Universitatea Tehnică ”Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Calculatoare și Tehnologia Informației Specializarea Tehnologia Informației

CryptoKey Manager

Studenți:

Bejinaru Mihnea-George, Grupa 1408B

Chircă Radu-Iulian, Grupa 1408B

Ciocan Ștefan-Robert, Grupa 1408B

Nistor Florin, Grupa 1408B

Cadru didactic coordonator:

Ș.l.dr.ing. Alexandru-Gabriel Tudorache

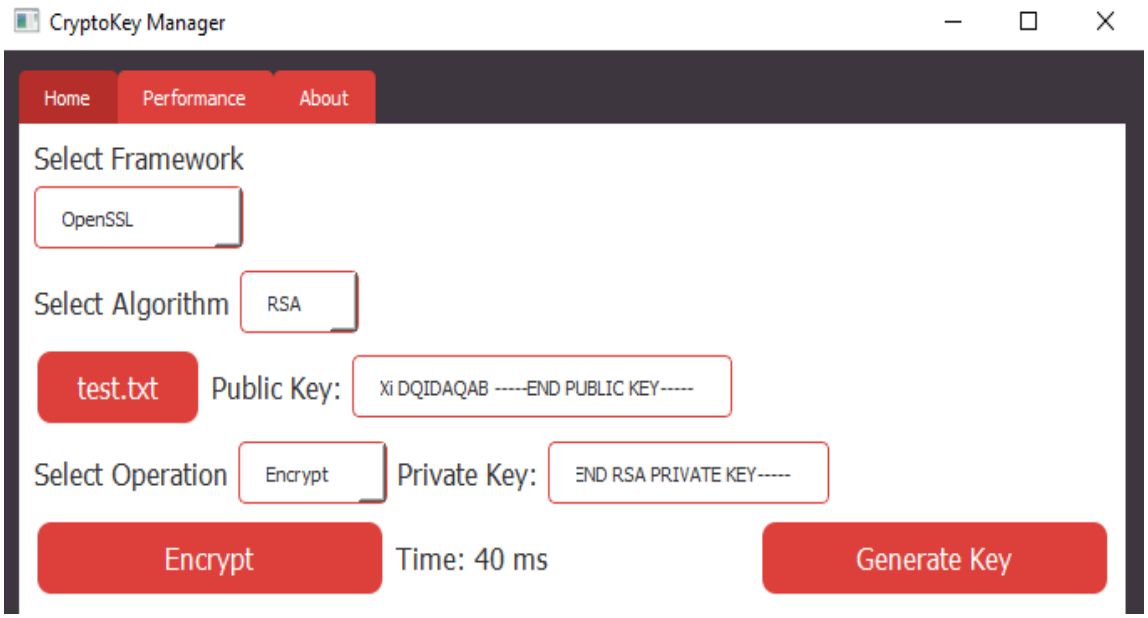
Mai 2024

Cuprins

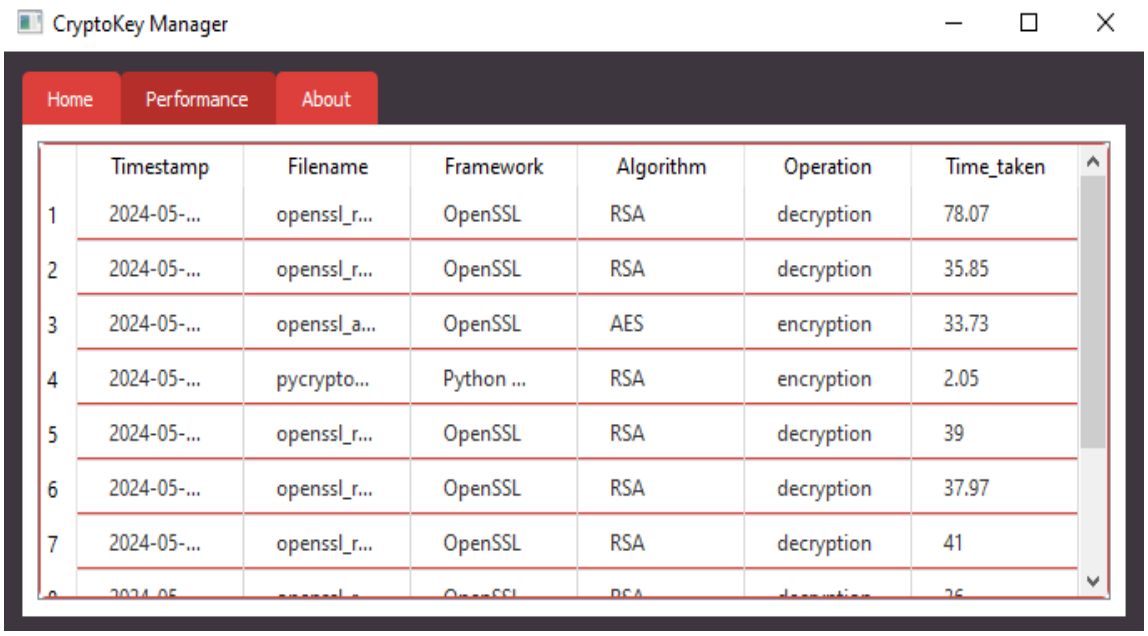
Capitolul 1. Descrierea proiectului

Proiectul nostru ofera o solutie desktop simpla si rapida pentru criptarea si decriptarea fisierelor cu diferiti algoritmi, precum RSA si AES, avand