LAPORAN PROYEK AKHIR MATA KULIAH KEAMANAN DAN INTEGRITAS DATA

IMPLEMENTASI KRIPTOGRAFI MENGGUNAKAN AES-256, ED25519, SHA-256, & HS256 PADA LAYANAN *PUNK RECORDS* BERBASIS *FASTAPI*

Disusun Oleh:

1. Cantika Latifatul N. E. (24031554023)
2. Sofia Dwi Kinasih (24031554079)
3. Ilmin Nur Lailiyah (24031554135)

Dosen Pengampu:

Hasanuddin Al-Habib, M.Si.
Moh. Khoridatul Huda, S.Pd., M.Si., Ph.D.

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

SEMESTER GASAL
2025/2026

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pemanfaatan *Application Programming Interface* (API) dalam sistem informasi modern terus meningkat, terutama pada layanan berbasis web yang melibatkan pertukaran data antar pengguna secara *real time*. Namun, layanan API memiliki risiko keamanan yang tinggi apabila tidak dilengkapi mekanisme keamanan dan perlindungan data yang memadai, seperti ancaman penyadapan, manipulasi pesan, dan pemalsuan identitas pengguna. *REST API* tanpa lapisan kriptografi yang kuat sangat rentan terhadap serangan *man-in-the-middle* (Shofyan dan Shita, 2024). Algoritma kriptografi simetris *Advanced Encryption Standard* (AES) dengan panjang kunci 256 bit merupakan salah satu algoritma yang direkomendasikan untuk pengamanan data karena memiliki tingkat keamanan tinggi dan efisiensi komputasi yang baik. AES-256 efektif digunakan untuk melindungi data sensitif dalam sistem berbasis web dan server (Setiadi et al., 2025). Dalam konteks layanan API, AES-256 berperan penting dalam menjaga kerahasiaan pesan yang dikirimkan antar pengguna. Penggunaan enkripsi ini mencegah pihak tidak berwenang membaca isi data meskipun berhasil melakukan pemotongan jaringan.

Selain kerahasiaan, aspek integritas dan autentikasi token juga harus dijaga untuk memastikan pesan tidak mengalami perubahan selama proses transmisi serta pengguna yang mengakses API adalah pengguna yang sah. Algoritma hash SHA-256 banyak digunakan dalam penelitian keamanan sistem informasi karena mampu mendeteksi perubahan sekecil apa pun pada data (Setiadi et al., 2025). Penggunaan SHA-256 memungkinkan sistem untuk melakukan pemeriksaan keutuhan pesan sebelum diproses lebih lanjut oleh penerima. HS256 (HMAC-SHA256) merupakan algoritma yang banyak digunakan dalam sistem autentikasi berbasis JWT *(JSON Web Token)* karena menggabungkan fungsi hashing SHA256 dengan mekanisme *Message Authentication Code* (MAC) menggunakan secret key simetris (Setiadi et al., 2025). Penggunaan HS256 memungkinkan sistem untuk membuat dan memverifikasi token autentikasi yang tidak dapat dipalsukan tanpa mengetahui *secret key*, sehingga meningkatkan kepercayaan terhadap identitas pengguna yang mengakses API. Di sisi lain, autentikasi pesan dapat diperkuat melalui tanda tangan digital berbasis *elliptic curve cryptography,* seperti Ed25519, yang dikenal memiliki performa tinggi dan tingkat keamanan yang kuat. Penelitian tentang sistem API modern menunjukkan bahwa penggunaan tanda

tangan digital meningkatkan kepercayaan terhadap keaslian sumber pesan serta mencegah pemalsuan identitas pengguna (Hendra et al., 2025).

Berdasarkan kebutuhan tersebut, layanan *Punk Records-v1* dirancang untuk menerapkan kriptografi menggunakan AES-256 untuk enkripsi data pada sisi client, HS256 untuk autentikasi token pengguna, SHA-256 untuk integritas file, dan Ed25519 untuk verifikasi tanda tangan digital pada layanan *REST API* berbasis *FastAPI.* Sistem ini berfungsi untuk menyimpan *public key* peneliti, memverifikasi tanda tangan digital, serta melakukan relay pesan secara aman antar pengguna. Pengamanan API dengan kombinasi enkripsi, autentikasi token, dan tanda tangan digital mampu meningkatkan keamanan sistem secara signifikan (Prasetya dan Suharjo, 2025). Dengan penerapan kriptografi tersebut, *Punk Records-v1* diharapkan mampu menjaga kerahasiaan, integritas, dan keaslian data dalam lingkungan laboratorium Vegapunk yang bersifat sensitif dan kolaboratif.

1.2 Rumusan Masalah

1. Bagaimana merancang dan mengimplementasikan layanan keamanan berbasis API yang menerapkan mekanisme kriptografi *public key* untuk menjamin keaslian *(authentication*) dan integritas data *(integrity*), khususnya dalam proses pengiriman pesan dan unggah dokumen PDF?
2. Bagaimana penerapan tanda tangan digital *(digital signature)* menggunakan algoritma kriptografi *public key* dapat digunakan untuk memverifikasi bahwa data atau dokumen yang diterima tidak mengalami perubahan selama proses transmisi?
3. Bagaimana mekanisme autentikasi berbasis *JSON Web Token* (JWT) diterapkan pada layanan API untuk memastikan bahwa setiap permintaan hanya dapat diakses oleh pengguna yang sah?

1.3 Tujuan

1. Untuk mengetahui bagaimana merancang dan mengimplementasikan layanan keamanan berbasis API yang menerapkan mekanisme kriptografi *public key* untuk menjamin keaslian dan integritas data, khususnya dalam proses pengiriman pesan dan unggah dokumen PDF.
2. Untuk mengetahui bagaimana penerapan tanda tangan digital menggunakan algoritma kriptografi *public key* dapat digunakan untuk memverifikasi bahwa data atau dokumen yang diterima tidak mengalami perubahan selama proses transmisi.

3. Untuk mengetahui bagaimana mekanisme autentikasi berbasis JWT diterapkan pada layanan API untuk memastikan bahwa setiap permintaan hanya dapat diakses oleh pengguna yang sah.

1.4 Manfaat

1. Manfaat Teoritis

   Laporan ini diharapkan dapat menambah wawasan dan pemahaman mengenai penerapan konsep kriptografi *public key,* tanda tangan digital, serta mekanisme autentikasi berbasis JWT dalam pengembangan layanan keamanan berbasis API.

2. Manfaat Praktis

   Secara praktis, laporan ini memberikan gambaran implementasi layanan keamanan digital berbasis API yang mampu menjamin keaslian dan integritas data, khususnya dalam proses pengiriman pesan dan unggah dokumen PDF, sehingga dapat dijadikan sebagai referensi dalam pengembangan sistem keamanan pada aplikasi berbasis web atau layanan terdistribusi.

3. Manfaat Akademis

   Laporan ini dapat digunakan sebagai bahan referensi atau acuan pembelajaran bagi mahasiswa atau pihak lain yang mempelajari keamanan sistem informasi, khususnya terkait penerapan kriptografi *public key, digital signature,* dan autentikasi JWT dalam sistem berbasis API.

