Laporan Akhir Proyek Keamanan dan Integritas Data

Implementasi Layanan Kriptografi API untuk Keamanan dan Integritas Data

Oleh Kelompok 2:

Ruthtatia Grace Astridia	24031554072
Tara Tabriza Rachman	24031554107
Bilqis Fadiyah Nisrina	24031554216
Frelin Theresia Pania	24031554220

Dosen pengampu:

Hasanuddin Al-Habib, M.Si.

Moh. Khoridatul Huda, S.Pd., M.Si., Ph.D.

PRODI S1 SAINS DATA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI SURABAYA

2025

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan pesat sistem pertukaran dokumen digital telah menciptakan tantangan signifikan terkait integritas dan otentikasi dari dokumen elektronik. Kerentanan dokumen terhadap modifikasi yang tidak terdeteksi menuntut mekanisme keamanan yang kuat untuk menjamin prinsip non-repudiation. Kemampuan untuk membuktikan bahwa suatu transaksi memang berasal dari pengirim yang sah (Mohammed Q.A.A.S., 2024). Proyek ini mengatasi kerentanan tersebut melalui implementasi Tanda Tangan Digital (Digital Signature) berbasis kriptografi kunci asimetris.

Dalam pengembangan sistem ini, kami memutuskan untuk memilih algoritma Ed25519, yang merupakan bagian dari *Elliptic Curve Cryptography*. Algoritma ini dipilih karena menawarkan kecepatan pembuatan dan verifikasi tanda tangan yang jauh lebih tinggi serta tingkat keamanan yang kuat (Bernstein et al., 2012). Untuk memaksimalkan proses verifikasi dokumen berukuran besar, sistem ini memakai prinsip standar industri Hash-then-Sign. Selain itu, untuk menjamin Secure Session dan autentikasi, kami mengimplementasikan standar JSON Web Token (JWT). Penggunaan JWT memastikan bahwa setiap akses ke *endpoint* pasti memerlukan otorisasi token yang valid, sesuai rekomendasi teknis dari IETF (Jones, 2015).

Kemudian, untuk akuntabilitas tambahan, sistem ini menggunakan Global Activity Logger (sebuah *middleware* pada service API). Fitur ini akan menciptakan Audit Trail, yang akan mencatat setiap *request* pengguna yang terautentikasi maupun anonim. Log ini esensial untuk pemeliharaan sistem, analisis forensik, dan kepatuhan regulasi dengan menyediakan catatan lengkap mengenai siapa melakukan apa dan kapan (Duncan & Whittington, 2017). Semua fitur yang dibangun ini menggunakan struktur kerja FastAPI untuk menjamin integritas, autentikasi, dan akuntabilitas.

1.2 Tujuan

1. Membangun API keamanan berbasis FastAPI yang berfungsi sebagai otoritas pusat untuk registrasi kunci publik, otentikasi, dan layanan *relay* pesan yang aman.
2. Implementasi Autentikasi dan Sesi dengan mengimplementasikan JSON Web Token (JWT), memastikan bahwa hanya pengguna yang terotentikasi dan memiliki token valid yang dapat mengakses *endpoint* /relay dan /verify.
3. Menerapkan algoritma Ed25519 untuk untuk pembuatan dan verifikasi Tanda Tangan Digital, guna menjamin autentikasi pengirim dan integritas pesan/dokumen digital.
4. Mengintegrasikan Global Activity Logger (*middleware*) pada API untuk mencatat setiap aktivitas sistem, sehingga menyediakan Audit Trail yang lengkap.

1.3 Manfaat

1. Menjamin Non-Repudiation dan Integritas Data, penggunaan Tanda Tangan Digital Ed25519 memastikan bahwa pengirim tidak dapat menyangkal pesan atau dokumen yang telah mereka kirim, ini juga memastikan bahwa data tidak diubah selama pengiriman.
2. Memverifikasi Dokumen, Metode Hash-then-sign yang diterapkan pada pesan yang dikirim mampu meningkatkan efisiensi proses verifikasi karena server hanya memproses hash pesan daripada seluruh dokumen berukuran besar.
3. Meningkatan Keamanan Sesi, Penggunaan JWT memberikan standar otorisasi yang aman, yang mengurangi risiko keamanan yang terkait dengan mekanisme sesi tradisional yang biasanya menggunakan *session cookie* yang disimpan di server.
4. Akuntabilitas Sistem yang Tinggi, penggunaan Global Activity Logger memberikan akuntabilitas yang penuh pada sistem, dimana semua request dicatat, baik suskes atau gagal.

BAB II
LANDASAN TEORI

2.1 Application Programming Interface (API)

Application Programming Interface (API) adalah jenis antarmuka pemrograman yang menyediakan fitur, fungsi, dan protokol komunikasi yang memungkinkan aplikasi untuk berinteraksi secara struktural dengan aplikasi atau sistem lain. API bertindak sebagai jembatan antara klien dan server, memungkinkan aplikasi untuk mengakses data atau layanan tanpa mengetahui detail implementasi internal sistem yang diakses (Muri et al., 2019).

Dalam sistem terdistribusi, API memiliki peran penting dalam memfasilitasi komunikasi antar komponen sistem yang beroperasi pada platform atau lingkungan yang berbeda. Melalui API, suatu sistem dapat menyediakan layanan yang dapat diakses oleh sistem lain melalui mekanisme permintaan-respons. Hal ini memungkinkan pemrosesan data dilakukan secara terkontrol, konsisten, dan efisien (Muri et al., 2019).

Penggunaan API memberikan berbagai manfaat dalam pengembangan perangkat lunak, seperti menyederhanakan proses pengembangan aplikasi, meningkatkan interoperabilitas sistem, dan menyederhanakan pengembangan dan pemeliharaan sistem. Selain itu, API dapat dikombinasikan dengan mekanisme keamanan seperti enkripsi dan dekripsi untuk memastikan keamanan data yang diakses..

2.2 Kriptografi Kunci (Asimetris & Simetris)

Kriptografi adalah teknik pengumpulan data yang bertujuan untuk melindungi informasi dari akses pengguna yang tidak sah. Proses kriptografi melibatkan transformasi data plaintext menjadi ciphertext menggunakan algoritma dan kunci tertentu, sehingga data menjadi tidak dapat dibaca tanpa dekripsi yang tepat (Harahap & Salim, 2023).

Berdasarkan kunci yang digunakan, kriptografi dibagi menjadi dua jenis utama, yaitu kriptografi kunci simetris dan kriptografi kunci asimetris. Kedua jenis kriptografi ini memiliki karakteristik yang berbeda dan sering digunakan secara kombinasi untuk meningkatkan keamanan sistem.

2.2.1 Kriptografi Kunci Simetris

Kriptografi kunci simetris adalah metode kriptografi yang menggunakan kunci yang sama untuk enkripsi dan dekripsi data. Oleh karena itu, agar data dapat

dikonversi kembali ke format aslinya, pengirim dan penerima harus memiliki kunci yang dapat diidentifikasi (Widiyastuti et al., 2019).

Advanced Encryption Standard (AES) adalah salah satu algoritma enkripsi yang paling banyak digunakan. Menurut Widiyastuti et al. (2019), AES menawarkan tingkat keamanan dan efisiensi yang tinggi, sehingga sering digunakan dalam pengolahan data sensitif. AES bekerja dengan memproses data melalui beberapa putaran enkripsi yang berbeda dan menghasilkan kunci dengan panjang yang bervariasi.

Widodo & Purnomo (2020) juga menyatakan bahwa AES efektif proses enkripsi dan dekripsi dokumen rahasia karena dapat mendeteksi data rahasia dan relatif cepat dalam pemrosesannya.

Meskipun memiliki keunggulan dalam kecepatan, kriptografi kunci simetris memiliki kelemahan pada proses distribusi kunci. Jika kunci jatuh ke tangan pihak yang tidak berwenang, keamanan data dapat terganggu.

2.2.2 Kriptografi Kunci Asimetris

Kriptografi kunci asimetris adalah metode kriptografi yang menggunakan dua kunci berbeda, yaitu kunci publik dan kunci privat. Kunci publik digunakan untuk proses enkripsi, sedangkan kunci privat digunakan untuk proses dekripsi data (Saputro et al., 2020).

Menurut Saputro et al. (2020), algoritma RSA adalah salah satu algoritma enkripsi publik yang paling sering digunakan. RSA memanfaatkan konsep matematika bilangan prima untuk menghasilkan pasangan kunci, sehingga menghasilkan tingkat keamanan yang sangat tinggi. Data yang didekripsi menggunakan kunci publik hanya dapat didekripsi menggunakan kunci privat yang sesuai.

Keunggulan utama kriptografi kunci asimetris terletak pada keamanan distribusi kunci. Namun, kelemahan metode ini adalah waktu pemrosesannya yang lebih lama dibandingkan dengan kriptografi kunci simetris, sehingga metode ini sering digunakan dalam kombinasi dengan algoritma simetris.

2.3 Digital Signature

Digital signature atau tanda tangan digital adalah mekanisme keamanan yang digunakan untuk memastikan integritas, keaslian, dan non-penolakan dokumen digital. Digital signature memanfaatkan kriptografi kunci asimetris untuk memastikan bahwa data benar-benar berasal dari pihak yang sah dan tidak mengalami perubahan selama proses pengiriman (Winoto, 2023).

Menurut Winoto (2023), proses digital signature dilakukan dengan membuat nilai hash dari pesan, kemudian mengenkripsinya menggunakan kunci privat pengirim. Penerima dapat memverifikasi keabsahan tanda tangan dengan menggunakan kunci publik pengirim. Jika hasil verifikasi sesuai, maka data dianggap valid dan autentik.