integrare cu framework-urile OpenSSL si PyCryptodome, cat si cu biblioteca standard de criptare din Python.

Se prezinta printr-o interfata grafica intuitiva, dezvoltata in PySide2, care permite utilizatorului sa selecteze in functie de preferinte frameworkul dorit, tipul de algoritm pentru a realiza operatiile si fisierul de criptat/decriptat. Cheile utilizate sunt la latitudinea utlizatorului, insa programul ofera si posibilitatea de a le genera automat.

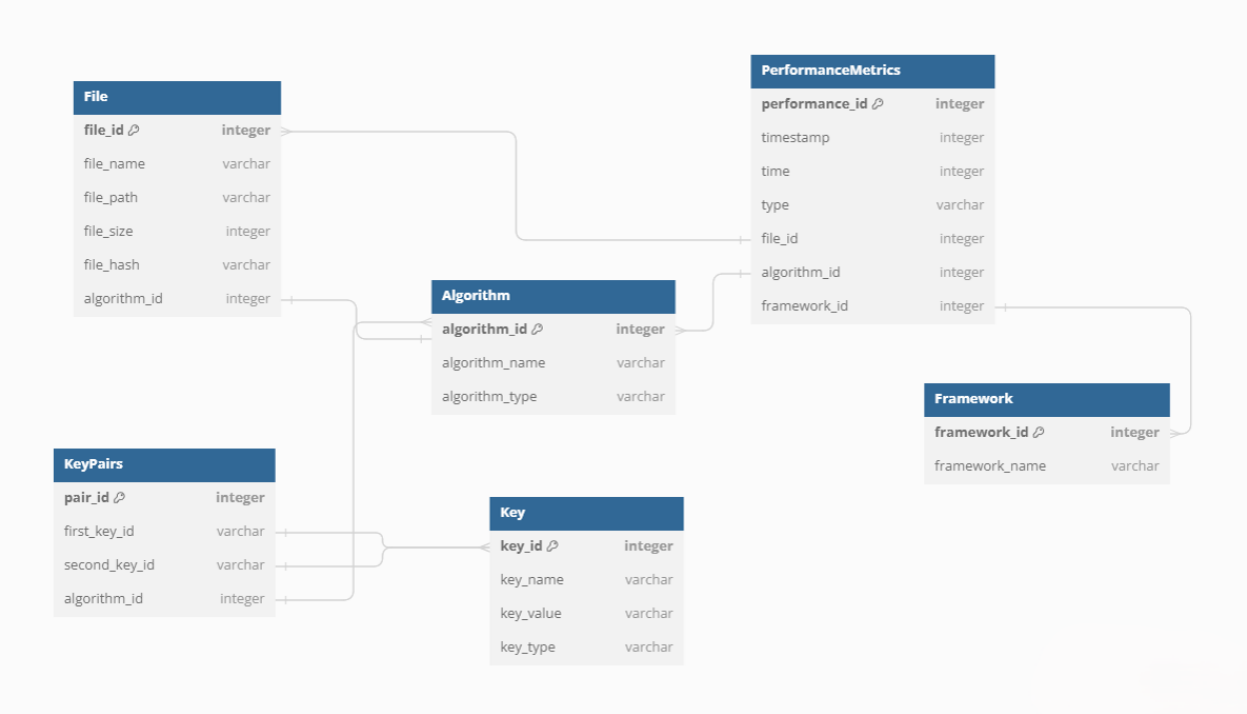


Mai mult, pentru fiecare operatie efectuata sunt prezentate ulterior metricile de performanta intr-un tabel care poate fi sortat in functie de preferintele utilizatorului.



