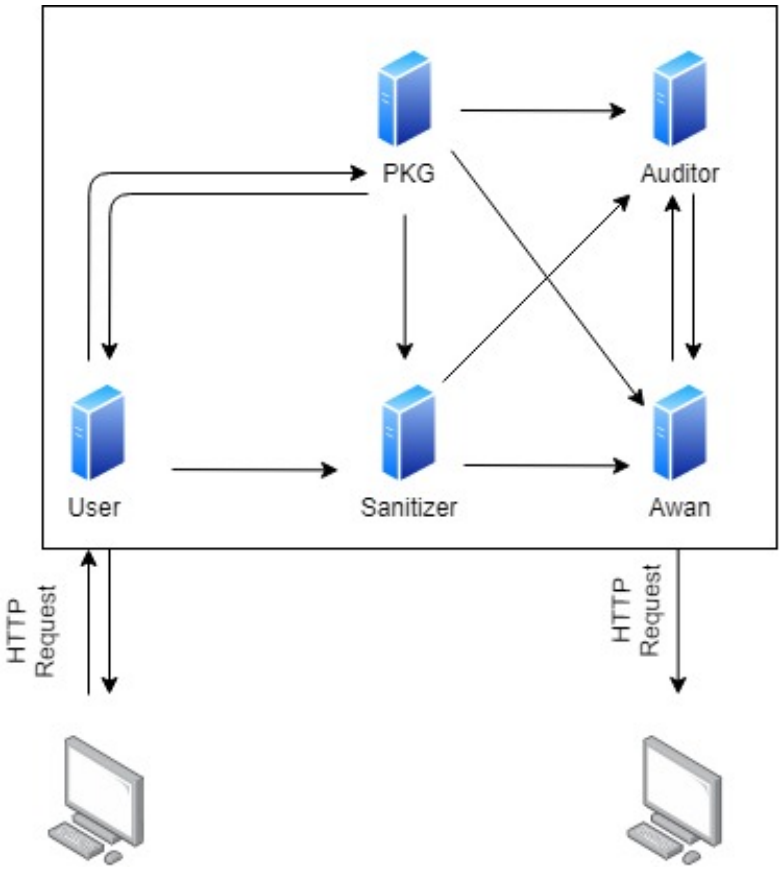
Auditor adalah pihak yang akan menverifikasi integritas data dari data yang telah diunggah di awan. Dalam sistem ini, siapapun bisa menverifikasi integritas data tersebut sebagai perwakilan pengguna bila memiliki metode verifikasi yang sesuai. [1]
Auditor kemudian mengecek apakah kebenaran dari bukti audit yang diberikan oleh awan dengan formula berikut:

$$e(\sigma, g) = e(g_1, g_2)^{\sum_{i \in I} v_i} \cdot e\left(\mu' \prod_{j=1}^{l} \mu_j^{ID_j}, g^{r_{ID}}\right)^{\sum_{i \in I} v_i} \cdot e(\prod_{i \in I} H(name||i)^{v_i} \cdot u^{\lambda},\ g^{r}) \quad \text{(III.6)}$$

Apabila persamaan III.6 ini berlaku, *file* tersanitasi yang disimpan di awan masih utuh; jika tidak, berarti *file* tersebut sudah tidak utuh.

3.3 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan serta desain dari sistem yang akan dibangun. Arsitektur sistem secara umum ditunjukkan pada Gambar 3.2.



Gambar 3.2: Arsitektur Sistem

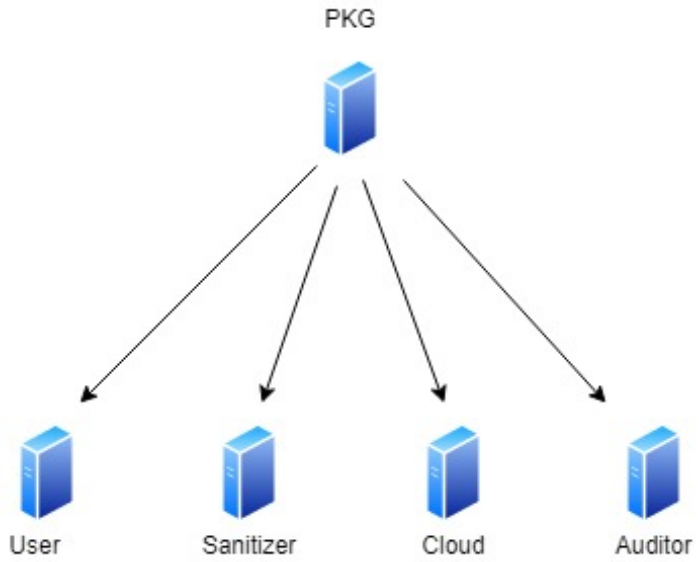
3.3.1 Desain Umum Sistem

Berdasarkan yang dijelaskan pada deskripsi umum sistem, dapat diperoleh kebutuhan sistem sebagai berikut:

1. *Private Key Generator* (PKG) untuk mempersiapkan sistem dan segala variabel yang digunakan untuk komputasi pembuatan dan validasi *signature*.

2. *User* untuk memberikan *file* identitas kepada klien, memproses dokumen yang diunggah oleh klien, dan membuat *signature* dari *file* yang diunggah.
3. *Sanitizer* untuk menverifikasi *file* asli, membersihkan *file* yang diberikan oleh *User*, dan membuat *signature* dari *file* yang telah dibersihkan.
4. Awan untuk menyimpan dokumen yang telah diunggah dan dibersihkan, menampilkan kapan *file* terakhir dicek, dan memberikan informasi yang diperlukan ketika Auditor meminta bukti verifikasi kepada Awan.
5. Auditor (Pengaudit) untuk melakukan komputasi dan verifikasi terhadap bukti yang diberikan Awan dan *Sanitizer*.

3.3.2 Perancangan PKG



Gambar 3.3: Desain PKG

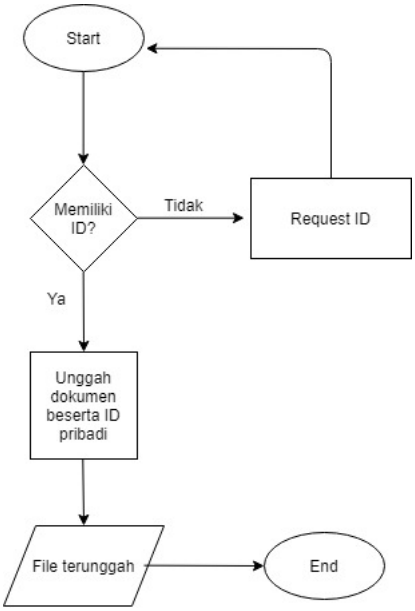
PKG adalah entitas yang dipakai sebagai penginisiasi sistem. Seluruh variabel yang dibuat dalam tahap 1 dilakukan di dalam PKG.

Di dalam PKG ditentukan panjang ID, bit sistem, bit bilangan prima yang digunakan, dan variabel lain yang diinisiasi secara acak guna memberikan keunikan di setiap instansi sistem yang dibuat. Sistem diinisiasi menggunakan modul *pbclib* yang dibungkus dalam Python. Tipe *pairing* yang digunakan dalam sistem keseluruhan adalah *pairing* tipe A, yang dibangun di atas kurva $y^2 = x^3 + x$. Untuk efisiensi, bilangan prima yang dibuat dalam PKG menggunakan bilangan prima Solinas, yang berbentuk $2^a + -2^n + -1$ dimana $0 < b < a$.[15] [16]

Setelah diinisiasi, PKG membagikan parameter-parameter ini

kepada seluruh bagian sistem, yang mana akan menggunakan variabel tersebut untuk komputasi integritas data.

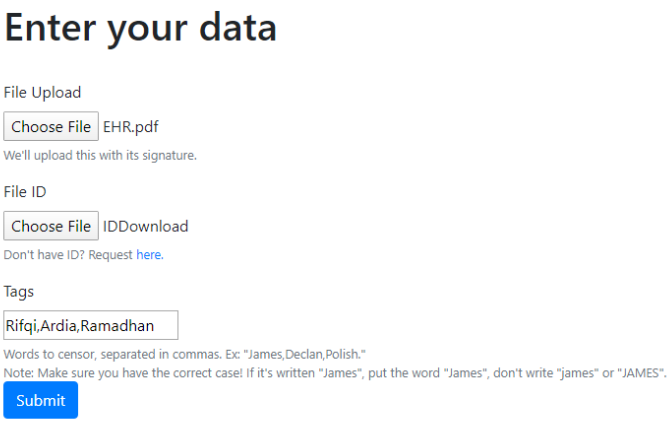
3.3.3 Perancangan *User*



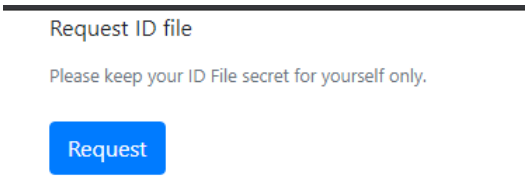
Gambar 3.4: Desain *User*

User adalah entitas yang dipakai sebagai antarmuka antara pengguna yang mengunggah dokumen dengan sistem untuk menyederhanakan interaksi dengan sistem. *User* bertugas membuat ID baru bila pengguna meminta, dan mempersiapkan dokumen yang diunggah untuk disanitasi oleh *sanitizer*, lengkap dengan informasi yang ingin dibersihkan, *file* ID milik pengunggah, identitas *file* itu sendiri, *signature* yang dibuat dengan persamaan III.3 dan beberapa variabel yang digunakan untuk melakukan transformasi *file*.