Penggunaan digital signature banyak digunakan dalam sistem informasi untuk meningkatkan kepercayaan dan keamanan, khususnya dalam transaksi elektronik dan pemrosesan dokumen digital.

2.4 Enkripsi Data

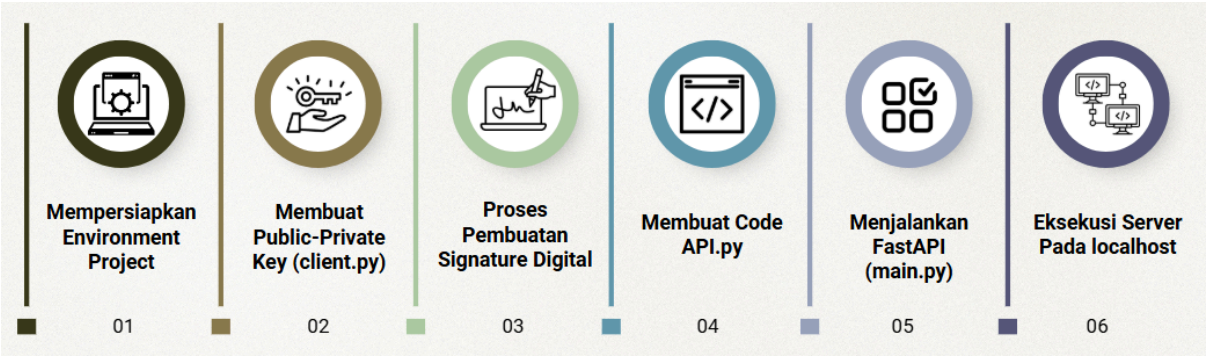
Enkripsi data adalah proses pengamanan informasi dengan cara mengubah data asli menjadi bentuk terenkripsi sehingga pihak yang tidak berwenang tidak dapat membacanya. Enkripsi bertujuan untuk menganalisis data terenkripsi, baik saat disimpan maupun saat ditransmisikan melalui jaringan (Harahap & Salim, 2023).

Menurut Harahap dan Salim (2023) enkripsi data merupakan komponen penting dari keamanan informasi digital. Dengan menggunakan algoritma enkripsi yang tepat, sistem dapat melindungi data sensitif dari manipulasi, dekripsi, dan penyadapan.

Dalam praktiknya, sistem keamanan modern sering menggabungkan enkripsi simetris dan asimetris. Menurut Widiyastuti et al. (2019), enkripsi simetris digunakan untuk menganalisis data primer karena efisiensinya, sedangkan enkripsi asimetris digunakan untuk menganalisis distribusi kunci enkripsi untuk keamanan yang lebih besar.

BAB III
ANALISIS DAN HASIL

3.1 Alur Tahapan yang Dilakukan



Gambar 3.1 Tahapan

Pada project Punk Records-v1 ini terdapat tahapan alur pengerjaan, dimana tahapan-tahapan utama ini sesuai pada gambar 3.1 Tahapan. Beberapa hal yang dilakukan diantaranya:

1. Mempersiapkan Environment Project

   Pada tahapan dilakukan untuk mempersiapkan proses implementasi sistem yang akan dijalankan, yaitu dengan melakukan instalasi seluruh dependensi menggunakan project manager uv. Sehingga segala eksekusi untuk API nanti dapat dilakukan tanpa adanya kendala.
2. Membuat Public-Private Key (client.py)

   Setiap pengguna akan melakukan proses pembuatan *public* dan *private key*, yang dimana proses ini dilakukan pada file client.py. Sehingga pengguna dapat menggunakan key tersebut untuk menandatangani dan membaca data dengan aman.
3. Proses Pembuatan Signature Digital

   Selanjutnya dilakukan tahapan untuk menjamin keaslian pengirim dan integritas data yang dikirimkan, dengan cara pengguna melakukan proses *signature digital* menggunakan *private key* yang telah dimiliki. Dimana proses ini dilakukan pada kalimat dan file pdf, sehingga segala perubahan dalam data tersebut akan dapat diketahui dan dibuktikan.
4. Membuat Code API.py

Tahapan ini dilakukan untuk memastikan layanan API yang dijalankan sebagai penghubung antara pengguna dan sistem Punk Records dapat berjalan dengan baik. Sehingga segala proses yang akan dijalankan pada localhost di rancang terlebih dahulu pada API.py ini. Mulai dari penyimpanan *key*, verifikasi, pengiriman dan sebagainya.

5. Menjalankan FastAPI (main.py)

   Segala hal yang telah dirancang dalam file API.py, maka akan di aktifkan atau dijalankan melalui file main.py. Sehingga setelah server FastAPI sudah dapat berjalan, maka layanan API dapat diakses melalui alamat localhost:8080.

6. Eksekusi Server Pada localhost

   Seluruh fungsi yang telah dibuat pada file API.py, maka dapat diimplementasikan sesuai dengan ketentuannya. Mulai dari melakukan penyimpanan key, pembuatakan token, verifikasi keaslian data, mengirim data pada pengguna lain dan sebagainya. Sehingga melalui tahapan ini dapat dipastikan bahwa sistem sudah dapat digunakan dan diuji sesuai dengan skenario yang telah ditetapkan

3.2 Implementasi Sistem Keamanan

3.2.1 Mempersiapkan Environment Project

Tahapan awal yang dilakukan yaitu dengan mempersiapkan environment untuk menjalankan segala eksekusi pada layanan FastAPI. Sehingga melalui hal ini dapat dipastikan segala hal yang dibutuhkan telah terpasang dengan benar. Adapun dua hal yang harus dilakukan diantaranya:

1. Instalasi UV

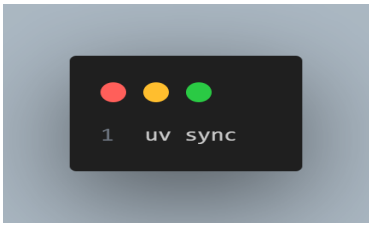
   Hal yang harus dilakukan diantaranya, melakukan instalasi uv sesuai pada gambar 3.2.1 Instalasi yang dilakukan di sistem operasi Windows Powershell. Hal ini dilakukan untuk memasang tools UV pada sistem operasi Windows agar dapat digunakan melalui PowerShell

Gambar 3.2.1.1 Instalasi

Perintah instalasi ini dilakukan dengan menggunakan ExecutionPolicy ByPass, yang dimana hal ini untuk memungkinkan eksekusi yang dilakukan tidak terhambat kebijakan keamanan sistem. Pengunduhan dilakukan menggunakan irm dan dieksekusi langsung dengan iex yang memungkinkan uv dapat berjalan secara otomatis pada sistem Windows.

2. Sinkronisasi Dependensi

   Setelah UV sudah terinstal maka selanjutnya melakukan sinkronisasi dependensi project sesuai dengan gambar 3.2.1.2 Sinkronisasi, hal ini dilakukan dengan membuat *virtual environment* serta menginstal seluruh pustaka Python yang dibutuhkan sesuai dengan konfigurasi pada file project, termasuk pustaka FastAPI. Sehingga ketika menjalankan layanan FastAPI, kode dapat berjalan dengan baik, konsisten dan terkontrol.



Gambar 3.2.1.2 Sinkronisasi

3.2.2 Membuat Public-Private Key (client.py)

Tahap selanjutnya akan dilakukan tahap implementasi dari pembuatan kunci kriptografi yaitu *public key* dan *private key*. Dimana dalam penerapan kriptografi asimetris ini merupakan dasar untuk digunakan ketika akan melakukan *digital signature* serta menjaga keamanan dan integritas data. Pembuatan kunci ini dilakukan sesuai dengan gambar 3.2.2 client.py.

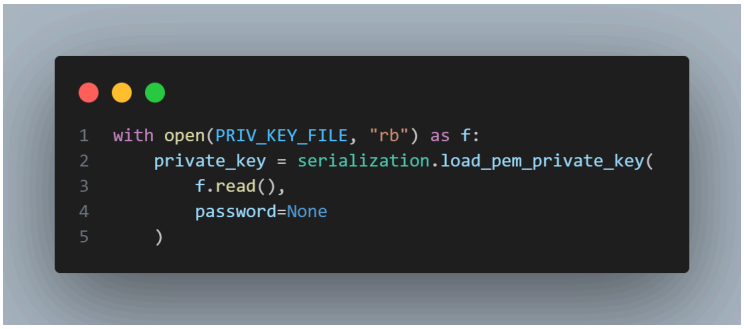
```python
import os
from cryptography.hazmat.primitives.asymmetric import ed25519
from cryptography.hazmat.primitives import serialization

# Nama pemilik key
Nama = "tara"

OUT_DIR = "punkhazard-keys"
os.makedirs(OUT_DIR, exist_ok=True)

# Generate Key Pair
priv = ed25519.Ed25519PrivateKey.generate()
pub  = priv.public_key()

# Simpan Private Key
with open(f"{OUT_DIR}/{Nama}_priv.pem", "wb") as f:
    f.write(
        priv.private_bytes(
            serialization.Encoding.PEM,
            serialization.PrivateFormat.PKCS8,
            serialization.NoEncryption()
        )
    )

# Simpan Public Key
with open(f"{OUT_DIR}/{Nama}_pub.pem", "wb") as f:
    f.write(
        pub.public_bytes(
            serialization.Encoding.PEM,
            serialization.PublicFormat.SubjectPublicKeyInfo
        )
    )

print(f"✓ Ed25519 key pair generated for {Nama}")

```

Gambar 3.2.2 client.py

Hal yang dilakukan pertama yaitu mendefinisikan direktori khusus punkhazard-keys untuk menyimpan pasangan *key* yang telah dibuat. Dimana direktori ini akan dibuat secara otomatis dengan os ketika direktori belum tersedia. Selanjutnya, untuk sistem pembuatan *private key* ini sendiri menggunakan Ed25519PrivateKey.generate() dan dari hasil *private key* yang telah didapatkan maka akan dihasilkan *public key* yang bersesuaian. Kedua hal ini kemudian akan disimpan dengan {Nama}_priv.pem, yang disimpan dengan format PEM dan standar PKCS8 tanpa dilakukan enkripsi tambahan.