BAB II
PEMBUATAN DAN PENGUJIAN SERVER

2.1 Gambaran Umum Server

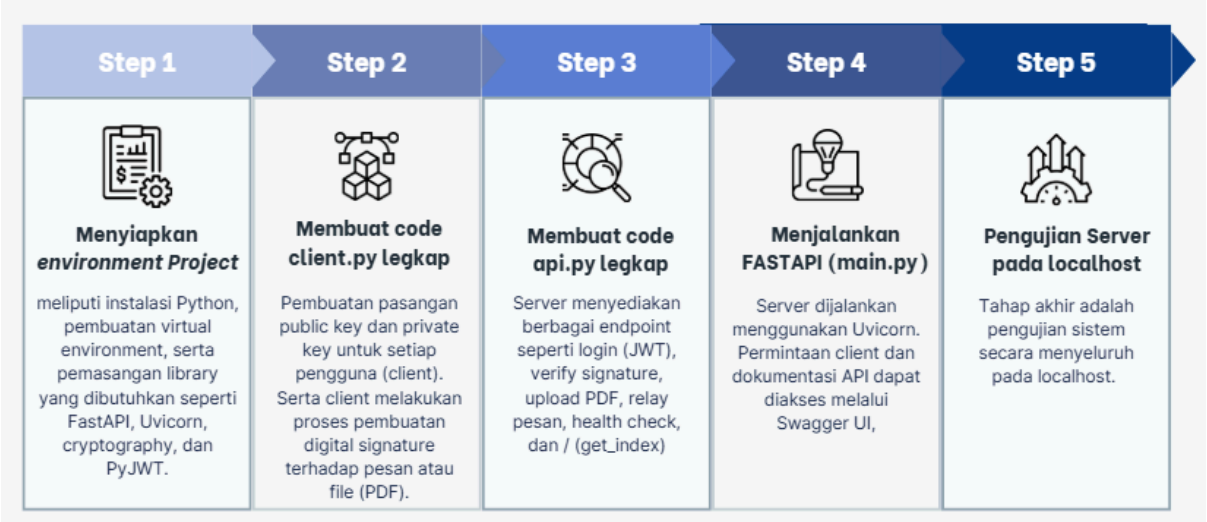
Server yang kami buat merupakan layanan keamanan berbasis *API* yang dikembangkan untuk mendukung operasional pertukaran data antar pengguna. *Serve*r ini berperan sebagai pusat pengelolaan keamanan dan integritas data, khususnya dalam proses penyimpanan kunci publik, verifikasi tanda tangan digital, pengiriman pesan aman, serta penandatanganan dokumen digital. Implementasi *server* ini bertujuan untuk memastikan bahwa setiap data yang dipertukarkan memiliki keaslian, keutuhan, dan hanya dapat diakses oleh pihak yang berwenang.

Server dibangun menggunakan *framework FastAPI* dengan bahasa pemrograman *Python*. *FastAPI* dipilih karena memiliki performa yang baik, dokumentasi *API* otomatis melalui *Swagger*, serta mendukung pengembangan layanan berbasis *REST* secara terstruktur. Selain itu, *server* ini memanfaatkan beberapa teknik kriptografi dasar, seperti kriptografi asimetris untuk pembuatan dan verifikasi tanda tangan digital, serta kriptografi simetris untuk pengamanan dokumen dalam bentuk enkripsi.

Dalam sistem ini, terdapat dua entitas utama, yaitu *client* dan *server* (api.py). *Client* disimulasikan sebagai pengguna yang bertugas membuat pasangan kunci kriptografi, menandatangani pesan atau dokumen, serta mengirimkan permintaan layanan ke *server*. *Server* berfungsi untuk menyimpan *public key* milik *client*, melakukan verifikasi *signature* berdasarkan *public key* tersebut, serta meneruskan pesan aman ke penerima yang dituju. Untuk meningkatkan keamanan komunikasi, server ini juga dilengkapi dengan mekanisme *secure session* menggunakan JSON Web Token (JWT). Setiap *client* diwajibkan melakukan autentikasi untuk memperoleh token sebelum dapat mengakses layanan-layanan penting yang tersedia pada *server*. Dengan demikian, hanya client yang telah terautentikasi yang dapat menyimpan *public key*, melakukan verifikasi *signature*, *relay* pesan, maupun menandatangani dokumen PDF. Penerapan *secure session* ini bertujuan untuk mencegah akses tidak sah terhadap layanan keamanan yang disediakan.

Secara keseluruhan, *server* dirancang untuk memenuhi kebutuhan keamanan dasar pada sistem pertukaran data, meliputi aspek keaslian data (*authenticity*), keutuhan data (*integrity)*, serta kontrol akses (*access control*). Seluruh layanan yang dikembangkan pada server ini selanjutnya akan diuji menggunakan antarmuka *Swagger API* untuk memastikan bahwa setiap fungsi berjalan sesuai dengan spesifikasi yang telah ditentukan.

2.2 Diagram Proses Pengerjaan



Adapun tahapan pengerjaan yang dilakukan, sebagai berikut:

1. Menyiapkan *Environment Project*

   Proses ini meliputi instalasi bahasa pemrograman Python sebagai platform utama pengembangan. Selanjutnya dibuat virtual environment untuk mengisolasi dependensi proyek sehingga tidak bercampur dengan konfigurasi sistem lain. Pada tahap ini juga dilakukan pemasangan *library* yang dibutuhkan, antara lain *FastAPI* sebagai *framework* pengembangan web *service*, *Uvicorn* sebagai *ASGI server* untuk menjalankan aplikasi, *cryptography* untuk kebutuhan kriptografi seperti pembuatan *key* dan *digital signature*, serta PyJWT untuk implementasi mekanisme *secure session* berbasis token JWT. Dengan environment yang telah disiapkan dengan baik, proses pengembangan dan pengujian dapat dilakukan secara terstruktur dan stabil.

2. Membuat code client.py lengkap

   Pada tahap ini dilakukan pembuatan pasangan *public key* dan *private key* untuk setiap pengguna. *Private key* digunakan oleh *client* untuk melakukan proses penandatanganan digital, sedangkan *public key* disimpan di server untuk proses verifikasi. Selain pembuatan *key, client* juga melakukan proses pembuatan *digital signature* terhadap pesan teks maupun file PDF. Data terlebih dahulu di-hash menggunakan algoritma SHA-256, kemudian hasil hash tersebut ditandatangani menggunakan private key milik pengguna. Pada sisi client juga diterapkan variasi cipher, yaitu kombinasi kriptografi asimetris (*digital signature*) dan kriptografi simetris (enkripsi file PDF menggunakan AES/Fernet). Tahap ini memastikan bahwa client mampu menghasilkan data yang aman, autentik, dan memiliki integritas.

3. Membuat code api.py lengkap

   Tahap selanjutnya adalah pengembangan kode *server* menggunakan *FastAPI. Server* dirancang untuk menyediakan berbagai endpoint yang mendukung kebutuhan sistem keamanan, antara lain login berbasis JWT, verifikasi *digital signature,* unggah dan verifikasi file PDF, relay pesan antar pengguna, *health check*, serta *endpoint roo*t (/) sebagai halaman awal layanan. Pada tahap ini, *server* bertanggung jawab untuk memverifikasi *signature* menggunakan *public key* pengguna, memvalidasi token JWT pada setiap *request,* serta memastikan bahwa hanya pengguna yang terdaftar dan terautentikasi yang dapat mengakses layanan. Implementasi ini mendukung konsep *multiuser*, karena setiap pengguna memiliki *public key* dan kredensial yang berbeda yang dikelola oleh server.

4. Menjalaskan *FASTAPI (*main.py)

   Setelah seluruh *endpoint* diimplementasikan, *server* dijalankan menggunakan *Uvicorn* sebagai *ASGI server.* Pada tahap ini, aplikasi *FastAPI* mulai aktif dan siap menerima permintaan dari *client.* Dokumentasi dan antarmuka pengujian API dapat diakses melalui *Swagger UI,* sehingga pengujian endpoint dapat dilakukan secara interaktif tanpa perlu alat tambahan.