Antarmuka yang ditampilkan di *User* berbasis Flask. Komunikasi dengan subsistem lain menggunakan fitur *request* dari Python dan *jsonify* dari Flask. *File* yang dapat diproses oleh *User* adalah PDF. Tampilan dari dua antarmuka yang ada di *user* ditunjukkan gambar 3.5 dan 3.6



Gambar 3.5: Desain Antarmuka pembuatan *signature*

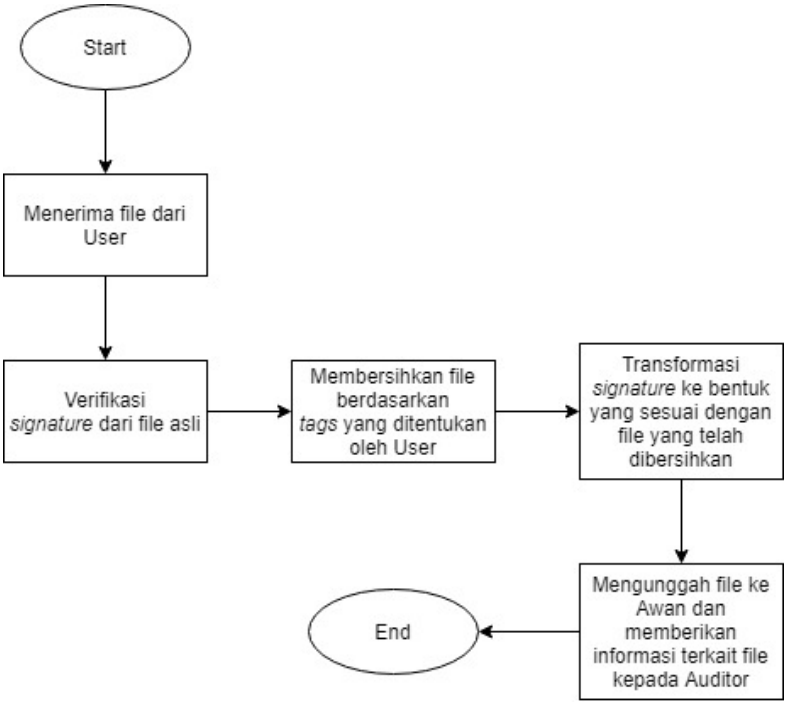


Gambar 3.6: Desain Antarmuka request ID

Di saat inisiasi, *User* mendapatkan parameter sistem dari PKG dan menyamakan instansi komputasi dengan seluruh subsistem lainnya.

3.3.4 Perancangan *Sanitizer*

Sanitizer akan menverifikasi bahwa *file* adalah *file* yang benar adanya berdasarkan *signature* yang dikirimkan bersamaan oleh *User* dengan persamaan III.2. Jika tidak benar, maka *Sanitizer* akan mengembalikan error kepada *User* dan melaporkan bahwa *file* gagal diunggah. Jika benar, maka *Sanitizer* melanjutkan untuk membersihkan *file* dari informasi sensitif. Semua kata-kata yang dihapus didefinisikan dalam sebuah *list tags* yang dikirim. Dengan bantuan pdf-redactor, kata-kata dalam *tags* yang terkandung di dalam *file* PDF yang diunggah dihapus atau digantikan oleh *wildcard* supaya informasi sensitif tidak ada dalam dokumen yang terunggah ke awan. Setelah informasi sensitif terhapuskan, *Sanitizer* membuat *signature* dari *file* yang telah terbersihkan, karena *file* mengalami perubahan dengan dihapuskannya informasi sensitif dengan persamaan III.5. Setelah terbuat, blok *file* yang telah dibersihkan, *signature*, dan identitas *file* ke awan, dan memberikan informasi variabel *file* kepada Auditor. Gambar 3.7 menunjukkan alur kerja dari *Sanitizer*. Sama seperti *User*, ketika *Sanitizer* diinisiasi, *Sanitizer* mendapatkan parameter sistem dari PKG. Komunikasi dilakukan dengan API, dengan *method* POST yang dibuat menggunakan Flask.

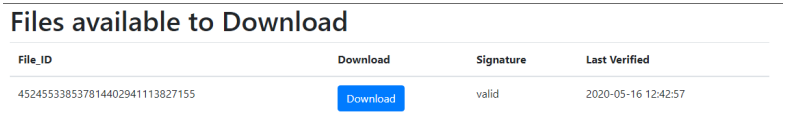


Gambar 3.7: Alur pemrosesan *Sanitizer*

3.3.5 Perancangan Awan

Awan dalam sistem ini bertugas menyimpan *file* dokumen dan memberikan bukti kepada Auditor untuk verifikasi *signature*. Setelah menerima blok data dari Sanitizer, Awan menyimpannya ke dalam basisdata MongoDB. Hanya Awan yang memiliki akses ke basisdata secara langsung. Ketika Auditor meminta bukti, Awan membuat bukti dengan cara yang dijelaskan di tahap 5. Untuk meningkatkan keamanan, blok data yang diverifikasi ditentukan secara acak. Antarmuka yang disediakan Awan berbasis Flask, dan menunjukkan identitas *file*, tombol unduh *file*, status *file* (apakah valid atau invalid), dan

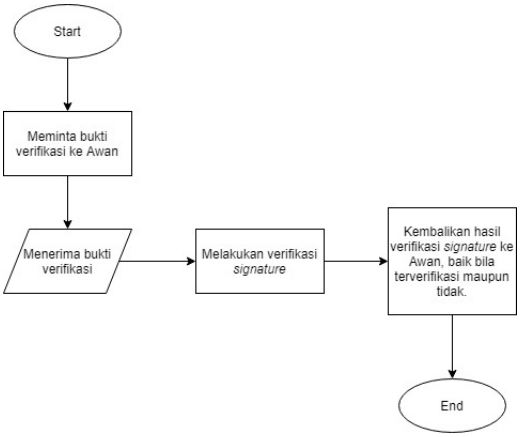
waktu terakhir *file* terverifikasi kebenarannya. Tampilan antarmuka digambarkan pada gambar 3.8. Ketika Awan menerima permintaan unduh dokumen oleh pengguna, Awan menyusun *file* dari blok-blok data yang tersimpan dalam basisdata kembali menjadi PDF lalu disalurkan kepada pengguna. Sama seperti subsistem lainnya, ketika Awan diinisiasi, Awan mendapatkan parameter sistem dari PKG.



Gambar 3.8: Desain Antarmuka daftar *file* yang terdapat di Awan

3.3.6 Perancangan Auditor

Dalam sistem yang dibangun, Auditor digunakan untuk menverifikasi apakah *file* yang berada di Awan masih valid atau tidak. Pengecekan ini dimulai dengan meminta bukti verifikasi kepada Awan dengan posisi blok data yang diverifikasi acak, dan setelah menerimanya, Auditor memeriksa apakah *file* dengan *signature* yang diberikan masih valid atau tidak. Hasil dari seluruh verifikasi ini, baik yang lolos verifikasi maupun tidak, dikembalikan ke awan. Alur kerja digambarkan pada gambar 3.9. Auditor berjalan secara terjadwal menggunakan modul *APScheduler* untuk Flask, dalam artian tidak ada *trigger* yang memulai berjalannya Auditor. Pengecekan langsung berjalan ketika Auditor diinisiasi dan berjalan setiap periode waktu.



Gambar 3.9: Alur Auditor

Sama seperti subsistem lainnya, ketika Auditor diinisiasi, Auditor mendapatkan parameter sistem dari PKG.

3.3.7 Perancangan Basis Data

Dalam sistem diperlukan basis data untuk menyimpan data-data yang disimpan di Awan. Basisdata ini dibuat dengan MongoDB. Karena bentuknya yang fleksibel dan mirip JSON, sehingga penyimpanan bisa disederhanakan dan dipercepat pencariannya. Data yang disimpan dalam basisdata ini adalah identitas *file*, blok-blok data dari *file*, dan tiap-tiap *signature* untuk tiap blok data. MongoDB akan diakses dalam program dengan menggunakan modul pymongo.

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Pada bab ini akan dibahas implementasi dari perancangan setiap subsistem pada bab sebelumnya. Setiap subsistem akan dibahas proses pembuatan dilengkapi dengan kebutuhan dan potongan kode dari sistem.

4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai berikut.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Virtual Private Server dengan CPU Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz dengan 8 Core Processor, RAM 32GB, dan SSD 25GB

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

1. Sistem Operasi Ubuntu 18.04 LTS 64 Bit.
2. Python versi 3.6.9
3. Flask versi 1.1.2 untuk komunikasi antar subsistem
4. *PBC Library* versi 0.5.14 untuk komputasi *signature* dan gmplib-dev versi 2:6.1.2 sebagai pendukung *PBC Library*

4.2 Persiapan *environment*

Sebelum mulai mengimplementasi, *environment* sistem perlu dipersiapkan agar implementasi bisa dilaksanakan. Persiapan yang paling pertama dilakukan adalah instalasi *PBC Library*. [6]

Setelah itu, instalasi berbagai alat dan pustaka dipersiapkan menggunakan Python Package Index (PyPI). [17] Alat dan Pustaka yang dipersiapkan untuk Python, yakni:

1. APScheduler
2. Flask
3. pdf-redactor
4. pymongo
5. pypbc
6. requests

Tabel 4.1: Daftar Antarmuka Flask dari tiap subsistem

No	Subsistem	Rute	*Method*	Keterangan
1	PKG	/params	GET	untuk meminta parameter pairing
		/showall	GET	untuk meminta parameter sistem (pp)
		/yourkey	POST	untuk membuat private key dari ID yang diberikan oleh User
2	User	/requestID	GET	untuk meminta kepada User untuk dibuatkan ID untuk pengguna
		/sigGen	GET	menampilkan antarmuka agar pengguna bisa mengunggah *file* yang ingin disanitasi beserta ID yang telah diminta

			POST	menerima dan mengirimkan *file* dan ID ke Sanitizer untuk diproses lebih lanjut
		/IDDownload	POST	mengunduh *file* ID untuk digunakan sebagai identitas pengguna
3	*Sanitizer*	/send	POST	menerima *file* dan membersihkannya dari informasi sensitif, kemudian mengunggahnya ke Awan
4	Awan	/documents	GET	melihat daftar *file* yang telah dibersihkan dan telah diunggah
		/challenge	POST	menerima tantangan dari Auditor dan mengirimkan kembali bukti verifikasi kepada Auditor
		/requestfileblocks	POST	mengirimkan blok-blok data yang akan diverifikasi oleh Auditor

		/acceptverification	POST	menerima hasil audit dari Auditor untuk ditunjukkan di sebelah daftar *file* yang telah diunggah
		/downloadfile	POST	mengunduh *file* yang dimaksud dari Awan
5	Auditor	/recvtau	POST	menerima informasi tau dari *file* yang ke depannya akan dicek secara berkala.