3.2.3 Proses Pembuatan Signature Digital

Selanjutnya akan dilakukan proses *digital signature* menggunakan private key yang sebelumnya telah berhasil dibuat, hal ini dilakukan untuk dapat memastikan keaslian pengirim dan integritas data baik dalam pesan teks ataupun file dokumen. Sehingga tahapan awal yang dilakukan pada proses ini adalah memuat private key dari file penyimpanan sesuai dengan gambar 3.2.3 Private Key.

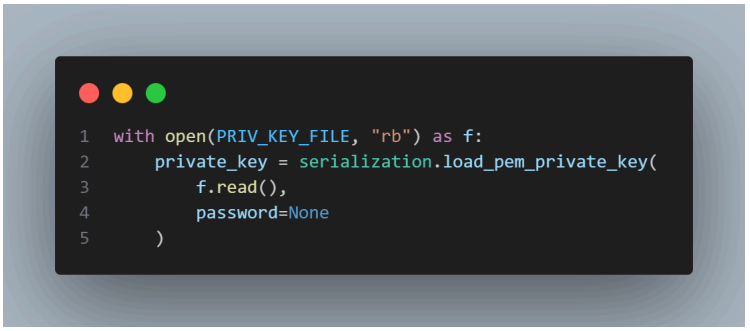


Gambar 3.2.3 Private Key

Adapun beberapa hal yang dilakukan setelah memuat file *private key,* diantaranya:

1. Penandatanganan Pesan Teks

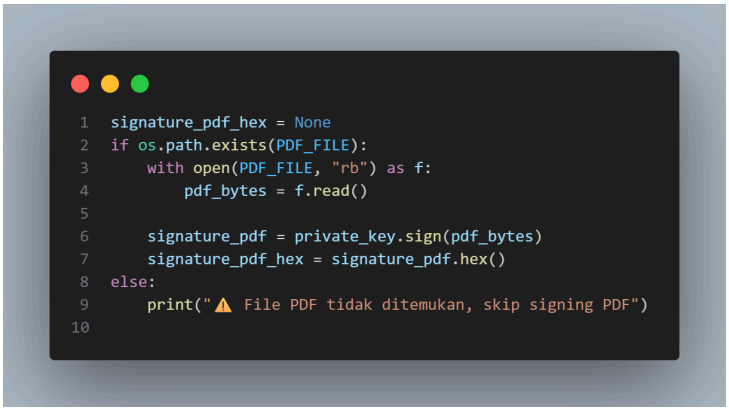
   Proses yang dilakukan pada penandatanganan pesan teks ini sesuai dengan gambar 3.2.3.1 Sign Message. Pesan teks yang telah dibuat dalam MESSAGE_TEXT akan diubah terlebih dahulu ke dalam format byte menggunakan encoding UTF-8, hal ini dilakukan karena algoritma Ed25519 hanya dapat bekerja pada data biner. Selanjutnya akan dilakukan proses tanda tangan digital dengan *private key* dan hasilnya akan dikonversi ke dalam format hexadecimal. Sehingga hasil signature dapat lebih mudah diproses pada layanan API.



Gambar 3.2.3.1 Sign Message

2. Penandatanganan File PDF

   Selain penandatanganan pada pesan teks, sistem juga mendukung untuk dilakukannya proses penandatanganan file dokumen dalam format PDF, dimana proses yang akan dilakukan sesuai dengan gambar 3.2.3.2 Sign PDF.



```
signature_pdf_hex = None
if os.path.exists(PDF_FILE):
    with open(PDF_FILE, "rb") as f:
        pdf_bytes = f.read()

    signature_pdf = private_key.sign(pdf_bytes)
    signature_pdf_hex = signature_pdf.hex()
else:
    print("⚠️ File PDF tidak ditemukan, skip signing PDF")

```

Gambar 3.2.3.2 Sign PDF

Sebelum melakukan proses *digital signature*, sistem akan terlebih dahulu memeriksa keberadaan file PDF pada direktori PDF_FILE. Apabila file PDF ditemukan maka akan dilakukan proses selanjutnya yaitu membaca isi file dalam bentuk byte dan ditandatangani menggunakan *private key*, sehingga file PDF telah dilindungi oleh tanda tangan digital. Ketika file PDF tidak ditemukan maka akan ditampilkan peringatan dan melewati proses penandatanganan dokumen. Hasil dari penandatanganan akan diubah ke dalam format hexadecimal seperti pada *sign message*.

3. Penyimpanan Hasil Digital Signature

Proses penyimpanan ini dilakukan sesuai dengan gambar 3.2.3.3 Penyimpanan Digital Signature, dimana seluruh hasil proses penandatanganan akan disimpan kedalam sebuah file dengan nama {USERNAME}_signed_data.txt. File ini berisi informasi username, pesan teks yang ditandatangani dan hasil *digital signature* dalam format hexadecimal, serta judul dari file PDF yang ditandatangani dan juga hasil *digital signature.* Sehingga dengan melakukan penyimpanan ini akan mempermudah proses dokumentasi dan pengiriman data pada localhost.

```
with open(f"{USERNAME}_signed_data.txt", "w") as f:
    f.write(f"Username: {USERNAME}\n\n")

    f.write("=== SIGNED TEXT ===\n")
    f.write(f"Message: {MESSAGE_TEXT}\n")
    f.write(f"Signature Hex:\n{signature_text_hex}\n\n")

    if signature_pdf_hex:
        f.write("=== SIGNED PDF ===\n")
        f.write(f"PDF File: {PDF_FILE}\n")
        f.write(f"Signature Hex:\n{signature_pdf_hex}\n")
```

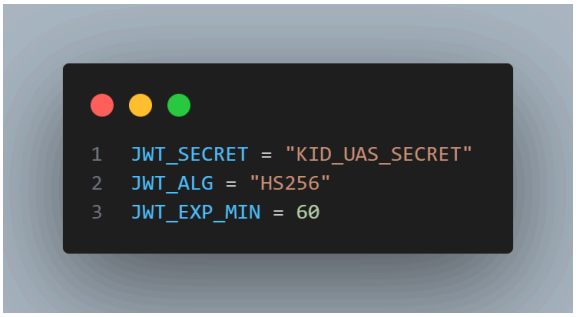
Gambar 3.2.3.3 Penyimpanan Digital Signature

3.2.4 Membuat Code API.py

Pada tahapan ini akan dilakukan proses implementasi untuk membuat layanan backend, yaitu komponen sistem yang berjalan disisi server dengan berbasis FastAPI. Berbagai proses ini akan diatur pada file api.py, beberapa hal yang dilakukannya diantaranya:

1. Konfigurasi Awal dan Manajemen Data

   Sistem diawali dengan pendefinisian konfigurasi dasar sesuai dengan gambar 3.2.4.1 Config, yaitu berisi *secret key*_untuk JSON Web Token (JWT), algoritma enkripsi JWT dan masa berlaku token.

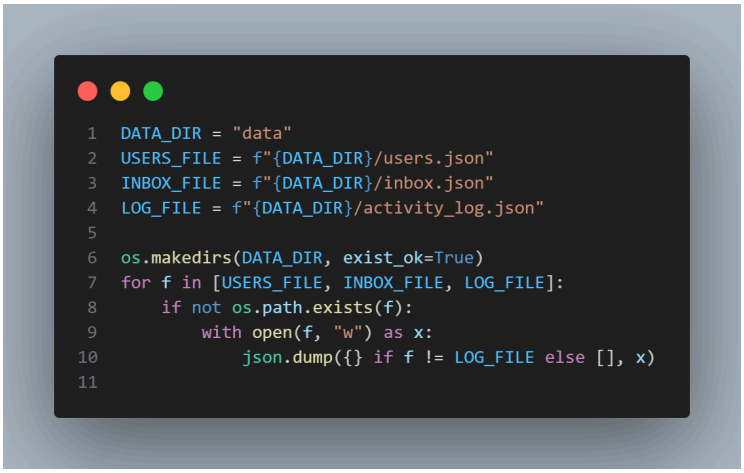


```
JWT_SECRET = "KID_UAS_SECRET"
JWT_ALG = "HS256"
JWT_EXP_MIN = 60
```

Gambar 3.2.4.1.a Config

Dimana JWT_SECRET ini digunakan sebagai *secret key* untuk menandatangani dan memverifikasi token JWT. Sehingga *secret key* yang diterima oleh server benar-benar dibuat oleh sistem itu sendiri dan tidak dimodifikasi. Untuk algoritma kriptografi yang digunakan untuk menandatangani JWT adalah algoritma HS256, yaitu HMAC menggunakan SHA-256. Serta untuk token ini sendiri memiliki masa berlaku selama 60 menit yang tertera pada JWT_EXP_MIN, sehingga melalui

pembatasan waktu ini risiko penyalahgunaan token dapat berkurang dan token tidak jatuh ke pihak lain yang tidak berwenang. Selanjutnya segala proses yang dilakukan dalam sistem akan secara otomatis tersimpan pada direktori DATA_DIR dan sesuai dengan nama file yang telah diatur sesuai dengan gambar 3.2.4.1.b Data. Perintah penyimpanan data ini dilakukan dengan menggunakan os, sehingga ketika direktori ataupun file belum tersedia maka sistem akan membuatnya secara otomatis. Dengan melakukan hal ini maka sistem dapat memiliki struktur penyimpanan data yang konsisten.



```
DATA_DIR = "data"
USERS_FILE = f"{DATA_DIR}/users.json"
INBOX_FILE = f"{DATA_DIR}/inbox.json"
LOG_FILE = f"{DATA_DIR}/activity_log.json"

os.makedirs(DATA_DIR, exist_ok=True)
for f in [USERS_FILE, INBOX_FILE, LOG_FILE]:
    if not os.path.exists(f):
        with open(f, "w") as x:
            json.dump({} if f != LOG_FILE else [], x)

```

Gambar 3.2.4.1.b Data

2. Inisialisasi FastAPI dan Konfigurasi Keamanan

```
app = FastAPI(title="Security Service Kelompok 2", version="1.0")
security = HTTPBearer()
```

Gambar 3.2.4.2.a Keamanan API

Selanjutnya akan dilakukan tahapan sesuai dengan kode pada gambar 3.2.4.2.a Keamanan API. Pada kode ini sistem backend dibangun menggunakan framework FastAPI, dimana instance FastAPI berfungsi sebagai inti layanan backend dan akan memberikan informasi judul dan versi aplikasi yang digunakan. Sehingga informasi ini akan digunakan sebagai identitas dari layanan API dan ditampilkan pada dokumentasi otomatis pada localhost. Selain itu, keamanan HTTP Bearer Token

akan dilakukan inisialisasi untuk mendukung autentikasi menggunakan token JWT pada endpoint yang dilindungi.