5. Pengujian *Server* pada *Localhost*

   Tahap terakhir adalah pengujian sistem secara menyeluruh pada *localhost*. Pengujian dilakukan dengan mensimulasikan *client* yang melakukan login untuk memperoleh token JWT, kemudian menggunakan token tersebut untuk mengakses *endpoint* lain seperti *verify signature*, upload PDF, dan relay pesan. Pengujian ini bertujuan untuk memastikan bahwa sistem berjalan sesuai dengan spesifikasi, yaitu hanya menerima permintaan dari pengguna yang terautentikasi, mampu memverifikasi keaslian dan integritas data, mendukung *multiuser*, serta menerapkan *secure session* menggunakan JWT.

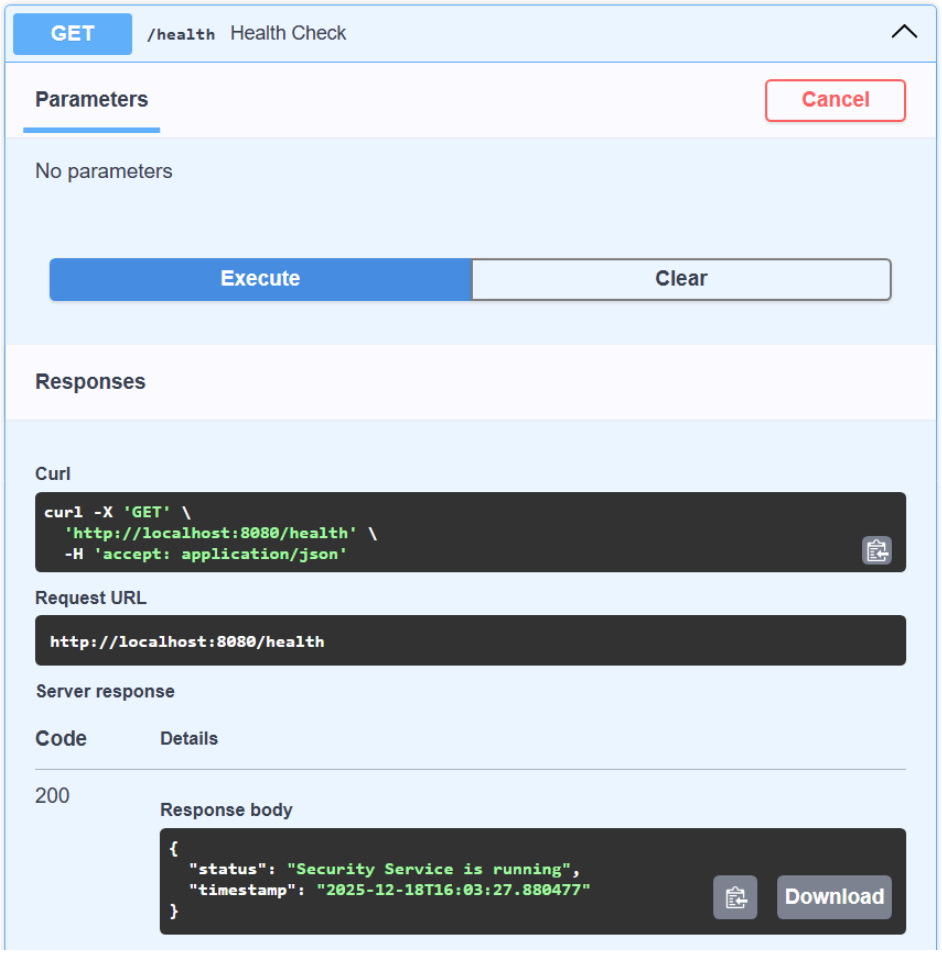
2.3 Pembuatan dan Pengujian Fungsi health_check

2.3.1 Pembuatan Fungsi health_check

```python
#HEALTH CHECK
# Fungsi contoh untuk memeriksa apakah layanan berjalan dengan baik (health check)
@app.get("/health")
async def health_check():
    return {
        "status": "Security Service is running",
        "timestamp": datetime.now().isoformat()
    }
```

Pembuatan endpoint /health digunakan sebagai mekanisme pada *health check* untuk memastikan bahwa layanan keamanan (*Security Service*) pada server berjalan dengan baik. Fungsi health_check() mengembalikan dua informasi utama, yaitu status layanan dan timestamp. Informasi status digunakan untuk menunjukkan bahwa layanan keamanan aktif dan dapat diakses, sedangkan timestamp menunjukkan waktu saat pengecekan dilakukan. Timestamp dihasilkan menggunakan fungsi datetime.now().isoformat() sehingga format waktu bersifat standar dan mudah dibaca. Endpoint ini sangat penting dalam pengujian awal sistem karena berfungsi sebagai indikator apakah server telah berhasil dijalankan sebelum layanan lain digunakan.

2.3.2 Pengujian Fungsi health_check



Pengujian dilakukan dengan mengakses endpoint /health melalui Swagger API dan browser. Berdasarkan hasil pengujian, *server* memberikan respons berupa objek JSON dengan status layanan dan waktu akses saat itu. Respons yang diterima menunjukkan bahwa layanan keamanan aktif dan berjalan tanpa error. Hasil ini membuktikan bahwa *server* telah berhasil dijalankan dan siap melayani permintaan dari *client.*

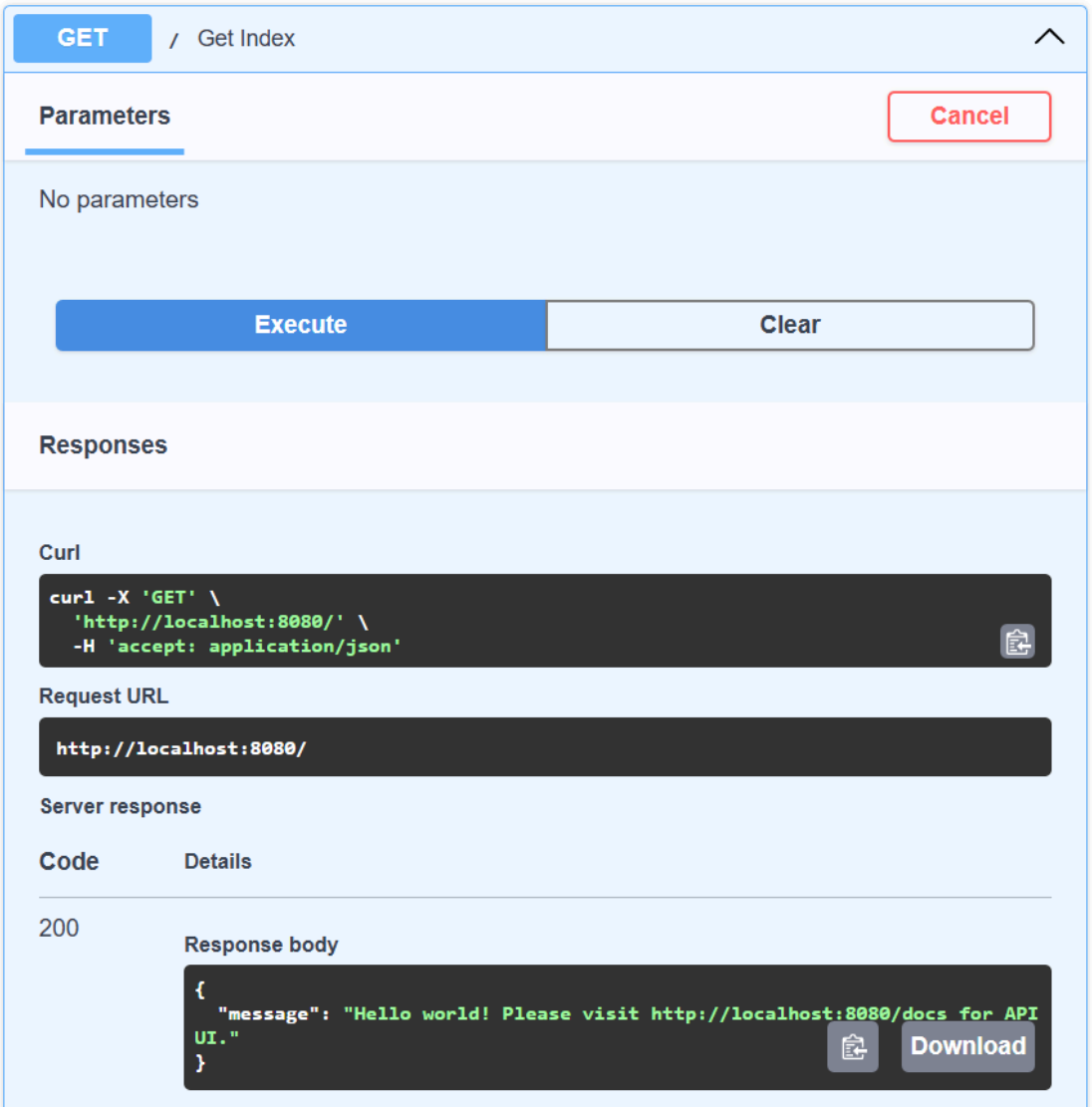
2.4 Pembuatan dan Pengujian Fungsi get_index (Endpoint /)