4.3 Implementasi PKG

PKG bertugas untuk menginisiasi seluruh variabel yang dibutuhkan sistem serta membuat *private key* yang diminta oleh *User*. Sehingga secara garis besar ada dua hal yang yang dilakukan PKG:

1. Menginisiasi variabel dan membagikannya ke seluruh subsistem.
2. Membuat *private key* berdasarkan ID dari permintaan *User*.

4.3.1 Implementasi inisiasi variabel

Untuk implementasi inisiasi variabel, inisialisasinya dijelaskan pada algoritma 1.

Algorithm 1: Inisialisasi variabel dengan *PBC Library*

Data: q, r
params ← generate_parameters(q, r) ;
pairing ← Pairing(params) ;
g ← pairing.random(G1) ;
$g2$ ← pairing.random(G1) ;
x ← pairing.random(Zr) ;
for $j = 0$ *to* $q + 1$ **do**
  $\mu[j]$ ← pairing.random(G1) ;
end
u ← pairing.random(G1) ;
$g1 \leftarrow g^x$;
$msk \leftarrow g2^x$;

Kode Sumber IV.1: Algoritma Inisiasi Variabel

Seluruh inisiasi pada algoritma 1 tersebut berdasarkan desain pada tahap 1 Persiapan pada bab sebelumnya [1], dengan input q sebagai panjang bit sistem, dan r sebagai panjang bit bilangan prima yang dibuat. Seluruh variabel telah dibuat, kemudian dibuatkan fungsi sebagai antarmuka berupa API antara PKG dengan subsistem lain yang berguna untuk berbagi variabel. Variabel yang telah diinisiasi di PKG akan digunakan pada algoritma-algoritma subsistem lain yang akan dibahas selanjutnya.

4.3.2 Implementasi pembuatan *private key*

Implementasi pembuatan *private key* dibuat dengan persamaan III.1, dan dibuat dengan ID yang diberikan oleh *User* dengan algoritma 3:

1. Menginisiasi variabel dan membagikannya ke seluruh subsistem.
2. Membuat *private key* berdasarkan ID dari permintaan *User*.

Untuk implementasi pembuatan private key, algoritmanya menggunakan 2.

Algorithm 2: Pembuatan *private key*

```
Data: ID
Result: Private keys
if ID in savedID then
    return savedID[ID], rID ;      /* return jika private key sudah
      pernah dibuat */
end
rID ← pairing.random(Zr) ;
temp ← μ[1]^ID[0] ;
for i = 2 to q + 1 do
    temp ← temp*(μ[i]^ID[i−1]) ;
end
skaksenID ← (g2^x)*((μ[0]*temp)^rID) ;
skdoubleaksenID ← g^rID ;
skID = [skaksenID, skdoubleaksenID] ;
savedID[ID] = [skID, rID] ;              /* simpan setelah membuat */
return skID, rID
```

Kode Sumber IV.2: Algoritma Pembuatan *Private Key*

Dari algoritma 2, setelah persamaan III.1 diaplikasikan, hasilnya dikembalikan langsung ke *User* untuk diproses lebih lanjut. Selain itu, private key juga disimpan agar ID yang sama tidak perlu membuat ulang *private key*, hanya menggunakan ulang yang sudah ada. Zr dalam algoritma-algoritma sekarang dan selanjutnya ekuivalen dengan Z_p^*, yakni *prime field* dengan elemen-elemen tidak nol. Sedangkan G_1 adalah *group* dengan orde p. rID akan digunakan untuk membuat *signature* yang nantinya dibuatkan oleh *User* ID dikirim dengan cara yang sama seperti bagian pembuatan *private key*, dengan antarmuka berupa API yang bisa dilihat rutenya yang tertera pada tabel 4.1.

4.4 Implementasi *User*

User memiliki tugas sebagai antarmuka antara pengguna dengan Awan, sebagai pihak yang memintakan *private key* kepada PKG agar tidak perlu disentuh oleh pengguna, dan

tempat ID dibuat. Hal-hal yang dilakukan oleh *User* yakni:

1. Meminta parameter sistem dan *pairing* kepada PKG.
2. Menerima unggahan *file* bersamaan dengan ID dari pengguna.
3. Membuat ID bila pengguna membutuhkannya.
4. Meminta *private key* kepada PKG ketika *file* diunggah dan dibutuhkan oleh *User*.

Saat item diinisiasi, *User* meminta parameter sistem dan *pairing*kepada PKG, melalui *request* GET /params dan /showall di subsistem PKG. Setelah itu, *User* siap untuk melakukan fungsinya.

4.4.1 Implementasi pembuatan ID

Panjang ID disamakan dengan panjang bit sistem untuk menghindari kesalahan. ID juga dipastikan berbeda dalam setiap pembuatannya. Algoritma pembuatan ID cukup sederhana, seperti ditunjukkan pada algoritma 3. Algoritma ini membuat sebuah *array* sebesar bit sistem *pairing*, dan unik untuk setiap ID yang dibuat. Banyak ID yang bisa ditampung adalah 2^q dengan q sebagai bit sistem *pairing*.

Algorithm 3: Pembuatan ID

```
Data: ID, l
Result: List bit sepanjang l bit
ID = generate_bit_list(l) ;
while ID in savedID do
    ID = generate_bit_list(l) ;        /* buat ulang jika sudah pernah
      dibuat */
end
return ID
```

Kode Sumber IV.3: Algoritma Pembuatan ID

4.4.2 Implementasi verifikasi *private key*

Private key sebelum digunakan untuk membuat *signature* perlu diverifikasi kebenarannya. Setelah meminta dari PKG, kemudian *User* menverifikasi kebenarannya dengan persamaan III.2. Kebenaran dicek dengan memanfaatkan aplikasi *pairing*.

Algorithm 4: Verifikasi *private key*

Data: skID, ID
; /* skID berisi sk'ID dan sk''ID */
Result: Cek apakah skID valid atau tidak
temp ← $\mu[1]^{ID[0]}$;
for *i = 2 to q+1* **do**
  temp ← temp∗$\mu[i]^{ID[i-1]}$;
end
left ← pairing.apply(skID[0], g) ;
right ← pairing.apply(temp∗μ[0], skID[1]) ∗ pairing.apply(g1, g2) ;
return left == right

Kode Sumber IV.4: Algoritma Verifikasi *Private Key*

4.4.3 Implementasi pembuatan *signature*

User menerima input dari pengguna melalui antarmuka Flask di rute /sigGen. /sigGen menampilkan halaman seperti gambar 3.5. Pengguna sebelum mengunggah harus memiliki 3 hal: *file* PDF yang akan diunggah, *file* ID yang telah diminta kepada subsistem *User* sebelumnya, dan kata-kata yang ingin dihapus dari *file*, yang dalam konteks ini adalah informasi sensitif, dipisahkan oleh tanda koma. Setelah memiliki 3 hal tersebut, pengguna bisa mengunggah data ke Awan, yang sebelumnya akan dilewatkan melalui *Sanitizer*. *User* kemudian memproses unggahan pengguna untuk dibuatkan variabel yang dibutuhkan oleh *Sanitizer* untuk melakukan pembersihan *file* dari informasi sensitif. Algoritma pembuatan *signature* adalah seperti pada algoritma di bawah ini:

Algorithm 5: Pembuatan *signature*

```
Data: file, ID
Result: Signature
fileblocks ← pairing.split_to_blocks(Zr, file, BLOCK_SIZE) ;
r ← pairing.random(Zr) ;
file_identifier ← pairing.random(Zr) ;
verification_value ← g^r ;
sigma ← Array ;
for i=0 to fileblocks.length do
    temp ← μ[1]^ID[0] ;
    for j=2 to q+1 do
        temp ← temp*μ[j]^ID[j−1] ;
    end
    left ← (g2^x) * ((μ[0] * temp)^rID) ;
    thehash ← pairing.from_hash(G1, file_identifier.toString + i.toString) ;
    right ← (thehash * u^fileblocks[i])^r ;
    sigma[i] ← left*right ;
end
return sigma ;        /* sigma sama dengan Φ di persamaan III.3 */
```

Kode Sumber IV.5: Algoritma Pembuatan *Signature*

Algoritma 5 berdasarkan persamaan III.3. Di awal, *file* dipecah menjadi beberapa blok data sesuai besar *byte* blok data yang telah ditentukan sebelumnya. Lalu *signature* dibuat dengan memanfaatkan beberapa variabel acak dan persamaan *hash*. Setelah itu, dibuatkan $\beta \leftarrow u^r$ yang nantinya dipakai untuk verifikasi variabel transformasi dan $\tau \leftarrow file\_identifier \parallel g^{rID} \parallel g^r$ yang dipakai untuk pembuatan *signature* selanjutnya dan verifikasi blok data yang dikirim ke *sanitizer*.