Gambar 3.2.4.2.b CORS

Selain itu pada kode ini juga terdapat middleware Cross-Origin Resource Sharing (CORS) sesuai dengan gambar 3.2.4.2.b CORS. Middleware CORS digunakan untuk mengatur izin akses API dari domain berbeda, sehingga dengan allow_origins=[“*”] maka API akan dapat diakses dari semua domain. Sedangkan untuk kode allow_methods dan allow_headers digunakan untuk mengizinkan segala metode HTTP dan header, dengan begitu proses pengujian dan integrasi dapat lebih mudah dilakukan.

3. Autentikasi Pengguna Dengan JWT

Tahapan ini dilakukan untuk mengelola proses autentikasi pengguna dengan menggunakan JSON Web Token (JWT), yang dilakukan melalui dua hal yaitu:

a. Pembuatan Token JWT

Gambar 3.2.4.3.a Autentikasi JWT

Sesuai dengan kode pada gambar 3.2.4.3.a Autentikasi JWT. Dengan fungsi load dan save kedua hal ini digunakan sebagai utilitas untuk membaca dan menyimpan data dalam format JSON, dimana dengan create_token maka akan menghasilkan token JWT setelah pengguna berhasil login menggunakan username yang telah terdaftar. Token ini berisi sub identitas pengguna dan exp batas waktu kadaluarsa token. Dimana token ini akan ditandatangani dengan *secret key* dan algoritma yang telah diinisialisasi sebelumnya untuk memastikan keaslian dan integritas dari token yang diberikan.

b. Validasi Tokenisasi

Gambar 3.2.4.3.b Validasi Token

Fungsi get_current_user yang ada pada gambar 3.2.4.3.b Validasi Token digunakan untuk memvalidasi token JWT yang dikirim melalui header Authorization. Token yang dikirimkan akan didekode menggunakan secret key dan algoritma yang sesuai, sehingga ketika token valid dan tidak kedaluwarsa maka fungsi akan mengembalikan identitas pengguna. Namun jika token tidak valid atau kadaluarsa, maka sistem akan menolak dan memberikan status 401 Unauthorized sehingga endpoint tidak dapat diakses.

4. Audit Log Aktivitas Sistem

```python
@app.middleware("http")
async def logger(req: Request, call_next):
    user = "anonymous"
    auth = req.headers.get("authorization")
    if auth and auth.lower().startswith("bearer"):
        try:
            user = jwt.decode(auth.split()[1], JWT_SECRET, algorithms=[JWT_ALG])["sub"]
        except:
            user = "invalid-token"

    res = await call_next(req)
    logs = load(LOG_FILE)
    logs.append({
        "time": datetime.utcnow().isoformat(),
        "path": req.url.path,
        "method": req.method,
        "user": user,
        "status": res.status_code
    })
    save(LOG_FILE, logs)
    return res
```

Gambar 3.2.4.4 Aktivitas Sistem

Untuk mendukung keamanan dan pelacakan aktivitas, maka disini terdapat kode sesuai dengan gambar 3.2.4.4 Aktivitas Sistem yang berisi kode tentang sistem dan telah dilengkapi dengan middleware audit log. Hal ini dijalankan setiap kali ada permintaan (request) yang masuk ke dalam server, sebelum dan sesudah proses endpoint. Dengan middleware segala permintaan HTTP yang masuk ke server akan di catat, mulai dari waktu akses, endpoint yang diakses, metode HTTP, identitas pengguna dan status respon. Segala aktivitas ini akan disimpan ke dalam alamat LOG_FILE dalam format JSON

5. Endpoint

Tahapan terakhir pada file api.py ini terdapat endpoint yang berisi antarmuka yang akan tersedia pada layanan API, sehingga server akan dapat menerima dan memproses segala permintaan yang masuk. Adapun beberapa hal yang terdapat dalam endpoint ini diantaranya:

a. /health

```python
@app.get("/health")
def health():
    return {"status": "OK", "time": datetime.utcnow().isoformat()}
```

Gambar 3.2.4.5.a Health

Endpoint ini digunakan untuk memastikan bahwa server FastAPI berjalan dengan normal. Kode yang dijalankan sesuai dengan gambar 3.2.4.5.a Health berisi @app.get("/health") yaitu menandakan fungsi yang didefinisikan di bawahnya akan berjalan ketika server menerima permintaan pada path /health. Respon yang dikembalikan berisi status sistem, waktu server dalam UTC saat permintaan diproses. Proses endpoint ini bersifat publik dan tidak diperlukan proses autentikasi.

b. /register

```python
@app.post("/register")
async def register(username: str = Form(...), pubkey: UploadFile = File(...)):
    users = load(USERS_FILE)
    if username in users:
        raise HTTPException(400, "User exists")

    key_bytes = await pubkey.read()
    key = serialization.load_pem_public_key(key_bytes)
    if not isinstance(key, Ed25519PublicKey):
        raise HTTPException(400, "Invalid Ed25519 key")

    users[username] = {"pubkey": key_bytes.decode()}
    save(USERS_FILE, users)
    return {"message": "Registered"}
```

Gambar 3.2.4.5.b Register

Endpoint ini diakses menggunakan metode HTTP POST dan berfungsi untuk mendaftarkan pengguna ke dalam sistem dengan mengirimkan data berupa username (nama pengguna) dan pubkey (file *public key*). Selanjutnya sistem akan mengecek apakah username telah terdaftar di users dan jika sudah ada maka server akan mengembalikan HTTP 400 Bad Request agar tidak terjadi duplikasi akun.

Selain itu *public key* yang telah diunggah akan dibaca dalam bentuk byte dan memastikan apakah *public key* benar-benar bertipe ED25519 dan jika bukan maka registrasi akan ditolak. Namun jika *public key* sesuai, server akan menyimpannya dalam format teks pem ke dalam file users.json agar dapat digunakan untuk proses verifikasi.

c. /token

```python
@app.post("/token")
async def login(username: str = Form(...)):
    users = load(USERS_FILE)
    if username not in users:
        raise HTTPException(401, "User not registered")
    return {"access_token": create_token(username)}
```

Gambar 3.2.4.5.c Token

Sesuai kode pada gambar 3.2.4.5.d Token terdapat endpoint yang digunakan untuk melakukan autentikasi pengguna dan menghasilkan token JWT. Endpoint ini akan menerima satu parameter berupa username, dimana username ini akan divalidasi dengan memuat daftar pengguna pada file users.json. Jika username belum terdaftar maka sistem akan mengembalikan HTTP 401 (Unauthorized) dan jika username valid maka server akan memanggil create_token(username) untuk memberikan token JWT yang dapat disalin pada Authorize. Sehingga pengguna dapat mengakses layanan lanjutan, seperti verify, relay dan inbox.

d. /verify

```python
@app.post("/verify-text")
async def verify_text(
    sender: str = Form(...),
    message: str = Form(...),
    signature_hex: str = Form(...),
    user: str = Depends(get_current_user)
):
    users = load(USERS_FILE)

    if sender not in users:
        raise HTTPException(404, "Sender not registered")

    try:
        signature = bytes.fromhex(signature_hex)
        pubkey = serialization.load_pem_public_key(
            users[sender]["pubkey"].encode()
        )

        pubkey.verify(signature, message.encode())

        return {
            "status": "VALID",
            "signed_by": sender,
            "verified_by": user
        }

    except InvalidSignature:
        raise HTTPException(400, "INVALID SIGNATURE")

    except ValueError:
        raise HTTPException(400, "Signature is not valid hex")
```

```python
@app.post("/verify-pdf")
async def verify_pdf(
    sender: str = Form(...),
    signature_hex: str = Form(...),
    pdf: UploadFile = File(...),
    user: str = Depends(get_current_user)
):
    users = load(USERS_FILE)

    if sender not in users:
        raise HTTPException(404, "Sender not registered")

    try:
        pdf_bytes = await pdf.read()
        signature = bytes.fromhex(signature_hex)

        pubkey = serialization.load_pem_public_key(
            users[sender]["pubkey"].encode()
        )

        pubkey.verify(signature, pdf_bytes)

        return {
            "status": "VALID",
            "signed_by": sender,
            "verified_by": user
        }

    except InvalidSignature:
        raise HTTPException(400, "INVALID SIGNATURE")

    except ValueError:
        raise HTTPException(400, "Signature is not valid hex")
```

Gambar 3.2.4.5.d Verify

Pada endpoint ini digunakan untuk melakukan verifikasi *digital signature* yang memastikan bahwa data yang diterima benar-benar ditandatangani oleh pengirim yang sah dan tidak ada perubahan sejak proses penandatanganan. Dalam implementasinya verifikasi ini dibagi menjadi dua hal yaitu memverifikasi keaslian pesan teks yang telah ditandatangani melalui /verify-text dan memverifikasi *digital signature* pada file PDF melalui /verify-pdf. Kedua endpoint ini memiliki alur proses kode yang serupa yaitu sesuai dengan kode pada gambar 3.2.4.5.d Verify, diantaranya:

1. Autentikasi Akses Layanan

   Kedua endpoint ini dilindungi mekanisme JWT authentication, dimana hanya pengguna yang telah login dan memiliki token JWT saja yang dapat melakukan verifikasi.