2.4.1 Pembuatan Fungsi get_index

```python
#ROOT
# Fungsi akses pada lokasi "root" atau "index"
@app.get("/")
async def get_index() -> dict:
    return {
        "message": "Hello world! Please visit http://localhost:8080/docs for API UI."
    }
```

Endpoint / merupakan endpoint dasar (*root endpoint*) yang berfungsi sebagai titik masuk utama ketika server diakses tanpa path tambahan. Fungsi get_index() dirancang untuk memberikan informasi awal kepada pengguna mengenai keberadaan layanan API dan cara mengakses antarmuka dokumentasi. Pesan yang dikembalikan berisi instruksi untuk mengunjungi halaman Swagger API melalui alamat /docs, sehingga memudahkan pengguna dalam melakukan eksplorasi dan pengujian endpoint lainnya.

2.4.2 Pengujian Fungsi get_index



Pengujian *endpoin*t ini dilakukan dengan mengakses alamat dasar *server* melalui browser atau Swagger API. *Server* berhasil mengembalikan pesan sambutan yang berisi instruksi untuk mengakses halaman dokumentasi API. Hasil pengujian ini menunjukkan bahwa *server* dapat diakses dengan normal pada alamat utama dan mampu memberikan respons sesuai dengan yang dirancang.

2.5 Pembuatan dan Pengujian Secure Session (JWT)

2.5.1 Pembuatan Secure Session (JWT)

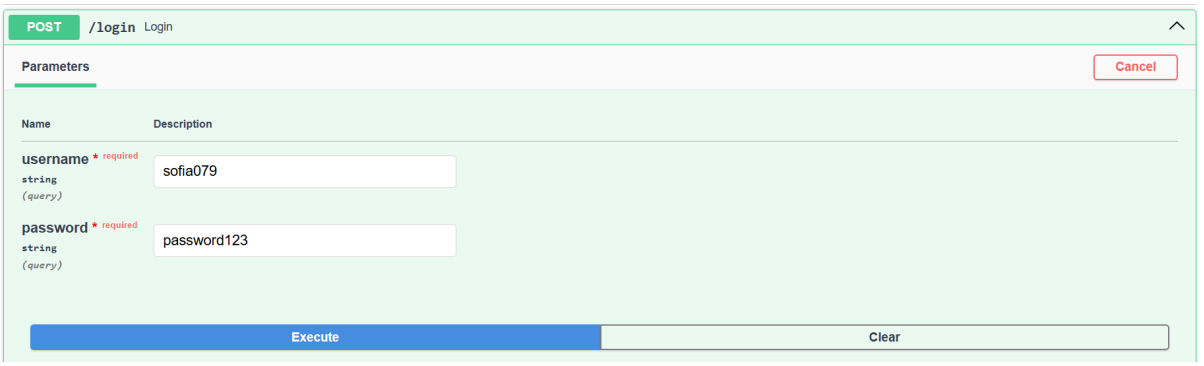
```python
#TOKEN JWT
#KONFIGURASI JWT
SECRET_KEY = "super-secret-key-kid-uas"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

security = HTTPBearer()
#ENPOINT LOGIN
USERS = {
    "sofia079": "password123",
    "penerima": "password123"
}

@app.post("/login")
async def login(username: str, password: str):
    if USERS.get(username) != password:
        raise HTTPException(status_code=401, detail="Invalid credentials")

    payload = {
        "sub": username,
        "exp": datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    }

    token = jwt.encode(payload, SECRET_KEY, algorithm=ALGORITHM)
    return {"access_token": token, "token_type": "bearer"}
```

```python
#VALIDASI TOKEN
def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security)
):
    token = credentials.credentials
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        return payload["sub"]
    except JWTError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid or expired token"
        )
```

Sistem autentikasi menggunakan JWT dengan konfigurasi SECRET_KEY yang berisi "super-secret-key-kid-uas", algoritma HS256, dan waktu kadaluarsa token adalah 240 menit. Endpoint `/login` menerima *username* dan *password*, kemudian memverifikasi kredensial pengguna terhadap database USERS yang berisi dua user (sofia079 dan penerima dengan masing-masing password). Jika kredensial valid, sistem membuat payload JWT yang berisi *username* (sub) dan waktu ekspirasi (exp), lalu mengenkripsi payload tersebut menggunakan SECRET_KEY dan algoritma HS256 untuk menghasilkan *access* token yang dikembalikan ke *client.* Untuk setiap *request* yang memerlukan autentikasi, fungsi `get_current_user()` memvalidasi token dengan mendekode token menggunakan SECRET_KEY dan algoritma

yang sama, kemudian mengembalikan *username* jika token valid atau mengembalikan HTTP Exception 401 (*Unauthorized*) jika token tidak valid atau sudah kadaluarsa. Mekanisme ini memastikan bahwa hanya pengguna yang telah login dengan kredensial yang benar dan memiliki token yang masih berlaku yang dapat mengakses endpoint-endpoint yang terlindungi.

2.5.2 Pengujian Secure Session (JWT)



Pengujian dilakukan melalui Swagger UI dengan mengakses endpoint /login menggunakan kredensial yang valid. Hasil pengujian menunjukkan server berhasil mengembalikan access_token JWT dengan status HTTP 200 OK, sebagaimana terlihat pada respons yang menampilkan field access_token dan token_type: bearer. Token yang diperoleh kemudian digunakan pada header *Authorization* untuk mengakses *endpoint* lain. Seluruh *endpoint* yang dilindungi berhasil diakses selama token masih valid. Ketika token tidak disertakan atau token tidak valid, *server* secara konsisten menolak permintaan dengan respons 401 *Unauthorized.* Hasil ini membuktikan bahwa mekanisme *secure session* menggunakan JWT telah berjalan dengan baik, mampu membatasi akses hanya pada pengguna terautentikasi, serta memenuhi kebutuhan keamanan sesi sesuai dengan spesifikasi tugas.

2.6 Pembuatan dan Pengujian Store

2.6.1 Pembuatan Store

Endpoint /store berfungsi untuk menyimpan *public key* milik pengguna ke dalam sistem. *Public key* ini nantinya digunakan oleh *server* untuk melakukan verifikasi *signature* pada proses komunikasi selanjutnya. Dengan menyimpan *public key* di *server,* sistem dapat memastikan bahwa setiap pesan yang diterima benar-benar berasal dari pengirim yang sah dan tidak mengalami perubahan.