4.5 Implementasi *Sanitizer*

Sanitizer hanya memiliki satu antar muka yang hanya boleh diakses oleh *User*, yakni melalui *request* ke /send dengan *method* POST. Ketika data sampai, data di-*parse*, dan *Sanitizer* langsung

mengecek kebenaran seluruh data yang dikirim yang algoritmanya dijelaskan pada algoritma 6 dan 7. Pada algoritma 6, kembali memanfaatkan aplikasi *pairing*.

Algorithm 6: Verifikasi *signature*

```
Data: fileblocks, sigma, ID, grID, gr
Result: Kebenaran signature
for i =0 to fileblocks.length do
    lefthandside ← pairing.apply(sigma[i], g) ;
    righthandleft ← pairing.apply(g1, g2) ;
    temp ← μ[1]^ID[0] ;
    for j = 2 to q+1 do
        temp ← temp ∗(μ[j]^ID[j−1]) ;
    end
    righthandmiddle ← pairing.apply(temp∗μ[0], grID) ;
    righthandright ← pairing.apply(pairing.from_hash(G1,
      file_identifier.toString + i.toString)∗u^fileblocks[i], gr) ;
    righthandside ← righthandleft ∗ righthandmiddle ∗ righthandright ;
    if lefthandside = righthandside then
        return False ;
    end
end
return True ;
```

Kode Sumber IV.6: Algoritma Verifikasi *Beta*

Algorithm 7: Verifikasi beta

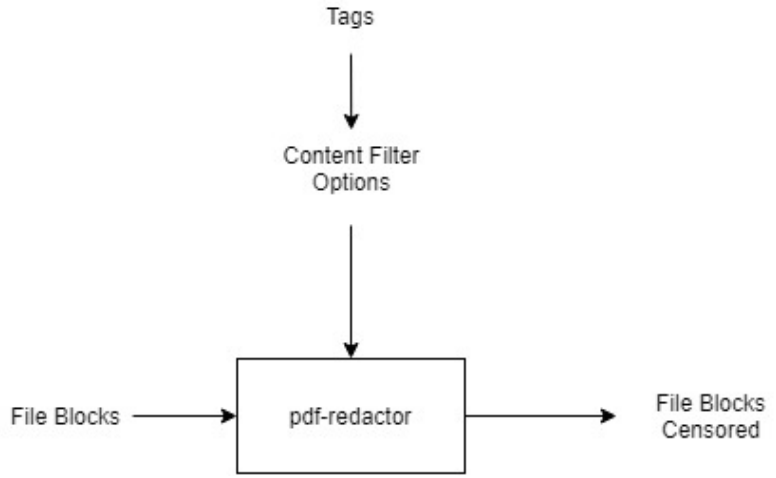
```
Data: β
Result: Kebenaran transformation value
return pairing.apply(u, gr) == pairing.apply(beta, g)
```

Kode Sumber IV.7: Algoritma Verifikasi *Signature*

4.5.1 Implementasi sanitasi atau pembersihan dari informasi sensitif

Dengan bantuan pdf-redactor, PDF dapat secara mudah menghapus informasi sensitif dan mengubahnya menjadi

wildcard atau *whitespace*, cukup dengan menerapkan opsi content_filters yang berisikan informasi sensitif yang telah dikompilasi dengan pustaka *regular expression* dari Python, lalu mengeksekusinya. Bagaimana masukan dan keluaran bekerja digambarkan pada Gambar 4.1. *Tags* yang diberikan oleh pengguna ketika mengunggah *file* dikompilasi dengan *regular expression*. Kemudian didefinisikan bagaimana informasi sensitif tersebut tergantikan, bisa dengan *wildcard* ataupun *whitespace*. Kedua hal tersebut dimasukkan dalam opsi *content filter*, kemudian dieksekusi.



Gambar 4.1: Alur kerja PDFRedactor

4.5.2 Implementasi transformasi *signature*

Algoritma transformasi ini mirip dengan algoritma 5. Dalam sebuah penelitian oleh Wenting Shen yang jadi referensi utama dalam tugas akhir ini, karena blok data sensitif yang tidak dinamis, yang artinya jika satu blok data mengandung informasi sensitif (bukan keseluruhan blok data, tapi hanya mengandung),

maka satu blok data tersebut akan dianggap informasi sensitif walaupun tidak semuanya mengandung informasi sensitif. Namun dalam praktiknya, tipe *file* seperti PDF yang lumrah digunakan untuk dokumen formal, blok data tidak bisa semerta-merta dihapus atau digantikan keberadaannya, karena posisinya yang dinamis, informasi sensitif bisa berada di antara blok data. Ini menjadi perbedaan mendasar referensi dengan tugas akhir ini. Sehingga bentuk algoritmanya tidak memanfaatkan β seperti yang ada di referensi, melainkan membuat ulang *signature*, dengan algoritma 8. [1]

Algorithm 8: Transformasi *signature* baru

Data: fileblockscensored, file_identifier
Result: Signature baru
for *i=0 to fileblockscensored.length* **do**
  temp ← $mu[1]^{ID[0]}$;
  for *j=2 to q+1* **do**
    temp ← temp∗$\mu[j]^{ID[j-1]}$;
  end
  left ← $(g2^x) * ((\mu[0] * temp)^{rID})$;
  thehash ← pairing.from_hash(G1, file_identifier.toString + i.toString) ;
  right ← $(thehash * u^{fileblockscensored[i]})^r$;
  new_sigma[i] ← left*right ;
end
return new_sigma

Kode Sumber IV.8: Algoritma Transformasi *Signature* Baru

4.6 Implementasi Awan

Awan berfungsi untuk menyimpan blok-blok data beserta hasil audit dan menjawab bukti verifikasi yang diminta oleh Auditor. Dari deskripsi tersebut, terdapat tiga hal yang menjadi tugas utama subsistem Awan:

1. Menyediakan akses ke *file* PDF yang telah disimpan di Awan.

2. Menyediakan informasi mengenai hasil verifikasi *signature*.
3. Menjawab bukti verifikasi kepada Auditor.

Dua nomor pertama dilakukan dari antarmuka dan basis data, sehingga akan dibahas di dalam bagian selanjutnya.

4.6.1 Implementasi pembuatan bukti

Sebelum memulai pembuatan, Awan mengambil blok data dan *signature*-nya dari basis data. Kemudian Awan memberikan blok data yang tersimpan di Awan kepada Auditor dan membuat bukti berdasarkan masukan yang diberikan oleh Auditor, karena Auditor yang menentukan blok mana yang akan diverifikasi. λ adalah formula *sum*, dan sigma dari *product*. Dengan syarat I dan tantangan harus diterima dahulu oleh Awan dari Auditor untuk membuat bukti dari tantangan tersebut.

Algorithm 9: Pembuatan Bukti

Data: chal, I, *file*_identifier, fileblocks, *signature*
Result: Bukti λ dan sigma
$\lambda \leftarrow 0$;
for *i in I* **do**
  $\lambda +=$ fileblocks[i]$*chal[i.toString]$;
end
sigma $\leftarrow$ 1 ;
for *i in I* **do**
  sigma $*= signature[i]^{chal[i.toString]}$;
end
return λ, sigma ;

Kode Sumber IV.9: Algoritma Pembuatan Bukti

4.7 Implementasi Auditor

Auditor merupakan entitas yang menverifikasi apakah data di dalam Awan masih benar atau tidak. Awal sebelum pengecekan,

Auditor meminta terlebih dahulu blok data yang ingin diverifikasi. Setelah itu, Auditor meminta bukti verifikasi kepada Awan yang blok-bloknya dipilih secara acak guna meningkatkan keamanan dalam melakukan audit. Hal ini dilakukan untuk semua data yang tercatat oleh Auditor berdasarkan data yang dikirimkan oleh *Sanitizer*. Setelah Awan menjawab, setelah itu algoritma verifikasi bukti dari persamaan III.6 dijalankan. Benar maupun salah, akan dikembalikan ke Awan untuk ditampilkan kepada pengguna hasilnya. Algoritma ini dijalankan setiap beberapa waktu sekali, untuk membuat pengguna dapat melakukan cek kebenaran tanpa perlu menyimpan *signature*-nya.

4.7.1 Implementasi permintaan bukti

Dalam algoritma 10, dibentuk angka acak dengan ukuran acak di bawah ukuran blok data yang akan diverifikasi, seperti yang dijelaskan pada tahap 5 Pembuatan Bukti. Di awal, Auditor menentukan secara acak c yakni jumlah blok data yang akan diverifikasi. Kemudian mengisi *array I* sebanyak c elemen secara unik dan acak dari 0 hingga panjang blok data. Kemudian Auditor membuat angka acak untuk setiap I sebagai tantangan kepada Awan. Kemudian tantangan, I, dan identitas *file* dikirim ke Awan.

Algorithm 10: Permintaan Verifikasi

```
Data: fileblocks, file_identifier
Result: chal, I, file_identifier
c ← random(0, fileblocks.length-1) ;
I ← Array ;
for i=0 to i=c do
    temp ← randominteger(0, fileblocks.length-1) ;
    while temp in I do
        temp ← randominteger(0, fileblocks.length-1) ;    /* memastikan
          semua elemen I unik */
    end
end
chal ← Array ;
for i in I do
    vi ← pairing.random(Zr) ;
    chal.append(vi) ;
end
return chal, I, file_identifier
```

Kode Sumber IV.10: Algoritma Permintaan Verifikasi

4.7.2 Implementasi verifikasi bukti

Setelah mendapatkan jawaban dari Awan, maka persamaan III.6 dijalankan. Algoritma 11 menunjukkan cara menverifikasinya untuk tiap-tiap *file* yang akan diverifikasi. Kerumitan verifikasi ini sebesar c yang mana akan berbeda-beda di setiap tantangannya.