2. Validasi Pengirim (Sender)

   Sebelum proses verifikasi dilakukan, sistem akan melakukan pengecekan pengirim atau *sender* pada USER_FILE, sehingga ketika pengirim tidak ditemukan maka proses verifikasi akan dihentikan dan server akan mengembalikan respons kesalahan.

3. Pengambilan Public Key Pengirim

   Setelah pengirim berhasil ditemukan maka selanjutnya server akan mengambil *public key* milik pengirim pada users.json. Dengan kode load_pem_public_key, maka server akan membaca *public key* dengan format pem dan dilakukan proses pubkey.verify untuk memverifikasi kecocokan tanda tangan.

4. Proses Verifikasi Tanda Tangan

   Verifikasi pada pesan teks akan dikonversi terlebih dahulu ke dalam bentuk byte, begitu pula dengan *digital signature* yang telah diberikan. Melalui kedua hal ini, sistem akan memverifikasi kecocokan tanda tangan terhadap isi pesan.

   Sedangkan untuk verifikasi dokumen PDF dilakukan dengan cara membaca isi file PDF dalam bentuk byte, begitu pula untuk *digital signature* nya.

Kemudian dilakukan verifikasi isi file PDF secara keseluruhan dengan *digital signature.*

5. Hasil Verifikasi

   Ketika tanda tangan valid maka sistem akan mengembalikan respon berupa status verifikasi, identitas pengirim data, dan identitas pengguna yang melakukan verifikasi. Namun jika tanda tangan tidak valid atau tidak sesuai, maka sistem akan mengembalikan status verifikasi dan pesan kesalahan yang sesuai.

e. /relay

```python
@app.post("/relay-text")
async def relay_text(
    to: str = Form(...),
    message: str = Form(...),
    signature_hex: str = Form(...),
    sender: str = Depends(get_current_user)
):
    users = load(USERS_FILE)
    pubkey = serialization.load_pem_public_key(users[sender]["pubkey"].encode())
    pubkey.verify(bytes.fromhex(signature_hex), message.encode())

    inbox = load(INBOX_FILE)
    inbox.setdefault(to, []).append({
        "type": "text",
        "from": sender,
        "message": message,
        "signature": signature_hex,
        "time": datetime.utcnow().isoformat()
    })
    save(INBOX_FILE, inbox)
    return {"message": "Text relayed"}
```

```python
@app.post("/relay-pdf")
async def relay_pdf(
    to: str = Form(...),
    signature_hex: str = Form(...),
    pdf: UploadFile = File(...),
    sender: str = Depends(get_current_user)
):
    users = load(USERS_FILE)
    pdf_bytes = await pdf.read()
    pubkey = serialization.load_pem_public_key(users[sender]["pubkey"].encode())
    pubkey.verify(bytes.fromhex(signature_hex), pdf_bytes)

    inbox = load(INBOX_FILE)
    inbox.setdefault(to, []).append({
        "type": "pdf",
        "from": sender,
        "filename": pdf.filename,
        "pdf_base64": base64.b64encode(pdf_bytes).decode(),
        "signature": signature_hex,
        "time": datetime.utcnow().isoformat()
    })
    save(INBOX_FILE, inbox)
    return {"message": "PDF relayed"}
```

Gambar 3.2.4.5.e Relay

Endpoint relay ini digunakan untuk meneruskan data yang telah dilakukan penandatanganan dari satu pengguna ke pengguna lain Namun sebelum data diteruskan, sistem wajib melakukan verifikasi *digital signature* untuk memastikan keaslian dan integritas data. Selain itu pada endpoint relay ini mendukung pengiriman dua jenis data, yaitu pesan teks yang dapat dilakukan pada /relay-text dan dokumen PDF pada /relay-pdf. Kedua relay ini memiliki alur yang sama sesuai dengan gambar 3.2.4.5.e Relay, diantaranya:

1. Autentikasi Pengirim

   Seperti yang dilakukan pada /verify, kedua endpoint /relay ini juga dilindungi dengan JWT *authentication*, dimana hanya pengguna yang telah login dan memiliki token JWT saja yang dapat mengaksesnya.

2. Verifikasi Digital Signature

   Sebelum data diteruskan ke penerima, sistem akan terlebih dahulu memverifikasi signature_hex yang telah dikirim pengirim dan mencocokkannya dengan message atau pdf dan *public key* pengirim. Sehingga ketika *digital signature* yang dikirimkan tidak valid, maka proses relay akan langsung dihentikan dan pesan tidak akan diteruskan.

3. Penyimpanan ke Inbox Penerima

   Setelah tanda tangan dinyatakan valid, maka data akan disimpan ke dalam inbox penerima menggunakan inbox.setdefault(to, []) dan diletakkan dalam INBOX_FILE. Pada relay pesan teks akan disimpan jenis data, identitas pengirim, isi pesan, tanda tangan digital dan waktu pengirim. Sedangkan untuk relay pdf akan disimpan file PDF yang telah dikonversi ke format Base64, metadata file dan tanda tangan digitalnya.

f. /inbox

Gambar 3.2.4.5.f Inbox

Endpoint ini digunakan untuk menampilkan semua pesan dan dokumen yang telah diterima pengguna yang telah berhasil melakukan login dan *authorize*. Pada endpoint ini bersifat *read-only*, sehingga pengguna hanya dapat melihat dan mengambil data. Dimana inbox akan melakukan *load* pada INBOX_FILE dan menghasilkan struktur data berupa dictionary, sesuai dengan kode pada gambar 3.2.4.5.f Inbox.

3.2.5 Menjalankan FastAPI (main.py)

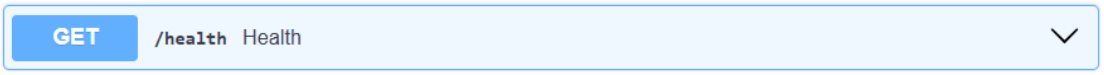
Gambar 3.2.5 main.py

File main.py ini berfungsi sebagai entry point yang menjalankan segala proses dalam file api.py, sehingga server akan dapat dijalankan menggunakan Uvicorn untuk aplikasi FastAPI. Sesuai dengan kode pada gambar 3.2.5 main.py yang menunjukkan bahwa def main() digunakan untuk mengkonfigurasi dan menjalankan server FastAPI. api:app digunakan untuk menunjukkan bahwa aplikasi FastAPI ini berada pada file api.py dan server ini dapat diakses dari alamat jaringan host:”0.0.0.0” sesuai dengan port. Kode terakhir if memastikan bahwa program dapat dieksekusi secara langsung. File main.py ini dapat di run pada powershell dengan kode uv run main.py.

3.2.6 Eksekusi Server Pada localhost

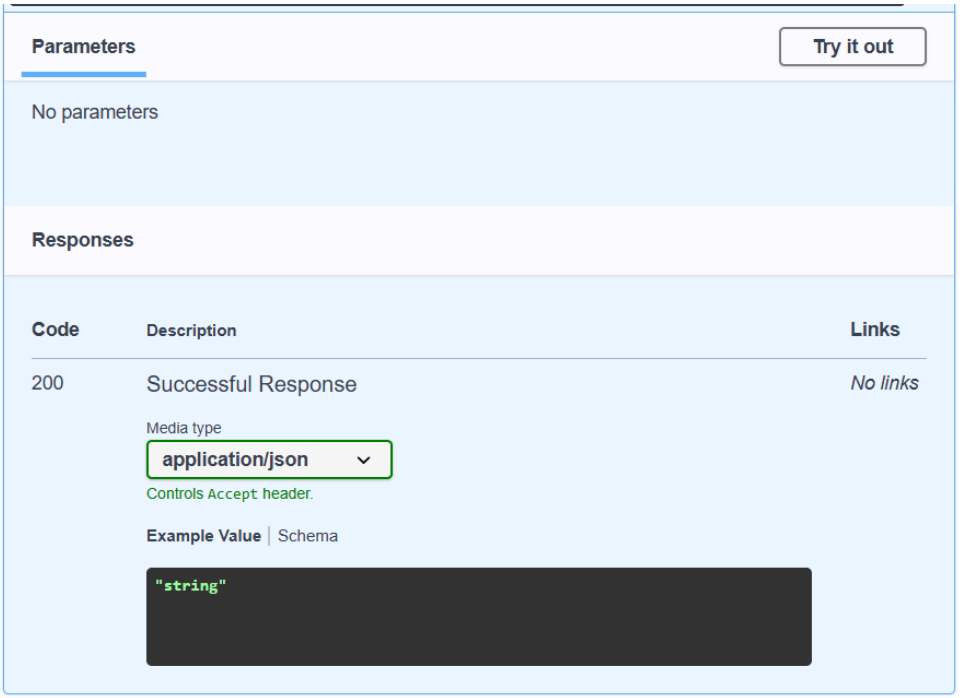
Pada tahapan ini, aplikasi layanan kriptografi Punk Records-v1 dapat dijalankan secara lokal dengan menggunakan komputer pengembang (localhost). Proses ini dilakukan untuk memastikan bahwa seluruh endpoint pada file api.py dapat berfungsi dan digunakan sepenuhnya. Aplikasi ini dapat diakses pada browser melalui http://localhost:8080 sesuai dengan alamat pada main.py. Tampilan endpoint yang telah diproses sebelumnya yaitu:

1. /health



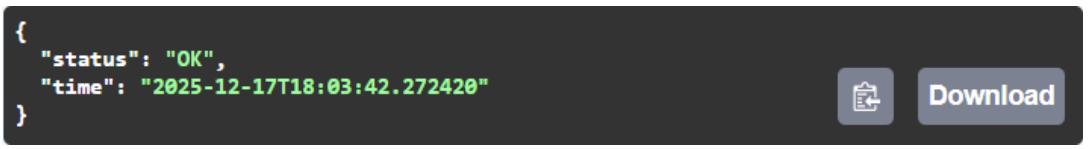
Gambar 3.2.6.1.a Tampilan Health

Tampilan awal endpoint /health pada localhost yaitu sesuai dengan gambar 3.2.6.1.a Tampilan Health, dimana ketika *dropdown arrow* di klik maka akan memunculkan tampilan seperti pada gambar 3.2.6.1.b Health



Gambar 3.2.6.1.b Health

Dimana ketika kita meng-klik Try it out maka akan memunculkan tulisan execute dan pada localhost akan ditampilkan status sistem dan waktu melakukan execute health seperti pada gambar 3.2.6.1.c Hasil Health.



Gambar 3.2.6.1.c Hasil Health

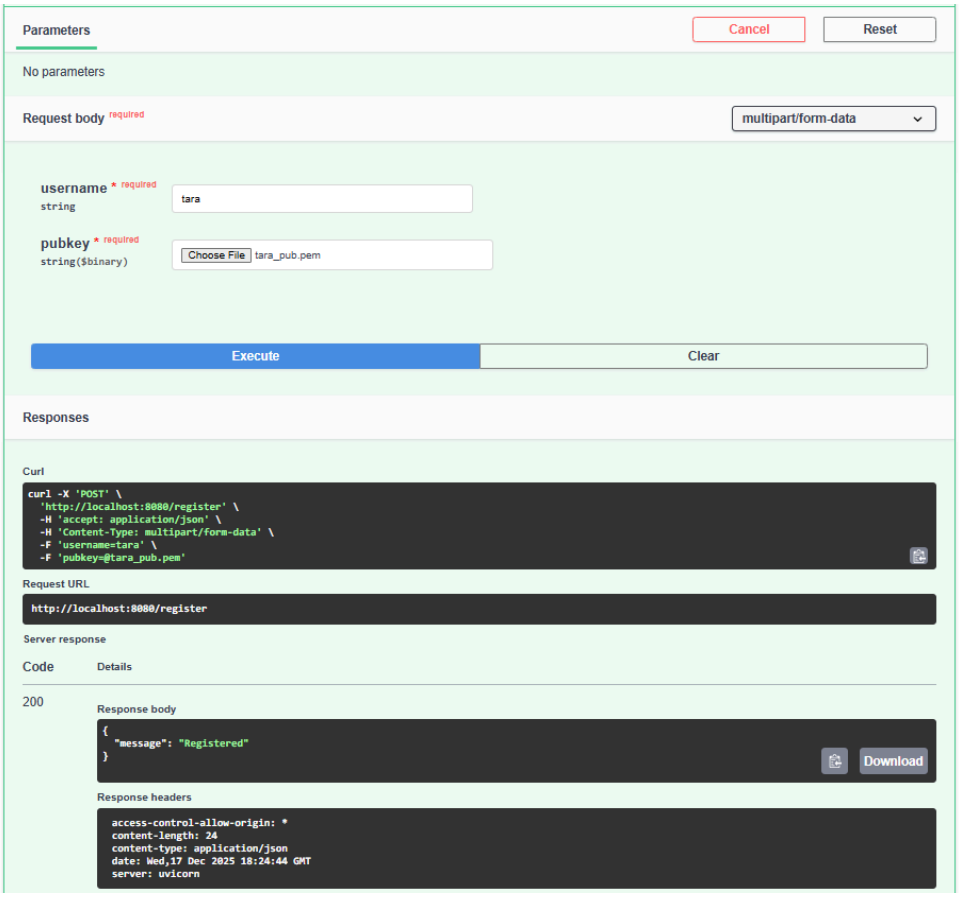
Kemudian hasil dari aktivitas yang dilakukan ini akan dicatat dan masuk ke dalam activity_log.json. Pada json ini terdapat keterangan waktu, path yang dijalankan, method path nya, user dan status yang sesuai dengan gambar 3.2.6.1.d Activity JSON.

Gambar 3.2.6.1.d Activity JSON

2. /register

Gambar 3.2.6.2.a Tampilan Register

Pada path /register ini, tiap anggota dapat melakukan seperti pada path /health yaitu mengklik Try it out untuk lanjut ke tahap registrasi dengan memasukkan username dan file public key sesuai dengan gambar 3.2.6.2.b Registrasi Pengguna. Ketika username telah dipastikan tidak ada dalam users.json, maka akan ditampilkan message registered atau username telah berhasil disimpan. Sehingga hasil yang telah di execute dapat disimpan dalam users.json dan melalui path register ini maka pengguna dapat lanjut untuk tahap-tahap selanjutnya.

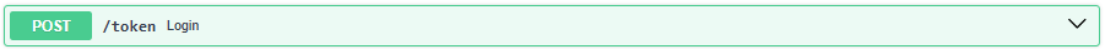


Gambar 3.2.6.2.b Registrasi Pengguna

Hasil registrasi ini akan disimpan ke dalam list users.json sesuai dengan gambar 3.2.6.2.c List Pengguna dan juga aktivitas dilakukannya registrasi ini akan dicatat pada activity_log.json seperti yang dilakukan pada path /health sebelumnya.

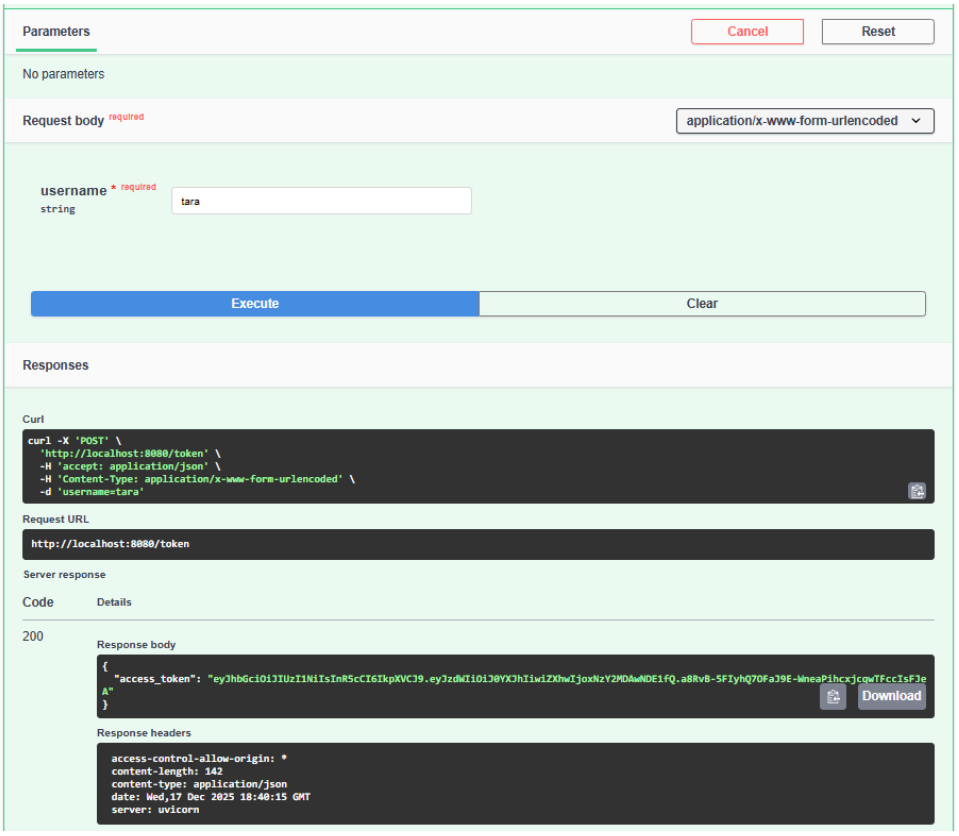
Gambar 3.2.6.2.c List Pengguna

3. /token



Gambar 3.2.6.3.a Tampilan Login

Path /token ini digunakan untuk membuat token JWT dengan memasukkan username yang telah terdaftar pada register dan disimpan pada users.json. Sehingga ketika username telah terdaftar, maka path ini akan otomatis memberikan message berupa access_token atau token yang dapat digunakan sesuai pada gambar 3.2.6.3.b Hasil Token dan token dapat dimasukkan pada Authorize. Aktivitas login ini juga akan dicatat otomatis dalam activity_log.json seperti pada path /health dan /register.