Endpoint /store diimplementasikan menggunakan metode POST dengan URL /store. *Endpoint* menerima dua input utama, yaitu *username* dan file *public key* dalam format .pem. Selain itu, *endpoint* ini dilindungi dengan mekanisme secure session menggunakan dependency get_current_user, sehingga hanya pengguna yang telah terautentikasi yang dapat menyimpan public key. Berikut adalah implementasi kode *endpoint* /store:

```python
@app.post("/store")
async def store_pubkey(username: str, pubkey: UploadFile = File(...), current_user: str = Depends(get_current_user)):
    # pesan kembalian ke user (sukses/gagal)
    msg = None

    try:
        contents = await pubkey.read()

        os.makedirs("punkhazard-keys", exist_ok=True)
        save_path = os.path.join("punkhazard-keys", f"{username}.pem")

        with open(save_path, "wb") as f:
            f.write(contents)

        msg = "Public key stored successfully"

    except Exception as e:
        msg = str(e)

    # Nilai kembalian berupa dictionary
    # Tambahkan keys dan values sesuai dengan kebutuhan
    return {
        "message": msg,
        "user": username
    }
```

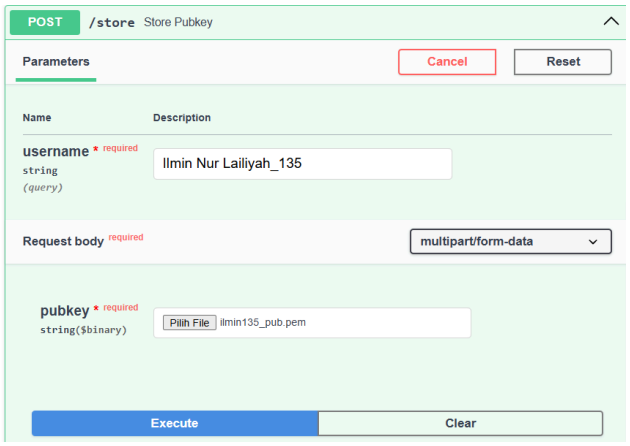
Pada proses ini, sistem akan:

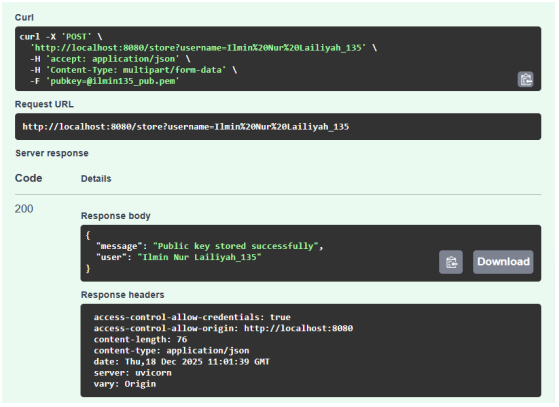
1. Membaca file public key yang diunggah oleh pengguna.
2. Membuat direktori penyimpanan punkhazard-keys jika belum tersedia.
3. Menyimpan public key dengan nama file sesuai username pengguna.
4. Mengembalikan respons berupa pesan keberhasilan atau kegagalan penyimpanan.

2.6.2 Pengujian Store

Pengujian endpoint /store dilakukan melalui antarmuka Swagger UI yang dapat diakses pada alamat http://localhost:8080/docs. Pengujian ini bertujuan untuk memastikan bahwa sistem dapat menerima input dengan benar serta menyimpan public key sesuai dengan username yang diberikan. Langkah Pengujian :

1. Pengguna mengakses *endpoint* POST /store melalui Swagger UI.
2. Pada parameter *username*, pengguna mengisi nama identitas yang akan digunakan sebagai pengenal *public key,* saya sebagai pengguna mengisinya dengan *username* "Ilmin Nur Lailiyah_135".
3. Pada parameter pubkey, pengguna mengunggah file *public key* dalam format .pem, misalnya, saya mengunggah dengan file *public key* saya yaitu "ilmin135_pub.pem"
4. Setelah seluruh parameter diisi, pengguna menekan tombol *Execute* untuk mengirim permintaan ke server.





Hasil pengujian menunjukkan bahwa *endpoint* /store telah berfungsi dengan baik. *Public key* berhasil diterima oleh *server* dan disimpan ke dalam direktori punkhazard-keys dengan nama file sesuai dengan username pengguna yaitu “Ilmin Nur Lailiyah_135”. Pesan “*Public key stored successfully*” menandakan bahwa proses penyimpanan berjalan sukses tanpa kendala. Keberhasilan ini memiliki peran penting dalam sistem keamanan yang dibangun, karena public key yang tersimpan akan digunakan pada tahap berikutnya. Dengan adanya mekanisme penyimpanan *public key* yang terstruktur dan terautentikasi, sistem mampu menjamin aspek *authentication* dan *integrity* dalam komunikasi data.

2.7 Pembuatan dan Pengujian Verify

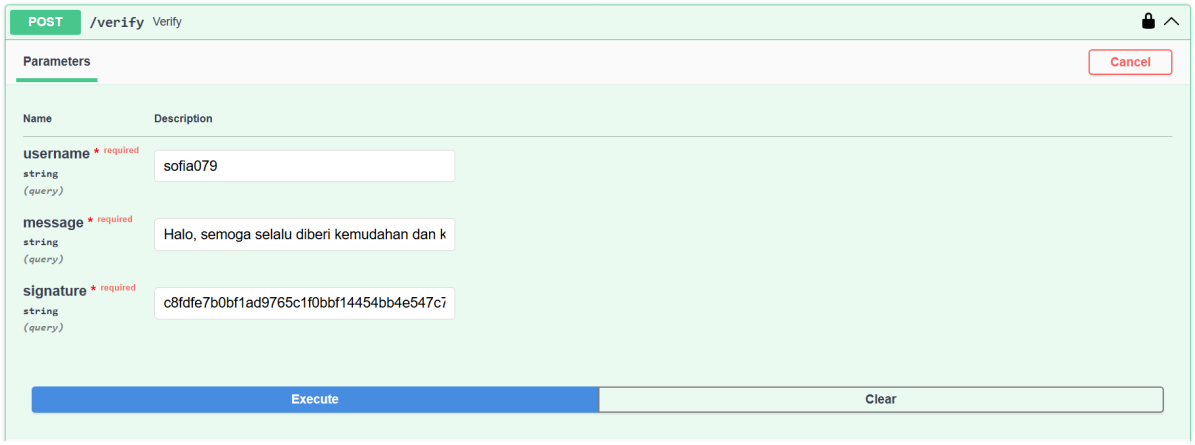
2.7.1 Pembuatan Verify

```python
#VERIFY SIGNATURE
@app.post("/verify")
async def verify(username: str, message: str, signature: str, current_user: str = Depends(get_current_user)):
    pub_key = os.path.join("punkhazard-keys", f"{username}.pem")

    # 1. Cek public key
    if not os.path.exists(pub_key):
        return {"message": "Public key not found"}

    try:
        # 2. Load public key
        with open(pub_key, "rb") as f:
            pub_key = serialization.load_pem_public_key(f.read())

        # 3. Verifikasi signature
        pub_key.verify(
            bytes.fromhex(signature),
            message.encode()
        )

        return {
            "message": "Signature valid",
            "username": username
        }

    except Exception:
        return {
            "message": "Signature invalid",
            "username": username
        }
```