Setelah verifikasi selesai, Auditor mengembalikan hasil verifikasi ke Awan untuk ditunjukkan kepada pengguna.

Algorithm 11: Verifikasi *signature file*

```
Data: sigma, λ, vi
Result: Kebenaran signature file
total_vi ← 0 ;
for i in chal do
    total_vi ← total_vi + i ;                    /* total semua vi */
end
pairingleft ← pairing.apply(sigma, g) ;
pairingrightleft ← (pairing.apply(g1, g2)) ^total_vi ;
temp ← μ[1]^ID[0] ;
for i=2 to i=q+1 do
    temp ← temp ∗μ[i]^ID[i−1] ;
end
pairingrightmiddle ← (pairing.apply(temp∗μ[0], grID)) ^total_vi ;
temp2 ← 1 ;
for i in I do
    temp2 ← temp2 ∗ (pairing.from_hash(G1, file_identifier.toString ||
      i.toString)) ^chal[i] ;
end
temp2 ← temp2 ∗u^λ ;
pairingrightright ← pairing.apply(temp2, gr) ;
pairingright ← pairingrightleft ∗ pairingrightmiddle ∗ pairingrightright ;
return pairingleft == pairingright ;
```

Kode Sumber IV.11: Algoritma Verifikasi *Signature File*

4.8 Implementasi Basis Data

Basis data yang digunakan adalah MongoDB, tipe basis data yang menyimpan data berupa *documents*. *Documents* ini mirip seperti tipe data *dictionary* dalam Python sehingga dokumen bisa langsung disimpan di basis data tanpa perlu konversi.

Tabel 4.2: Tabel Basis Data

No	*Field*	*Value type*	Catatan
1	file_identifier	*String*	Nomor identifikasi *file*
2	file_blocks	*Array of Strings*	Blok-blok data dari *file*
3	signature	*Array of Strings*	*Signature* dari tiap-tiap blok data yang tersimpan
4	file_status	*String*	Status kebenaran dari *file*. Bernilai ”valid” atau ”invalid”
5	last_verified	*String*	Waktu verifikasi berhasil paling terakhir

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

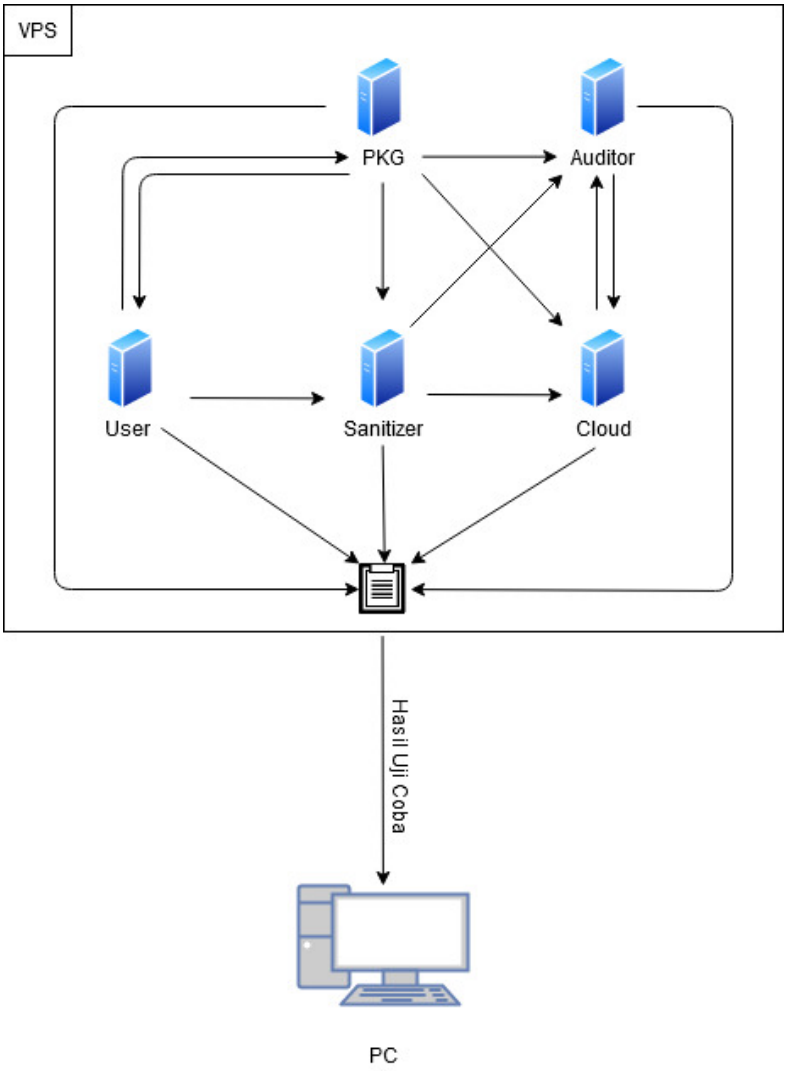
Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang sudah dibuat. Sistem akan diuji coba fungsionalitasnya dengan menjalankan skenario pengujian fitur-fitur dari sistem yang dibuat. Sistem juga akan diuji coba performa dengan skenario pengujian beban terhadap sistem. Uji coba dilakukan untuk mengevaluasi kinerja dari sistem dengan lingkungan uji coba yang ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan uji coba sistem ini terdiri dari beberapa komponen yang dijalankan pada 1 mesin virtual (VM) dengan beberapa port virtual. Komponen itu yakni PKG, *User*, Sanitizer, Awan, dan Auditor. Server yang digunakan adalah Virtual Private Server dari DigitalOcean. Spesifikasi dari VM bisa dilihat di Tabel 5.1

Tabel 5.1: Spesifikasi komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	*VM*	Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz dengan 8 Core Processor, 32GB RAM, SSD 25 GB	Ubuntu 18.04.3, MongoDB 4.2.7, Flask 1.0.3, Python 3.6



Gambar 5.1: Lingkungan Uji Coba

Uji coba dilakukan di dalam VPS yang telah tersedia,

masing-masing subsistem kecuali Awan akan memberikan waktu performanya ke dalam *file* yang mana dapat diakses oleh PC lokal untuk kemudian selanjutnya diolah.

5.2 Skenario Uji Coba

Uji coba ini dilakukan untuk menguji fungsionalitas dari sistem yang dibuat telah sesuai dengan perancangan dan sistem benar-benar diimplementasikan dan bekerja sesuai seharusnya. Skenario pengujian dibedakan menjadi 2 bagian yaitu:

- **Uji Fungsionalitas**
  Pengujian yang dilakukan berdasarkan fungsionalitas yang disediakan sistem.
- **Uji Performa**
  Pengujian yang dilakukan untuk melihat waktu yang diperlukan melakukan tiap tahapan sistem yang dapat dipantau.

5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas dibagi menjadi beberapa bagian antara lain yaitu pengguna mengunduh ID, mengunggah dokumen ke Awan dan telah tersanitasi, mengunduh *file*, dan verifikasi *signature*.

5.2.1.1 Uji Fungsionalitas Mengunduh ID

Uji coba ini bertujuan untuk mengizinkan user untuk mengunduh *file* ID agar dapat mengunggah dokumen yang ingin diunggah. Mengunduh ID diterapkan dengan melakukan tahap kedua dari tahapan yang telah dijelaskan. Uji coba ini dilakukan dengan mengakses halaman unduh ID pada *User* dan kemudian melakukan request. Uji coba dianggap berhasil apabila setelah request, pengguna mengunduh *file* tidak berukuran nol bernama ”IDDownload”.

5.2.1.2 Uji Fungsionalitas Unggah Dokumen

Uji coba ini bertujuan untuk memastikan dokumen dibuatkan *signature*-nya, disanitasi, dan berhasil diunggah ke Awan. Unggah dokumen ini diterapkan dengan melakukan tahap ketiga dan keempat dari tahapan yang telah dijelaskan pada bab 3.

Uji coba ini dilakukan dengan cara mengunggah *file* melalui antarmuka *User*, dan sistem akan mengembalikan status keseluruhan kebenaran *signature* yang telah dibuat. Dikembalikan berupa JSON. Uji coba dianggap berhasil apabila sistem mengembalikan status ”OK”.

5.2.1.3 Uji Fungsionalitas Unduh *File*

Uji coba ini bertujuan untuk memastikan dokumen dapat diunduh secara utuh dan tidak memiliki informasi sensitif yang telah didefinisikan oleh pengguna ketika mengunggah dokumen.

Uji coba ini dilakukan dengan cara mengakses halaman Awan, dan melakukan unduh *file*. Uji coba dianggap berhasil apabila *file* tersebut dapat dibuka dan bersih dari informasi sensitif.

5.2.1.4 Uji Fungsionalitas Verifikasi *Signature*

Uji coba ini bertujuan untuk memastikan dokumen masih utuh di dalam Awan. Dalam tugas akhir ini, uji coba ini dilakukan dengan melakukan dua tahap terakhir dari tahapan yang dijelaskan pada bab 3. Uji coba ini dilakukan dengan cukup menyalakan Auditor, dan setiap beberapa menit akan memperbarui status *file* di Awan. Uji coba berhasil apabila Awan menunjukkan hasil verifikasi pada 5 menit terakhir.

Tabel 5.2: Skenario uji fungsionalitas

No	Rute	Uji Coba	Harapan
1	`<IPUser> /requestID`	Mengirimkan request GET menuju halaman request ID	Request berhasil diterima *User*, kemudian *User* mengembalikan *file* tidak berukuran nol ”IDDownload”.
2	`<IPUser> /sigGen`	Mengirimkan request POST berisi *file* ID, *file* dokumen, dan *tags* dari kata yang akan dihilangkan dari *file* dokumen tersebut	Request berhasil diterima dan menampilkan status *file* unggahan dan waktu unggah.
3	`<IPAwan> /documents`	Mengirimkan request GET menuju halaman daftar dokumen dan menekan tombol Unduh	Request berhasil diterima Awan, kemudian Awan mengembalikan *file* PDF yang dimaksud.
4	`<IPAwan> /documents`	Mengirimkan request GET menuju halaman daftar dokumen	Request berhasil diterima Awan, kemudian Awan menunjukkan status *signature* setiap *file*.