Capitolul 2. Schema proiectului (baza de date)

Pentru lucrul cu baza de date de tip SQLite, proiectul nostru utilizează SQLAlchemy, o bibliotecă pentru gestionarea relațională a bazelor de date în Python. Definim un model de date pentru a gestiona informații despre fișiere, algoritmi, performanțe și chei criptografice.



1. File
   - Aceasta tabela reprezinta informatiile despre fisierele criptate stocate in sistem.
   - Fiecare fisier criptat este asociat cu un anumit algoritm si poate avea multiple inregistrari in tabelul PerformanceMetrics pentru a tine evidenta performantelor sale.

2. Key
   - Reprezinta informatiile despre cheile criptografice.
   - key_type este un enum cu valorile "public" si "private", sau poate fi null.
   - Cheile pot fi asociate in cadrul anumitor perechi de chei (KeyPair) in cazul algoritmilor asimetrici.

3. Algorithm
   - Reprezinta informatiile despre algoritmii criptografici disponibili în programul nostru (RSA si AES).
   - algorithm_type este un enum cu valorile "symmetric" si "asymmetric".
   - Fiecare algoritm poate fi asociat cu mai multe fisiere si poate avea perechi de chei asociate.

4. PerformanceMetrics
   - Tine evidenta metricilor de performanta asociate cu operatiile de criptare si decriptare pe fisiere.
   - Fiecare inregistrare este asociata cu un fisier (File), un algoritm (Algorithm) si un framework (Framework).

5. Framework
   - Reprezinta informatii despre framework-urile utilizate in masurarea performantelor.
   - Framework-urile disponibile sunt OpenSSL, PyCryptodome si biblioteca standard de criptografie din Python.

6. KeyPair
   - Tine evidenta perechilor de chei asociate cu algoritmii asimetrici.
   - O pereche de chei este asociata cu un algoritm si contine doua chei prezente in tabela Key.

Capitolul 3. Implementarea

3.1 Detalii de implementare

Aplicatia dispune de o arhitectura pe nivele pentru o separare clara a atributiilor claselor. Operatiunile asupra bazei de date au loc in cadrul layerului persistence, unde este prezent agentul SQLite. Acesta se foloseste de operatiuni modulare pentru a interactiona cu fiecare tabela in parte.

```
17 usages   Mihnea Bejinaru
class SQLiteAgent:
    Mihnea Bejinaru
    def __init__(self, db_name='my_database.db'):
        super().__init__()
        self.db_name = db_name
        self.engine = None

        # establish the database conn
        self.session = self.establish_connection()

        # init the helper classes
        self.file_operations = SQLiteFileOperations(self.session)
        self.key_operations = SQLiteKeyOperations(self.session)
        self.algorithm_operations = SQLiteAlgorithmOperations(self.session)
        self.performance_operations = SQLitePerformanceOperations(self.session)
        self.framework_operations = SQLiteFrameworkOperations(self.session)
        self.key_pair_operations = SQLiteKeyPairOperations(self.session)
```

O astfel de clasa contine logica pentru tranzactiile necesare in cadrul acelei tabele.

```
1 usage (1 dynamic)   Chirca
def insert_performance(self, timestamp, time_taken, op_type, file_id, algorithm_id, framework_id):
    performance = PerformanceMetrics(
        timestamp=timestamp,
        time=time_taken,
        op_type=op_type,
        file_id=file_id,
        algorithm_id=algorithm_id,
        framework_id=framework_id
    )
    self.session.add(performance)
    try:
        self.session.commit()
        return performance
    except IntegrityError:
        self.session.rollback()
        return Exception("Performance metrics insertion failed. Possible duplicate entry.")
```

De asemenea, pentru a facilita lucrul cu baza de date exista scriptul populate_hard_reset.