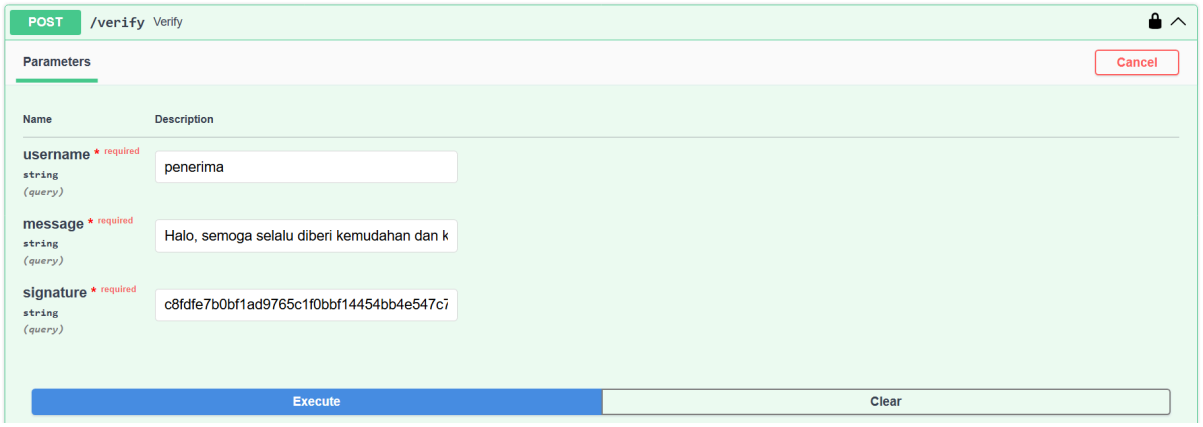
Pembuatan *endpoint* /verify digunakan untuk melakukan verifikasi tanda tangan digital terhadap sebuah pesan menggunakan kriptografi *public key* Proses verifikasi *signature* yaitu *server* melakukan verifikasi dengan mencocokkan *signature* (yang dikirim *client* dalam format heksadesimal) dengan pesan asli (*massage)* menggunakan *public key* yang sesuai. jika proses verifikasi berhasil tanpa error, *server* mengembalikan pesan *"Signature valid"*. Sebaliknya, jika terjadi kesalahan (misalnya pesan diubah atau signature tidak sesuai), *server* mengembalikan pesan *"Signature invalid"*.

Endpoint ini menerima beberapa parameter utama, yaitu *username, message*, dan *signature,* serta satu parameter tambahan current_user yang diperoleh melalui mekanisme *dependency injection* menggunakan Depends(get_current_user). Parameter current_user berfungsi sebagai bagian dari *secure session,* di mana hanya *client* yang telah terautentikasi dan memiliki token yang valid yang dapat mengakses layanan ini.

2.7.2 Pengujian Verify



Pengujian endpoint /verify dilakukan melalui Swagger API dengan memasukkan parameter *username, message, dan signature*, serta menyertakan token autentikasi terlebih dahulu pada header Authorization (*Bearer Token*). Pada pengujian pertama, *signature* dibuat di sisi client.py menggunakan *private key* milik pengguna yang sesuai dengan *username.* Pesan yang dikirimkan ke *server* sama persis dengan pesan yang digunakan saat proses penandatanganan. Hasil pengujian menunjukkan respons *"Signature valid"*, yang menandakan bahwa proses verifikasi berhasil dan pesan memiliki integritas yang terjaga.



Details

Error: Unauthorized

Response body

```
{
  "detail": "Invalid or expired token"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 37
content-type: application/json
date: Thu,18 Dec 2025 11:18:33 GMT
server: uvicorn
```

Pada pengujian kedua, *signature* dibuat menggunakan private key milik pengguna A, namun saat verifikasi *server* diberikan *username* pengguna B. Akibatnya, *server* menggunakan *public key* yang tidak sesuai untuk proses verifikasi. Hasil pengujian menunjukkan respons "*Signature invalid*". Hal ini membuktikan bahwa sistem mampu mendeteksi ketidaksesuaian antara *identity* pengguna dan kunci kriptografi, serta memastikan bahwa *signature* hanya valid jika dibuat oleh *private key* yang sesuai dengan *username* yang diberikan.

Berdasarkan pada kedua hasil pengujian menunjukkan bahwa *endpoint* /verify berfungsi sesuai dengan tujuan, yaitu melakukan verifikasi tanda tangan digital secara aman, mendukung banyak pengguna (*multi user*), serta hanya dapat diakses melalui sesi yang terautentikasi.

2.8 Pembuatan dan Pengujian Relay

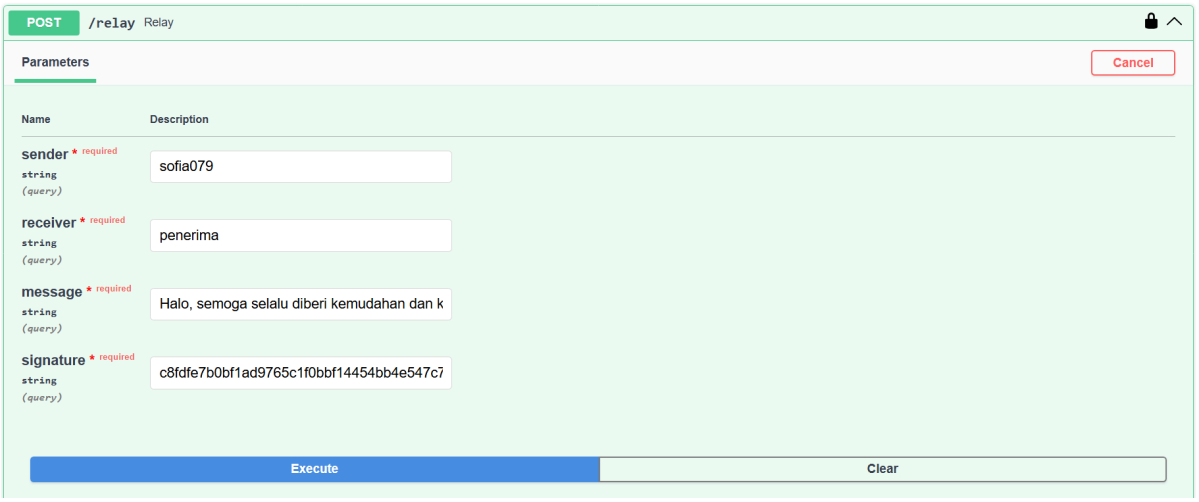
2.8.1 Pembuatan Relay

```python
#RELAY
@app.post("/relay")
async def relay(
    sender: str,
    receiver: str,
    message: str,
    signature: str,
    current_user: str = Depends(get_current_user)
):
    sender_pubkey_path = f"punkhazard-keys/{sender}.pem"
    receiver_pubkey_path = f"punkhazard-keys/{receiver}.pem"

    # 1. Cek user terdaftar
    if not os.path.exists(sender_pubkey_path):
        raise HTTPException(status_code=404, detail="Sender tidak terdaftar")

    if not os.path.exists(receiver_pubkey_path):
        raise HTTPException(status_code=404, detail="Receiver tidak terdaftar")

    # 2. Load public key pengirim
    with open(sender_pubkey_path, "rb") as f:
        pub_key = serialization.load_pem_public_key(f.read())
```

```python
    # 3. Verifikasi signature
    try:
        pub_key.verify(
            bytes.fromhex(signature),
            message.encode()
        )
    except InvalidSignature:
        raise HTTPException(
            status_code=400,
            detail="Signature tidak valid, pesan mungkin telah diubah"
        )

    # 4. Simpan pesan (simulasi relay)
    os.makedirs("data", exist_ok=True)
    timestamp = datetime.now().isoformat()
    with open("data/relay_log.txt", "a") as f:
        f.write(f"[{timestamp}] {sender} -> {receiver}: {message}\n")

    return {
        "message": "Pesan berhasil direlay dan terverifikasi",
        "from": sender,
        "to": receiver,
        "content": message
    }
```

Endpoint /relay berfungsi untuk mengirimkan pesan terverifikasi dari satu pengguna ke pengguna lain dengan mekanisme digital *signature*. *Endpoint* ini menerima parameter *sender, receiver, message, signature,* dan *current_user*. Proses kerjanya dimulai dengan pengecekan apakah kedua pengguna terdaftar dan memiliki *public key* yang valid, kemudian sistem memuat *public key* pengirim untuk memverifikasi *signature pesan.* Verifikasi dilakukan dengan mengkonversi *signature* dari format hexadecimal menjadi *bytes* dan mencocokkannya dengan pesan yang telah di encode. Jika *signature* valid, pesan berhasil di relay dan disimpan ke file relay_log.txt dengan format *timestamp, sender, receiver*, dan isi pesan; jika tidak valid, maka akan menghasilkan output pesan detail bahwa *signature* tidak valid.