5.2.2 Skenario Uji Coba Performa

Uji performa digunakan untuk menguji bagaimana performa sistem dalam melakukan satu atau beberapa tahapan.

Uji performa dilakukan dengan cara mengirimkan *file* dengan beberapa kali iterasi, lalu dirata-rata agar mendapat waktu yang mendekati sesungguhnya. Tahap yang dilakukan uji performa adalah tahap Persiapan, Ekstraksi, Pembuatan *Signature*, Sanitasi, dan dua tahap terakhir yang digabung menjadi satu; Pembuatan Bukti dan Verifikasi Bukti.

Dalam uji ini penulis akan bereksperimen dengan variabel-variabel yang dapat diubah dalam sistem; bit sistem *pairing*, bit bilangan prima, dan besar pecahan blok data.

Hasil yang diharapkan dari pengujian ini adalah sistem memiliki komponen perangkat keras yang mampu untuk menyimpan, membuat, mensanitasi, dan membagikan *file* minimal selama satu tahun.

5.3 Hasil Uji Coba dan Evaluasi

Berikut ini dijelaskan hasil coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada subbab sebelumnya.

5.3.1 Uji Fungsionalitas

Berikut ini dijelaskan hasil dari pengujian fungsionalitas pada sistem yang sudah dibangun.

5.3.1.1 Uji Fungsionalitas Mengunduh ID

Uji coba ini dilakukan dengan cara membuka halaman untuk meminta *file* ID dari *User*, lalu menekan tombol ”Request”.

Tabel 5.3: Hasil uji fungsionalitas unduh ID

No	Unduhan	Harapan	Hasil
1	*File binary* ID	*File* Berhasil Terunduh	OK

5.3.1.2 Uji Fungsionalitas Unggah Dokumen

Uji coba dilakukan dengan cara membuka halaman pembuatan *signature*, lalu mengisikan form sesuai dengan persyaratan sistem untuk membersihkan *file* dari informasi sensitif. Form berisi 3 masukan: *file* PDF yang akan dibersihkan, *file* ID yang diminta pada bagian sebelumnya, dan masukan berupa teks berisi kata-kata informasi sensitif yang dipisahkan dengan tanda koma.

Tabel 5.4: Hasil uji fungsionalitas unggah dokumen

No	Unggahan	Harapan	Hasil
1	Mengunggah dokumen PDF	*File* berhasil diproses dengan sistem mengembalikan status ”OK”	OK

5.3.1.3 Uji Fungsionalitas Unduh *File*

Uji coba dilakukan dengan cara membuka halaman daftar dokumen pada Awan, lalu menekan tombol ”Unduh” untuk melakukan request kepada Awan untuk membangun *file* PDF dari basis data.

Tabel 5.5: Hasil uji fungsionalitas unduh *file*

No	Unduhan	Harapan	Hasil
1	*File* PDF yang telah dibersihkan	Sistem berhasil membangun *file* PDF dan mengembalikan *file* kepada user	OK

5.3.1.4 Uji Fungsionalitas Verifikasi *Signature*

Uji coba ini dilakukan dengan cara membuka halaman daftar dokumen pada Awan dan melihat apakah verifikasi terakhir terjadi pada beberapa menit terakhir. Namun jika hasil verifikasi pernah tidak valid, tanggal dan jam verifikasi terakhir tidak akan berubah dan *file* dianggap tidak valid.

Tabel 5.6: Hasil uji fungsionalitas verifikasi *signature*

No	Hasil Verifikasi	Harapan	Hasil
1	Valid	Tanggal verifikasi diperbarui dan terjadi di beberapa menit terakhir.	OK
2	Tidak Valid	Tanggal verifikasi tidak diperbarui	OK

5.3.2 Hasil Uji Performa

Uji performa dilakukan pada seluruh subsistem, yakni PKG, *User*, *Sanitizer*, Awan, dan Auditor. Pengujian dilakukan dengan mengunggah *file* ke Awan, dan memantau verifikasi di daftar dokumen di Awan. Variabel-variabel yang dapat diubah akan divariasikan untuk melihat pengaruh variabel terhadap performa sistem. Pemantauan performa kecepatan dibuat berdasarkan tahap yang dijelaskan pada bab sebelumnya dan dibagi menjadi 5 tahap: Persiapan, Ekstraksi, Pembuatan *Signature*, Sanitasi, dan Pembuatan Bukti dan Verifikasi Bukti, dengan dua tahap terakhir digabung menjadi satu. Pengujian dilakukan dengan mengunggah dokumen sebanyak minimal 50 kali, dan memantau hasil verifikasi bukti sebanyak minimal 10 kali pada 51 *file* yang diunggah. Lalu waktu verifikasi bukti ini dibagi 51, untuk mendapatkan performa tiap-tiap dokumen yang diverifikasi.

Performa lain yang dipantau adalah penggunaan maksimum

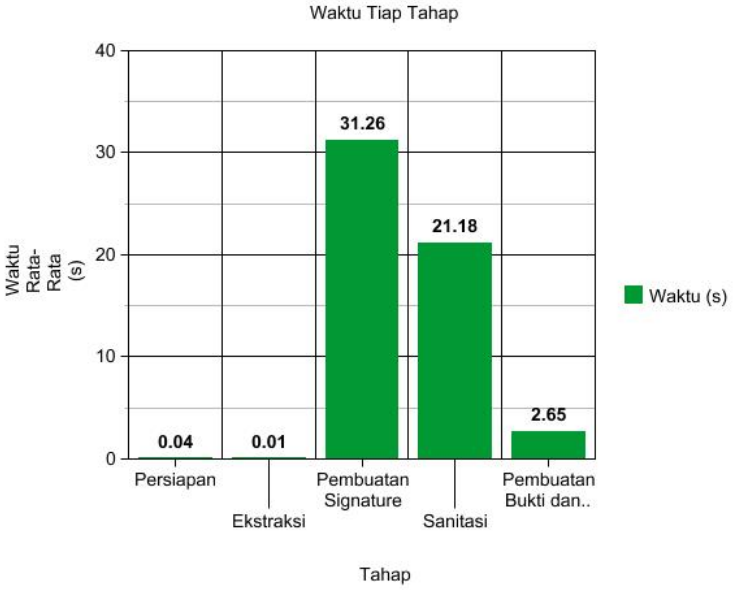
memori, dan dibagi kepada 4 subsistem yang relevan dengan sistem: PKG, *User*, *Sanitizer*, dan Auditor.

Data yang digunakan adalah *file* PDF berukuran 65095 *byte* yang berisikan contoh data yang informasi sensitifnya akan dihapus. Informasi sensitif yang dihapus adalah 3 kata berasal dari nama yang terkandung dalam dokumen.

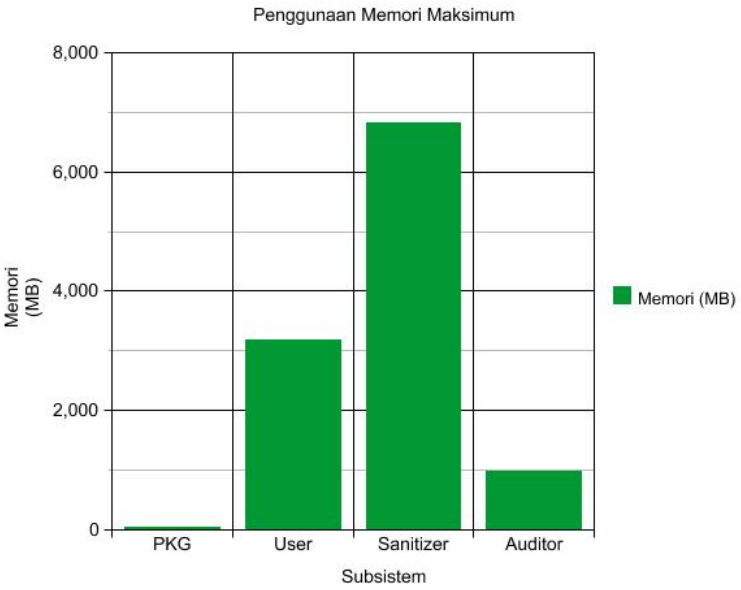
Sebagai eksperimen performa awal, parameter ditetapkan seperti pada Tabel 5.7.

Tabel 5.7: Eksperimen Performa

No	Bit *Pairing*	Bit Prima	*Byte* Blok Data
1	128	100	12
2	256	200	24



Gambar 5.2: Performa kecepatan tiap tahapan Eksperimen 1



Gambar 5.3: Penggunaan Memori Maksimum Eksperimen 1

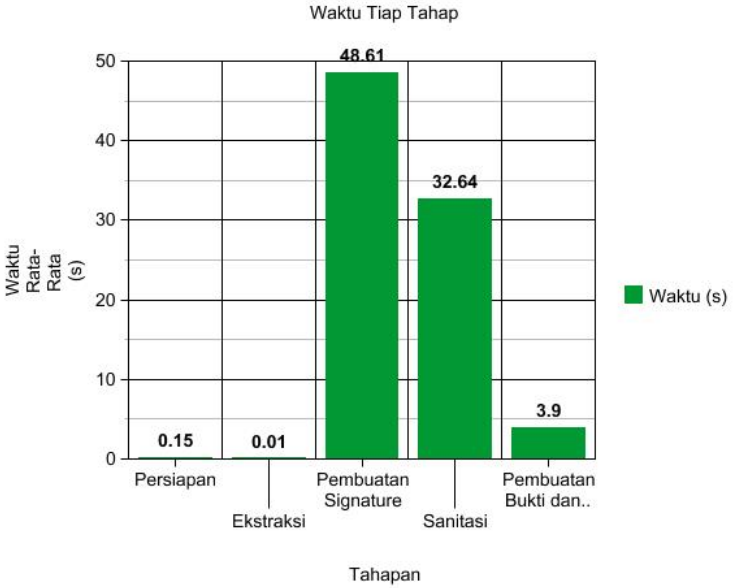
Dengan parameter yang ditulis di nomor 1 pada Tabel 5.7, Gambar 5.2 menunjukkan bahwa dalam skema ini, dua tahap yang performanya mencolok adalah Pembuatan *Signature* dan Sanitasi.