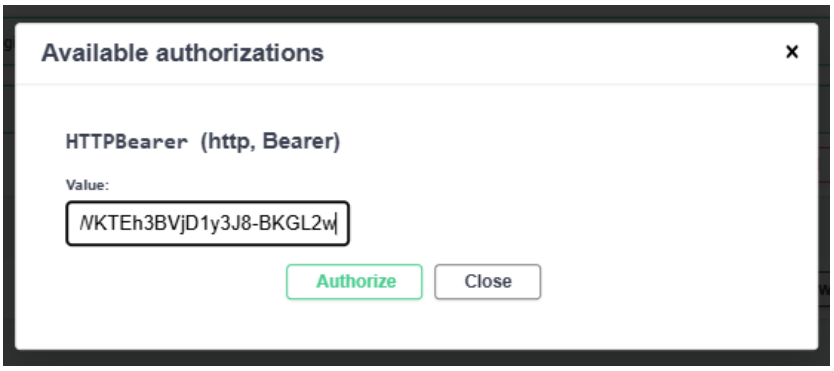
Gambar 3.2.6.3.b Hasil Token

4. Authorize

Gambar 3.2.6.4.a Tampilan Authorize

Authorize terdapat pada pojok kanan atas tampilan localhost sesuai pada gambar 3.2.6.4.a Tampilan Authorize, dimana hal ini digunakan untuk menentukan apakah pengguna sudah login dan mendapatkan token yang sesuai. Sehingga pada authorize ini pengguna diarahkan untuk melakukan input token pada Value sesuai dengan gambar 3.2.6.4.b Input Token yang telah didapatkan pada path /token. Ketika

pengguna sudah berhasil dipastikan identitasnya, maka pengguna dapat melanjutkan ke path selanjutnya.

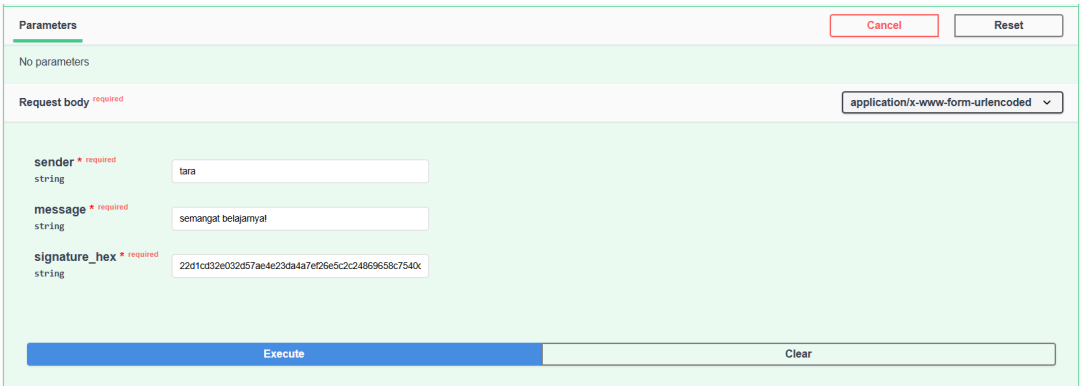


Gambar 3.2.6.4.b Input Token

5. /verify-text

Gambar 3.2.6.5.a Tampilan Verify Text

Endpoint ini digunakan untuk memverifikasi keaslian pesan teks menggunakan *digital signature*, dimana verifikasi ini dilakukan dengan mengambil *public key* pengirim yang telah terdaftar dan tersimpan pada users.json. Sehingga beberapa hal yang harus dimasukkan diantaranya ada sender, message atau pesan asli, signature hex atau *digital signature* dan user sesuai pada gambar 3.2.6.5.b Input Verifikasi Message dan segala aktivitas verify ini juga akan tercatat pada activity_log.json.



Gambar 3.2.6.5.b Input Verifikasi Message

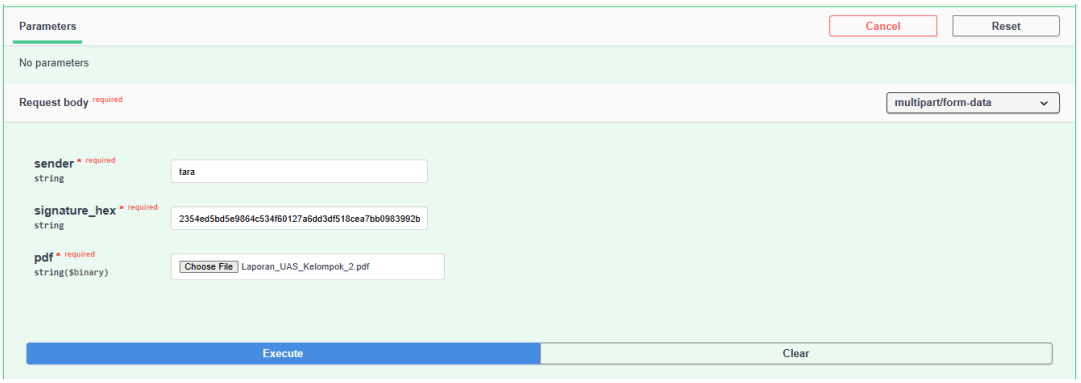
Jika sender telah terdaftar pada users.json dan *digital signature* yang dikirimkannya sesuai ketika di verifikasi dengan message dan *public key* nya, maka ketika execute di klik akan memunculkan message status, signed_by dan verified_by seperti pada gambar 3.2.6.5.c Hasil Verify Message

Gambar 3.2.6.5.c Hasil Verify Message

6. /verify-pdf

Gambar 3.2.6.6.a Tampilan Verify PDF

Seperti pada endpoint verify-text, untuk verifikasi PDF ini juga melakukan hal yang sama dan juga segala aktivitas pada verify ini juga akan tercatat pada activity_log.json. Namun dengan perbedaan hal yang dikirimkan, jika pada verifikasi ini dilakukan input data berupa file dari dokumen yang akan diverifikasi. Sehingga hal-hal yang harus di input diantaranya ada sender, signature_hex atau *digital signature* nya dan file PDF yang akan diverifikasi sesuai pada gambar 3.2.6.6.b Input Verifikasi PDF



Gambar 3.2.6.6.b Input Verifikasi PDF

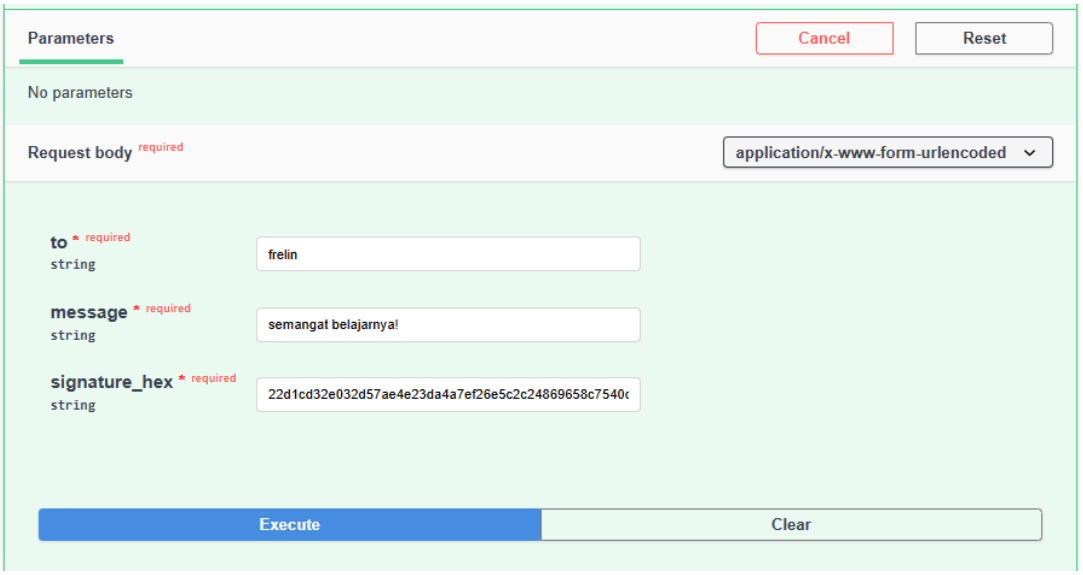
Seperti halnya pada verifikasi message, maka pada verifikasi file PDF ini juga akan mengeluarkan message status, signed_by dan verified_by ketika *digital signature* sudah terbukti sesuai saat diverifikasi dengan *public key* pengirim dan isi dari file PDF tersebut seperti yang terjadi pada gambar 3.2.6.6.c Hasil Verify PDF.

Gambar 3.2.6.6.c Hasil Verify PDF

7. /relay-text

Gambar 3.2.6.7.a Tampilan Relay Text

Endpoint /relay-text ini hanya dapat dilakukan oleh pengirim yang terautentikasi (JWT) dan memiliki fungsi sebagai pengirim pesan teks dari satu pengguna ke pengguna lain, namun dengan syarat pengguna telah terdaftar pada sistem atau tersimpan di users.json. Mekanisme pengiriman ini dilakukan dengan menginput pengguna yang akan dikirim pesan (to), pesan aslinya (message) dan *digital signature* (signature_hex) dari message tersebut, sesuai dengan gambar 3.2.6.7.b Input Relay Text. Selain itu aktivitas yang dilakukan pada relay ini juga akan tercatat pada activity_log.json dengan menyertakan user yang melakukan aktivitas.