2.8.2 Pengujian Relay



Server response

Code	Details
200	Response body

```
{
  "message": "Pesan berhasil direlay dan terverifikasi",
  "from": "sofia079",
  "to": "penerima",
  "content": "Halo, semoga selalu diberi kemudahan dan kelancaran"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 152
content-type: application/json
date: Thu,18 Dec 2025 09:16:05 GMT
server: uvicorn
```

Pengujian *endpoint* /relay dilakukan melalui Swagger API dengan memasukkan parameter *sender, receiver, message,* dan *signature* serta menyertakan token autentikasi terlebih dahulu pada header Authorization (*Bearer Token*). Pada pengujian, pengirim dan penerima yang keduanya telah terdaftar berhasil melakukan relay pesan dengan memasukkan *username sender, username receiver,* pesan, dan *signature* yang sesuai. Server memberikan respons "Pesan berhasil direlay dan terverifikasi", serta mencatat pesan ke dalam log server. Sebaliknya, ketika *signature* tidak sesuai atau pesan dimodifikasi tanpa memperbarui *signature, server* menolak permintaan dengan respons "*Signature* tidak valid". Hasil ini membuktikan bahwa sistem mampu mendeteksi pelanggaran integritas data dan hanya meneruskan pesan yang valid secara kriptografis.

2.9 Pembuatan dan Pengujian Upload pdf

2.9.1 Pembuatan Upload pdf

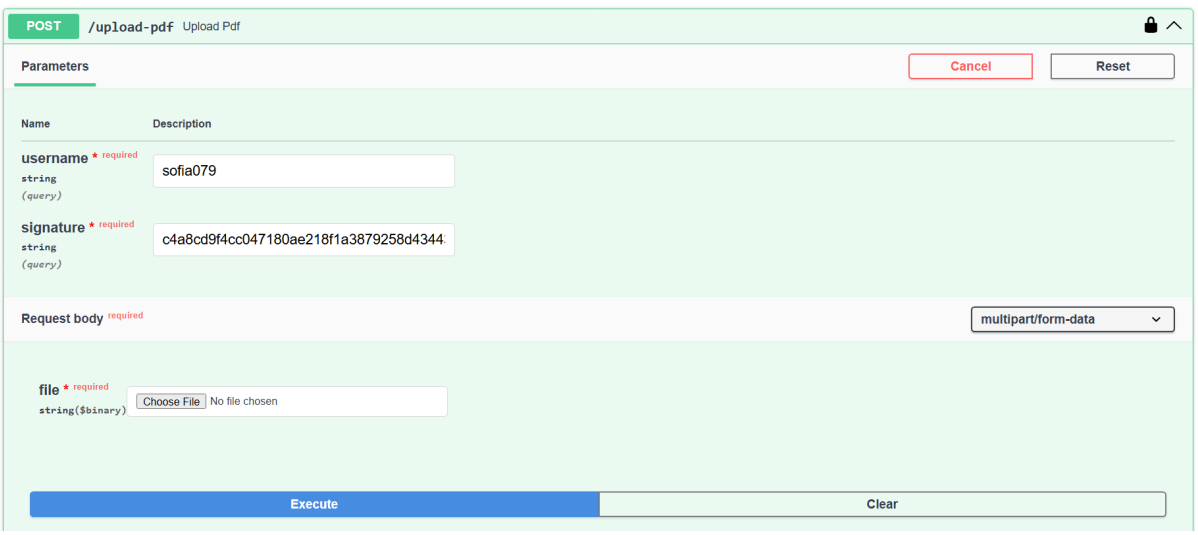
```python
#UPLOAD PDF
@app.post("/upload-pdf")
async def upload_pdf(
    username: str,
    signature: str,
    file: UploadFile = File(...),
    current_user: str = Depends(get_current_user)
):
    pubkey_path = f"punkhazard-keys/{username}.pem"

    if not os.path.exists(pubkey_path):
        raise HTTPException(status_code=404, detail="Public key user tidak ditemukan")

    pdf_bytes = await file.read()

    # HASH PDF
    digest = hashes.Hash(hashes.SHA256())
    digest.update(pdf_bytes)
    pdf_hash = digest.finalize()

    # LOAD PUBLIC KEY
    with open(pubkey_path, "rb") as f:
        pub_key = serialization.load_pem_public_key(f.read())
```

```python
# VERIFY SIGNATURE
    try:
        pub_key.verify(
            bytes.fromhex(signature),
            pdf_hash
        )
    except InvalidSignature:
        raise HTTPException(status_code=400, detail="Signature PDF tidak valid")

    # SIMPAN PDF
    os.makedirs("uploads", exist_ok=True)
    save_path = f"uploads/{file.filename}"
    with open(save_path, "wb") as f:
        f.write(pdf_bytes)

    return {
        "message": "PDF berhasil diverifikasi dan disimpan",
        "user": username,
        "file": file.filename,
        "timestamp": datetime.now().isoformat()
    }
```

Pembuatan *endpoint* /upload-pdf digunakan mengunggah file PDF yang telah ditandatangani secara digital oleh pengguna. Proses ini bertujuan untuk memastikan keaslian pengirim dan integritas dokumen sebelum file disimpan di *server.* Terdapat beberapa parameter yang digunakan yaitu username digunakan untuk menentukan pemilik *public key,* parameter *signature* berisi tanda tangan digital dari hash PDF, dan parameter file menerima file PDF dari pengguna. Depends(get_current_user) memastikan bahwa hanya pengguna dengan token (Bearer) yang valid yang dapat mengakses endpoint ini.

File PDF dibaca dalam bentuk byte. Server menghitung hash SHA-256 dari isi PDF sebagai representasi unik dokumen. Kemudian *Public key* pengguna dimuat dari file. *Server* memverifikasi *signature* menggunakan hash PDF. Jika signature tidak cocok, server menolak file dengan pesan "*Signature* PDF tidak valid". Jika verifikasi berhasil, file PDF disimpan ke folder *uploads.* Server mengembalikan *respons* sukses beserta metadata waktu unggah.

2.9.2 Pengujian Upload pdf



Pengujian *endpoint* /upload-pdf dilakukan melalui Swagger API dengan memasukkan parameter *username, signature,* dan unggah file pdf serta menyertakan token autentikasi terlebih dahulu pada header Authorization (*Bearer Token*). Pengujian *endpoint* /upload-pdf menunjukkan bahwa sistem berhasil menerima unggahan PDF dengan parameter *username, signature,* dan file PDF yang sesuai. *Server* memverifikasi keabsahan dokumen dengan cara menghitung hash SHA-256 dari isi PDF, kemudian memvalidasi *signature* menggunakan public key milik user yang bersangkutan. Apabila signature valid, file PDF disimpan ke server dan sistem mengembalikan respons sukses.