Hal ini dikarenakan dalam kedua tahap tersebut memiliki bagian komputasi yang kompleks, yakni komputasi pembuatan *Signature* yang memiliki kompleksitas $O(n) + O(s)$, di mana n adalah banyaknya blok data yang diproses dan s yaitu Sanitasi dari *file*. Sanitasi memiliki ongkos perhitungan yang cukup besar dikarenakan diperlukannya beberapa kali proses. Proses relevan yang membuat *Sanitizer* menggunakan memori besar adalah verifikasi *signature file* yang diberikan oleh subsistem *User*, pembersihan *file* dari informasi sensitif, dan pembuatan ulang *signature* dari data yang telah dibersihkan. Pembuatan Bukti dan

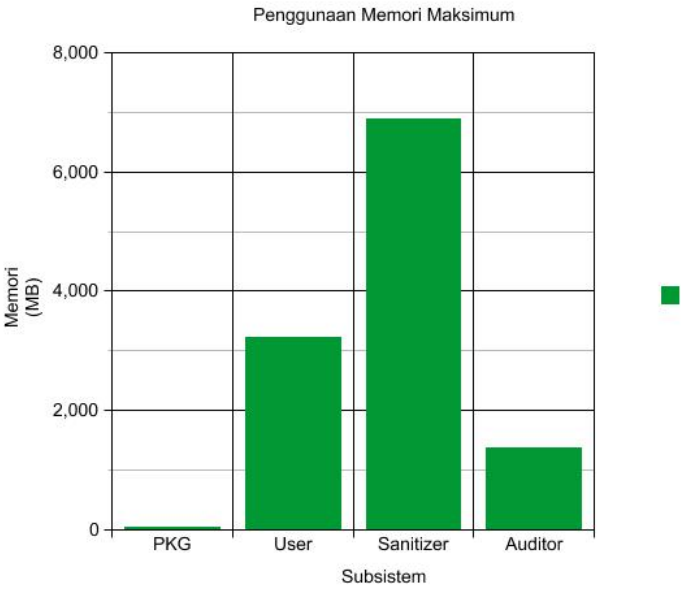
Verifikasi Bukti relatif cepat karena hanya memiliki kompleksitas sebesar c yaitu banyaknya blok data yang diverifikasi di tiap verifikasi.

Dari grafik di Gambar 5.3 menunjukkan bahwa *Sanitizer* menggunakan paling banyak memori ketimbang tahap lain. Hal ini dikarenakan Sanitasi melakukan tiga hal sekaligus yang menggunakan aktif menggunakan pustaka, PBC Library dan pdf-redactor, yaitu verifikasi *signature* dari *User*, sanitasi *file* dari informasi sensitif, dan membuat ulang *signature* untuk diletakkan di dalam Awan.

Eksperimen kedua, bit *pairing* ditetapkan menjadi 256 bit, bit bilangan prima 100 bit, dan besar blok data 24 *byte*.



Gambar 5.4: Performa kecepatan tiap tahapan Eksperimen 2



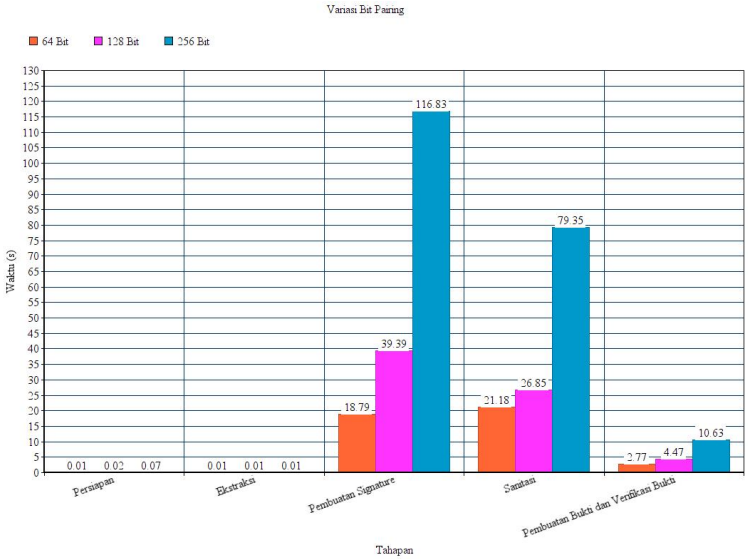
Gambar 5.5: Penggunaan Memori Maksimum Eksperimen 2

Dengan parameter eksperimen kedua, Gambar 5.4 menunjukkan kecepatan proses untuk tiap tahapan. Terlihat bahwa waktunya sedikit lebih lambat dari eksperimen pertama. Ongkos perhitungan dan relasi antara parameter satu dengan parameter lain dibahas bagian selanjutnya.

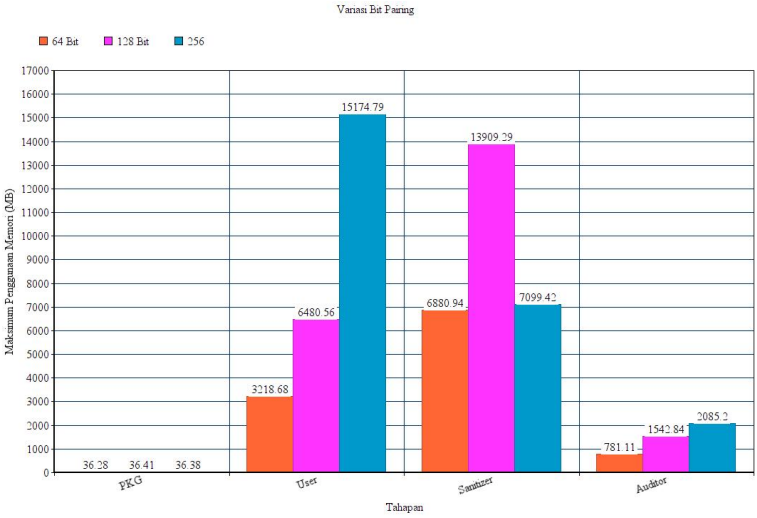
Selanjutnya dibuat uji performa dengan melakukan variasi terhadap bit *pairing*, bit bilangan prima, dan besar blok data.

Tabel 5.8: Tabel Pengujian Parameter Bit *Pairing*

No	Bit *Pairing*	Bit Prima	*Byte* Blok Data
1	64	50	6
2	128	50	6
3	256	50	6



Gambar 5.6: Variasi bit *pairing* terhadap kecepatan

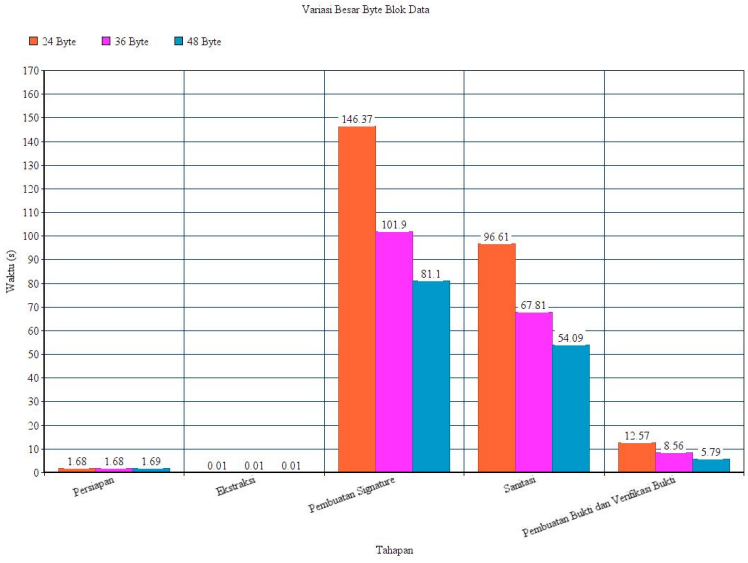


Gambar 5.7: Variasi bit *pairing* terhadap penggunaan memori maksimum

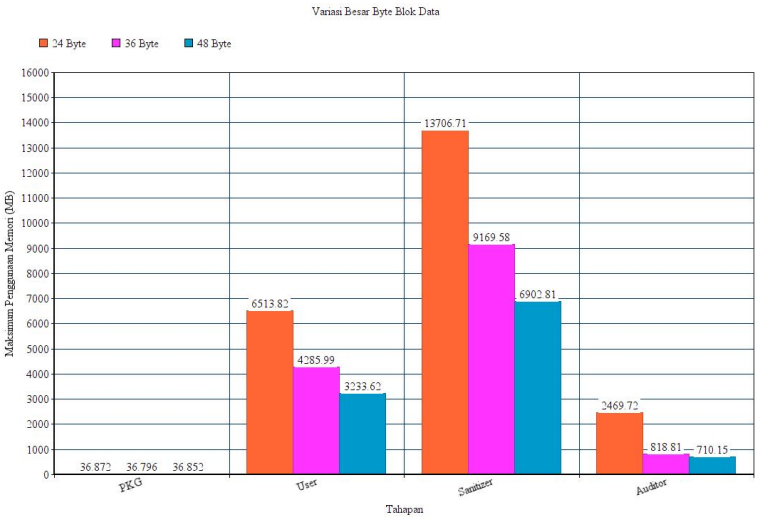
Pengujian bit *pairing* dapat dilihat di Tabel 5.8, dengan bit prima dan *byte* blok data dibuat tetap dan bit *pairing* divariasikan dari 64, 128, dan 256 bit. Eksperimen pada grafik di Gambar5.6 menggunakan bit prima 50 dan *byte* blok data 6. Bit pairing divariasikan untuk melihat performa di tiap bit *pairing* yang diuji. Dalam grafik pada Gambar 5.6, menunjukkan bahwa semakin besar bit *pairing* yang digunakan, semakin lama komputasi berjalan. Untuk performa maksimum penggunaan memori, terkecuali subsistem *Sanitizer*, menunjukkan bahwa semakin besar bit *pairing*, semakin intensif penggunaan memori. Hal ini dikarenakan angka-angka yang terbuat juga besar, mengikuti bit yang diatur sehingga komputasi berjalan lebih lama dan berat. Tapi semakin besar bit *pairing*, *signature* yang terbuat semakin aman, jadi ada timbal balik di dalam penggunaan besar bit *pairing*.