Gambar 3.2.6.7.b Input Relay Text

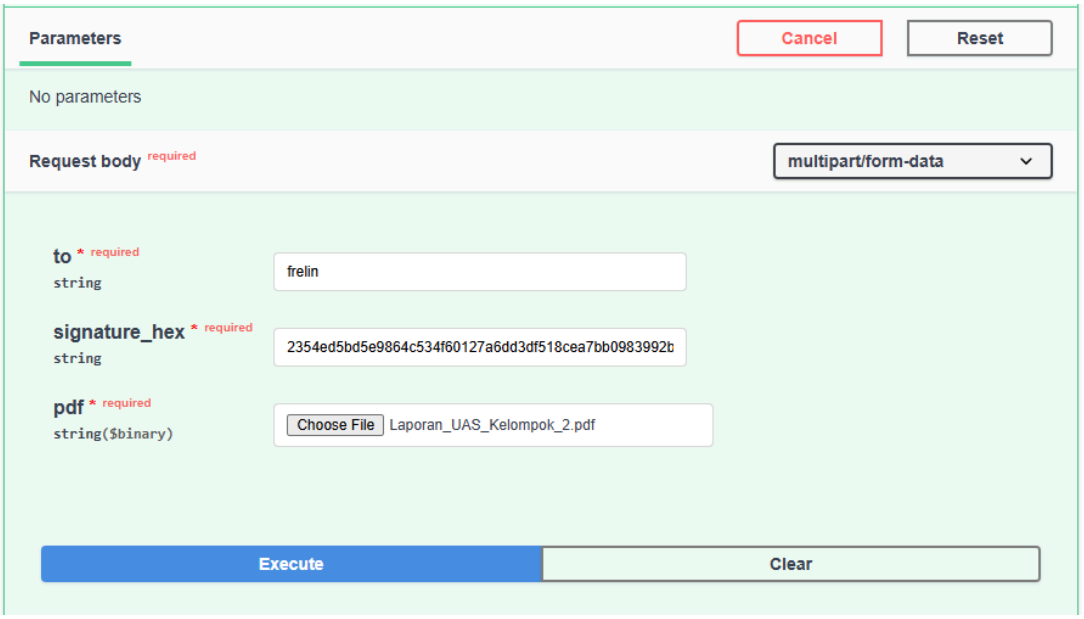
Ketika keaslian pesan telah berhasil terverifikasi dengan menyesuaikan pada tanda tangan digital yang dikirim dan *public key* pengirim maka pesan akan otomatis memberikan message berupa text relayed atau teks berhasil terkirim, seperti pada gambar 3.2.6.7.c Hasil Relay Text. Selain itu pesan yang telah terverifikasi akan tersimpan pada inbox penerima.

Gambar 3.2.6.7.c Hasil Relay Text

8. /relay-pdf

Gambar 3.2.6.8.a Tampilan Relay PDF

Endpoint ini juga memiliki fungsi yang sama seperti relay-text dan aktivitas relay ini juga akan tercatat dalam activity_log.json, namun bedanya pada relay ini pesan yang akan dikirim dalam bentuk file PDF bukan pesan langsungnya. Sehingga pengirim disini akan melakukan input username penerima (to), *digital signature* (signature_hex) dari file PDF yang akan dikirimkan dan file PDFnya, hal ini sesuai pada gambar 3.2.6.8.b Input Relay PDF.

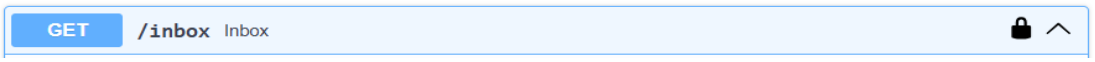


Gambar 3.2.6.8.b Input Relay PDF.

Ketika *digital signature* atau signature_hex telah berhasil terverifikasi dengan *public key* pengirim dan isi pesan dari file PDF tersebut, maka saat klik tombol execute sistem akan mengirimkan pesan PDF relay atau file telah berhasil dikirim sesuai dengan gambar 3.2.6.8.c Hasil Relay PDF. Selain itu, pesan atau file PDF yang dikirimkan ini akan langsung tersimpan ke dalam inbox penerima.

Gambar 3.2.6.8.c Hasil Relay PDF

9. /inbox



Gambar 3.2.6.9.a Tampilan Inbox

Endpoint /inbox ini memiliki kegunaan untuk menampilkan seluruh pesan yang telah dikirimkan pengguna lain, dimana endpoint ini akan dapat terlihat ketika pengguna telah berhasil login, memiliki token dan authorize. Hasil yang ditampilkan berupa hasil relay pesan teks ataupun file PDF yang telah terverifikasi *digital signature* nya, hal ini sesuai dengan tampilan pada gambar 3.2.6.9.b Hasil Inbox. Selain itu aktivitas inbox ini juga akan tercatat pada activity_log.json dengan menyertakan username pengguna yang melakukan aktivitas.

Gambar 3.2.6.9.b Hasil Inbox

Selain itu respon inbox ini juga akan tersimpan pada file inbox.json, dimana pada file tersebut akan berisi username pemilik inbox dan semua list pesan yang sama seperti tampilan hasil inbox pada localhost. Namun bedanya list pesan yang berhasil dikirim ini akan tercatat dalam format JSON seperti pada gambar 3.2.6.9.c inbox.json.

Gambar 3.2.6.9.c inbox.json

BAB IV
PENUTUPAN

4.1 Kesimpulan

Pada proyek ini, hasil perancangan dan implementasi menunjukkan bahwa sistem keamanan berbasis FastAPI dibangun dengan baik. API dirancang sebagai pusat layanan keamanan dalam sistem komunikasi. Selain itu, API juga dirancang untuk mengelola sejumlah fungsi penting, seperti pengiriman pesan aman antar pengguna, proses autentikasi pengguna melalui login, dan pendaftaran kunci publik pengguna.

Implementasi Autentikasi dan Sesi dengan mengimplementasikan JSON Web Token (JWT) telah berhasil diterapkan. Mekanisme ini memberikan sebuah token kepada pengguna yang melakukan login sebagai bukti autentikasi. Selanjutnya, token ini digunakan untuk setiap permintaan ke endpoint penting seperti /relay dan /verify. Server memverifikasi validitas token sebelum memproses permintaan, sehingga hanya pengguna yang telah terotentikasi dan memiliki token yang sah yang dapat mengakses layanan tersebut. Ini menunjukkan bahwa sistem dapat menjaga keamanan layanan API dan mengontrol akses pengguna dengan baik.

Algoritma Ed25519 menggunakan kunci publik untuk membuat digital signature dan kunci privat pengguna untuk memastikan keaslian pengirim dan integritas pesan. Global Activity Logger berhasil diintegrasikan ke dalam API dalam bentuk middleware dan berfungsi untuk mencatat setiap permintaan dan respons yang melewati sistem. Middleware ini mencatat informasi penting seperti waktu akses, endpoint yang diakses, dan status permintaan. Ini memungkinkan rekam seluruh aktivitas sistem secara terpusat. Dengan mekanisme ini, sistem dapat menyediakan audit trail lengkap untuk keperluan evaluasi keamanan dan pemantauan aktivitas pengguna.

4.2 Saran

Pertama, mekanisme autentikasi dapat dikembangkan lebih lanjut dengan menambahkan fitur pembaruan token (refresh token) dan pembatasan jumlah percobaan login, guna meningkatkan keamanan terhadap serangan brute force dan penyalahgunaan akun.

Kedua, Sistem pencatatan aktivitas (Global Activity Logger) dapat dikembangkan dengan menyimpan data log ke dalam basis data serta menambahkan fitur untuk melihat dan

menganalisis log tersebut. Dengan adanya fitur ini, proses pemantauan aktivitas sistem dan evaluasi keamanan dapat dilakukan dengan lebih mudah dan jelas.

DAFTAR PUSTAKA

Bernstein, D. J., Duif, N., Lange, T., Schwab, P., & Yang, B. (2012). *High-speed high-security signatures*. [Paper Teknis]. Retrieved from https://ed25519.cr.yp.to/ed25519-20110926.pdf

Duncan, B., & Whittington, M. (2017). Creating an immutable database for secure cloud audit trail and system logging. *IARIA Cloud Computing 2017*, *8*, 30–35. Retrieved from https://personales.upv.es/thinkmind/dl/conferences/cloudcomputing/cloud_computing_2017/cloud_computing_2017_3_30_28009.pdf

Harahap, A. R., & Salim, T. A. (2023). Penerapan enkripsi data untuk keamanan informasi digital. *Khazanah Informatika,* *7*(1), 33–41. Retrieved from https://journal.ugm.ac.id/khazanah/article/view/81893

Jones, M., Bradley, J., & New, E. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force (IETF). Retrieved from https://datatracker.ietf.org/doc/html/rfc7519

Mohammed Q. A. A. S. (2024). A survey on digital signature schemes. *AIP Conference Proceedings*, *3232*(1). Retrieved from https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/5.0236576/20206190/020057_1_5.0236576.pdf

Muri, M. F. A., Utomo, H. S., & Sayyidati, R. (2019). Pengembangan application programming interface pada sistem terdistribusi. *Jurnal Sains dan Informatika, 6*(2), 45–52. Retrieved from https://jsi.politala.ac.id/index.php/JSI/article/view/175/102

Saputro, T. H., Hidayati, N., & Ujianto, E. I. H. (2020). Implementasi kriptografi kunci publik menggunakan algoritma RSA. *Jurnal Informatika Polinema, 5*(3), 210–218. Retrieved from https://jurnal.polinema.ac.id/index.php/jip/article/view/2558

Widiyastuti, S., Ariandi, W., & Sulistiono, V. (2019). Implementasi kriptografi AES dalam pengamanan data seleksi peserta JAMKESMAS. *Jurnal Ilmiah Intech: Information Technology Journal of UMUS,* *1*(2), 13–22. Retrieved from https://jurnal.umus.ac.id/index.php/intech/article/view/66

Widodo, B. E., & Purnomo, A. S. (2020). Implementasi advanced encryption standard pada enkripsi dan dekripsi dokumen rahasia. *Jurnal Teknik Informatika (JUTIF), 1*(2), 69–77. Retrieved from https://jutif.if.unsoed.ac.id/index.php/jurnal/article/view/21

Winoto, P. H. M. (2023). Penggunaan digital signature sebagai keamanan sistem informasi. *Jurnal Sistem Informasi, 14*(2), 98–107. Retrieved from https://www.researchgate.net/publication/370097835

Lampiran

Link GitHub: https://github.com/Tatya17/Api-Keamanan-Kriptografi