Endpoint ini mendukung *multiuser,* karena proses verifikasi *public key* dilakukan secara dinamis berdasarkan *username* yang dikirimkan. Dengan demikian, setiap pengguna memiliki pasangan kunci yang berbeda dan *server* dapat memverifikasi PDF dari banyak *user* tanpa bergantung pada satu kunci tetap.

Selain itu, pengujian ini juga memenuhi kriteria variasi *cipher* (Pythagoras). Kriptografi asimetris digunakan untuk penandatanganan dan verifikasi PDF (Ed25519), sedangkan kriptografi simetris digunakan untuk proses enkripsi PDF menggunakan algoritma AES (Fernet). Proses variasi cipher ini diimplementasikan pada sisi client di dalam file client.py.

2.10 Evaluasi Pengujian Server

Berdasarkan hasil pengujian terhadap seluruh *endpoint* yang telah diimplementasikan, *server Punk Records-v1* menunjukkan kinerja yang sesuai dengan tujuan perancangan sistem keamanan yang telah ditetapkan. Setiap layanan diuji melalui antarmuka Swagger API dengan memperhatikan aspek fungsionalitas, keamanan, serta kesesuaian dengan skenario penggunaan *multiuser*.

	Aspek pekerjaan			
	Simpan public key	Verifikasi signature	Relay message	Tandatangan pdf
York (Jalan)	Endpoint berfungsi dengan baik	Endpoint berhasil melakukan verifikasi	Endpoint berhasil meneruskan pesan	Endpoint berhasil menerima dan memverifikasi PDF
Edison (Static entry)	Public key tersimpan di direktori punkhazard-keys	Menggunakan public key yang tersimpan	Menggunakan public key untuk verifikasi	-
Edison (Integrity check)	File .pem tersimpan dengan aman	Memverifikasi integritas pesan dengan Ed25519	Memverifikasi signature sebelum relay	-
Pythagoras (Variasi cipher)	Menggunakan kriptografi public key	-	-	Kombinasi Ed25519 (signature) + AES-256/Fernet (enkripsi)
Pythagoras (Multiuser)	Mendukung penyimpanan untuk banyak user berdasarkan	Dapat memverifikasi signature dari berbagai user	-	Mendukung upload PDF dari berbagai user

	username			
Stella (Secure session)	Dilindungi dengan JWT (get_current_user)	Dilindungi dengan JWT (get_current_user)	Dilindungi dengan JWT (get_current_user)	Dilindungi dengan JWT (get_current_user)

Secara keseluruhan, hasil evaluasi menunjukkan bahwa sistem telah memenuhi aspek *static entry*, *integrity check*, variasi cipher (kriptografi simetris dan asimetris), dukungan multiuser, serta *secure session*. Dengan demikian, server Punk Records-v1 dapat dikatakan telah berfungsi sesuai dengan spesifikasi dan tujuan keamanan yang dirancang pada bab-bab sebelumnya.

BAB III
PENUTUP

3.1 Kesimpulan

Berdasarkan hasil pembuatan dan pengujian *server* Punk Records-v1, dapat disimpulkan bahwa layanan keamanan berbasis API yang menerapkan mekanisme kriptografi public key telah berhasil dirancang dan diimplementasikan menggunakan *framework FastAPI*. Sistem ini mampu menjamin keaslian dan integritas data dalam proses pengiriman pesan serta unggah dokumen PDF melalui penerapan tanda tangan digital berbasis algoritma Ed25519. Penerapan tanda tangan digital tersebut terbukti efektif dalam memverifikasi bahwa data atau dokumen yang diterima tidak mengalami perubahan selama proses transmisi, sebagaimana ditunjukkan pada pengujian *endpoint* /verify yang mampu mendeteksi ketidaksesuaian *signature* dan menolak pesan yang telah dimodifikasi. Selain itu, mekanisme autentikasi berbasis JSON Web Token (JWT) dengan algoritma HS256 telah berhasil diterapkan untuk memastikan bahwa setiap permintaan layanan hanya dapat diakses oleh pengguna yang sah. *Secure session* dengan masa berlaku token selama 240 menit memberikan lapisan keamanan tambahan pada seluruh endpoint sensitif, seperti /store, /verify, /relay, dan /upload-pdf. Secara keseluruhan, *server* Punk Records-v1 telah memenuhi seluruh aspek keamanan yang ditetapkan, meliputi kemampuan *server* untuk berjalan dengan baik (York), penyimpanan *public key* secara terstruktur (Edison), penerapan *integrity check* melalui verifikasi signature (Pythagoras), penggunaan variasi cipher dengan kombinasi kriptografi asimetris dan simetris (Stella), dukungan terhadap sistem multiuser, serta perlindungan akses menggunakan *secure session* berbasis JWT. Implementasi kriptografi dengan kombinasi AES-256 untuk enkripsi data, HS256 untuk autentikasi token, dan Ed25519 untuk verifikasi tanda tangan digital terbukti mampu meningkatkan keamanan sistem secara signifikan dalam lingkungan pertukaran data yang bersifat kolaboratif dan sensitif.

3.2 Kendala

Selama proses pengembangan dan pengujian server Punk Records-v1, terdapat beberapa kendala yang dihadapi. Pada tahap awal pengujian beberapa *endpoint,* hasil verifikasi seringkali menunjukkan status *invalid*. Hal ini disebabkan oleh ketidaksesuaian penamaan file *public key* yang digunakan antara kode pada sisi *client* dan kode pada sisi

server (API), sehingga *server* memuat *public key* yang tidak sesuai saat proses verifikasi *signature.* Kendala ini dapat diatasi dengan mencocokkan dan menyamakan penamaan *public key* pada kedua sisi agar proses verifikasi dapat berjalan dengan benar. Kendala lainnya adalah pada proses pembaruan kode *server*. Setiap kali dilakukan perubahan atau penambahan fitur pada kode, server harus dijalankan ulang menggunakan perintah uv run main.py agar dokumentasi Swagger API dapat diperbarui dan menampilkan endpoint sesuai dengan versi kode terbaru. Apabila server tidak dijalankan ulang, Swagger masih menampilkan konfigurasi lama sehingga dapat menimbulkan kebingungan saat pengujian. Meskipun demikian, kendala ini dapat diatasi dengan memastikan server selalu *restart* setelah dilakukan perubahan kode.

DAFTAR PUSTAKA

Hendra, H., Awan, A., Waisen, W., Wilianto, W., & Yudi, Y. (2025). Memperkuat Autentikasi dan Integritas Data REST-API Menggunakan Token HMAC SHA-256. Jurnal Minfo Polgan. https://jurnal.polgan.ac.id/index.php/jmp/article/view/14406

IETF. (2015). JSON Web Algorithms (JWA). RFC 7518. https://tools.ietf.org/html/rfc7518

Prasetya, A. B., & Suharjo, I. (2025). Backend API Data Protection Menggunakan JWT Token dan Algoritma AES 256-bit dengan Bahasa Pemrograman Golang. Jurnal Informatika dan Teknik Elektro Terapan. https://journal.eng.unila.ac.id/index.php/jitet/article/view/5699

Setiadi, I., Widianti, S., & Kayuan, I. P. P. (2025). Implementasi Kriptografi Pengamanan Data Soal Ujian di Lingkungan Perguruan Tinggi Menggunakan Algoritma AES-256 dan SHA-256. Jurnal Penelitian Rumpun Ilmu Teknik. https://ejurnal.politeknikpratama.ac.id/index.php/JUPRIT/article/view/4569

Shofyan, F., & Shita, R. T. (2024). Implementasi Web Service RESTful API dengan Autentikasi Personal Access Tokens dan Algoritma AES-256. *Jurnal Ticom: Technology of Information and Communication. https://jurnal-ticom.jakarta.aptikom.org/index.php/Ticom/article/view/130

Lampiran

Link Github : Link github kelompok 9_UAS KID