Tabel 5.9: Tabel Pengujian Parameter *Byte* Blok Data

No	Bit *Pairing*	Bit Prima	*Byte* Blok Data
1	512	400	6
2	512	400	12
3	512	400	24



Gambar 5.8: Variasi *byte* blok data



Gambar 5.9: Variasi *byte* blok data terhadap penggunaan memori maksimum

Pengujian *byte* blok data dapat dilihat di Tabel 5.9, dengan bit *pairing* dan bit bilangan prima dibuat tetap dan *byte* blok data divariasikan dari 6, 12, dan 24 *byte*. Hasil eksperimen berikutnya ditunjukkan pada Gambar 5.8 menggunakan bit *pairing* 512 dan bit prima 400. *Byte* blok data divariasikan untuk melihat performa tiap blok data yang teruji. Dalam grafik pada Gambar 5.8, terlihat bahwa semakin besar *byte* blok data, semakin cepat komputasi berjalan. Untuk performa maksimum penggunaan memori, terlihat bahwa semakin besar *byte* blok data semakin kecil penggunaan memori. Hal ini dikarenakan ongkos komputasi yang bergantung pada jumlah blok data yang dibentuk, semakin besar *byte* blok data bisa diartikan jumlah blok data dengan ukuran *Byte* yang sama semakin kecil. Dari percobaan tersebut, terlihat bahwa blok data yang besar semakin bagus, namun dalam penerapannya hal ini tidak memungkinkan

karena ukuran bit prima mempengaruhi besar blok data yang bisa dibuat. Semakin kecil bilangan primanya, semakin kecil pula blok data yang bisa digunakan.

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Sistem ini terdiri dalam lima subsistem: *Private Key Generator* (PKG), *User*, *sanitizer*, penyimpanan awan, dan pengaudit (Auditor). Tujuan akhir dari pembuatan sistem ini adalah agar data dapat dibagikan dan diverifikasi tanpa perlu mempertahankan seluruh informasi sensitif yang terdapat pada data yang dibagikan. Audit dapat dilakukan sewaktu-waktu oleh pihak ketiga. Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

- Sistem dapat melakukan sanitasi informasi sensitif pada sebuah data.
  Sanitasi ini dilakukan dengan cara mendefinisikan informasi sensitif dalam bentuk *tags*, kemudian dimasukkan ke dalam pustaka pdf-redactor untuk digantikan. Keluaran berupa *file* PDF yang telah disanitasi dari informasi sensitif.
- Sistem dapat melakukan pengauditan integritas data berbasis identitas terhadap data yang akan disanitasi dari informasi sensitif.
  Rumusan ini diimplementasi pada tahap Pembuatan *Signature* dan Sanitasi. *Signature* dibuat pada tahap Pembuatan *Signature*, kemudian diverifikasi pada tahap Sanitasi untuk diproses lebih lanjut. Bila hasil audit ini tidak valid, maka *Sanitizer* menolak kebenaran *file* dan

menghentikan proses unggahan sepenuhnya. Bila valid, maka *file* akan disanitasi dan kemudian diunggah ke Awan, beserta *signature* barunya.

- Sistem dapat melakukan pengauditan integritas data berbasis identitas terhadap data yang dibagikan di awan. Rumusan ini diimplementasi pada tahap Sanitasi, Pembuatan Bukti, dan Verifikasi Bukti. Di tahap Sanitasi, *file* yang akan diunggah dibuatkan ulang *signature*-nya setelah disanitasi. Kemudian *file* dan *signature*-nya diletakkan di Awan agar bisa diakses. Di tahap Pembuatan Bukti, Auditor membuat tantangan secara acak kepada Awan untuk memberikan bukti yang digunakan Auditor untuk menverifikasi bahwa *file* di Awan masih terjaga. Dan di tahap Verifikasi Bukti, setelah Auditor menerima jawaban dari Awan, Auditor melakukan verifikasi yang kemudian hasilnya dikembalikan kepada Awan untuk ditunjukkan kepada pengguna.

6.2 Saran

Berikut saran dari penulis untuk pengembangan sistem ini:

1. Memberi keamanan tambahan ketika berkomunikasi dari satu subsistem ke subsistem lain supaya tidak mudah dilakukan *sniffing*.
2. Menambahkan fitur *tagging file* supaya mempermudah pengguna mencari *file* yang dibutuhkan.
3. Melakukan optimasi pada sistem.
4. Menerapkan bentuk *file* yang lain seperti DOCX, RTF, dan sebagainya.
5. Membuat konversi blok data yang lebih efisien.

DAFTAR PUSTAKA

[1] W. Shen, J. Qin, J. Yu, R. Hao, dan J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, hal. 331–346, Feb 2019.

[2] "Chapter 3 - an introduction to cryptography," in *Next Generation SSH2 Implementation*, D. Liu, M. Caceres, T. Robichaux, D. V. Forte, E. S. Seagren, D. L. Ganger, B. Smith, W. Jayawickrama, C. Stokes, dan J. Kanclirz, Eds. Burlington: Syngress, 2009, hal. 41 – 64.

[3] "An Overview of Cryptography," 16 Juni 2020. [Daring]. Tersedia pada: https://www.garykessler.net/library/crypto.html. [Diakses: 16 Juni 2020].

[4] A. Menezes, "An Introduction to Pairing-based Cryptography," *Recent Trends in Cryptography Contemporary Mathematics*, hal. 47–65, 2009.

[5] "The GNU MP Bignum Library," 10 Mei 2020. [Daring]. Tersedia pada: https://gmplib.org/. [Diakses: 10 Mei 2020].

[6] B. Lynn, "PBC Library - Pairing-Based Cryptography - About," 10 Mei 2020. [Daring]. Tersedia pada: https://crypto.stanford.edu/pbc/. [Diakses: 10 Mei 2020].

[7] V. M Nieles, K. Dempsey, "An introduction to information security," *NIST Special Publication 800-12 Revision 1*, Jun 2017.

[8] Barker, E. dan Roginsky, A., "Recommendation for cryptographic key generation," in *NIST Special Publication 800-133 Revision 1*, 2019.

[9] “The making of Python,” 11 Desember 2019. [Daring]. Tersedia pada: http://www.artima.com/intv/python.html. [Diakses: 11 Desember 2019].

[10] “General Python FAQ,” 11 Desember 2019. [Daring]. Tersedia pada: https://docs.python.org/3/faq/general.html#what-is-python. [Diakses: 11 Desember 2019].

[11] “Flask | The Pallets Projects,” 10 Mei 2020. [Daring]. Tersedia pada: https://palletsprojects.com/p/flask/. [Diakses: 10 Mei 2020].

[12] “Faq | MongoDB,” 5 Agustus 2019. [Daring]. Tersedia pada: https://www.mongodb.com/faq. [Diakses: 5 Agustus 2019].

[13] “JoshData/pdf-redactor: A general purpose PDF text layer redaction tool for Python 2/3,” 16 Mei 2020. [Daring]. Tersedia pada: https://github.com/JoshData/pdf-redactor. [Diakses: 16 Mei 2020].

[14] “dkloving/pdf-redactor: A general purpose PDF text layer redaction tool for Python 2/3,” 21 Mei 2020. [Daring]. Tersedia pada: https://github.com/dkloving/pdf-redactor. [Diakses: 21 Mei 2020].

[15] B. Lynn, “On the implementation of pairing-based cryptosystems,” Ph.D. dissertation, Stanford University, 2007, https://crypto.stanford.edu/pbc/thesis.pdf.

[16] “Type A internals - PBC,” 6 Juni 2020. [Daring]. Tersedia pada: https://crypto.stanford.edu/pbc/manual/ch08s03.html. [Diakses: 6 Juni 2020].

[17] “Pypi - The Python Package Index,” 21 Mei 2020. [Daring]. Tersedia pada: https://pypi.org/. [Diakses: 21 Mei 2020].

BIODATA PENULIS

Rifqi Ardia Ramadhan, biasa dipanggil Rifqi, lahir pada 14 Januari 1998 di Ponorogo. Penulis adalah mahasiswa yang sedang menjalani studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Selama menempuh studi penulis aktif dalam kompetisi *Capture the Flag*, dan menjadi finalis nasional dalam Kompetisi Komunitas Siber Indonesia 2019 bersama dengan dua rekan lain. Pernah menjadi staff dan staff ahli di dua tahun kepanitiaan Schematics.