In layerul de business sunt prezente serviciile apelate direct din cadrul interfetei, acestea fiind cele legate de generarea cheilor, de masurare si prelucrare a datelor legate de performanta criptarii/decriptarii cat si serviciile fiecarui framework ce expun algoritmii RSA si AES propriu-zisi, fiecare avand o functie de criptare si una pentru decriptare.

```
def encrypt_rsa_file(self, file_path: str, public_key_file_path, algorithm_id: int, framework_id: int) -> (PerformanceMetrics, str):
    try:
        if not file_exists(file_path):
            raise FileNotFoundError(f"File '{file_path}' does not exist.")

        dir_path = get_directory_path(file_path)

        with open(file_path, 'rb') as file:
            plaintext = file.read()

        start_time = time.time()

        encrypted_data = self.rsa_encrypt(plaintext, public_key_file_path)

        end_time = time.time()

        time_taken = end_time - start_time

        encrypted_file_path = f"openssl_rsa_encrypted_{datetime.now().timestamp()}{os.path.splitext(file_path)[1]}"

        full_path = os.path.join(dir_path, encrypted_file_path)
        full_path = full_path.replace( __old: "\\", __new: "/")
        save_file(full_path, encrypted_data)

        encrypted_file = self.db_agent.file_operations.insert_file(file_path=full_path,
                                                                   algorithm_id=algorithm_id)

        performance = self.performance_agent.log_performance(timestamp=datetime.fromtimestamp(end_time), time_taken=time_taken,
                                                             op_type=CryptoOperation.DECRYPT.value,
                                                             file_id=encrypted_file.file_id, algorithm_id=algorithm_id,
                                                             framework_id=framework_id)
        return performance, encrypted_file_path

    except FileNotFoundError as e:
        raise FileNotFoundError(f"Error: {e}")
    except Exception as e:
        raise Exception(f"Encryption failed: {e}")
```

Se poate observa modul in care timpul necesar pentru realizarea criptarii este calculat, salvarea fisierului criptat in baza de date si a performantei.

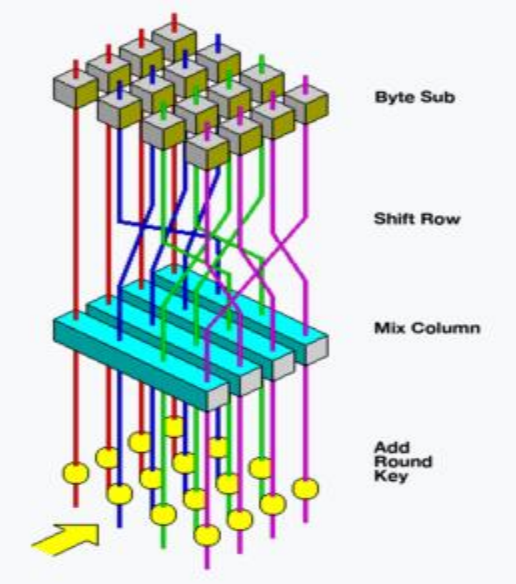
```python
@staticmethod
def rsa_encrypt(plaintext: bytes, public_key_file_path: str) -> bytes:
    try:
        if not file_exists(public_key_file_path):
            raise FileNotFoundError("PEM file containing public key does not exist.")

        openssl_cmd = f'openssl rsautl -encrypt -pubin -inkey {public_key_file_path}'
        encrypted_data = subprocess.check_output(openssl_cmd.split(), input=plaintext)

        return encrypted_data

    except Exception as e:
        raise Exception(f"Encryption failed: {e}")
```

In cazul OpenSSL criptarea/decriptarea propriu-zisa are loc in cadrul unui subproces, utilizandu-se comenzi openssl in lipsa suportului in cadrul bibliotecilor din Python. De asemenea, cheile sunt preluate din fisiere temporare create in urma rularii programului.

3.2 Algoritmi aleși

Pentru o mai mare versatilitate, s-au implementat atat un algoritm simetric (AES) cat si unul asimetric (RSA).

Algoritmul de criptare AES (Advanced Encryption Standard), cunoscut si sub numele de algoritmul Rijndael, este un algoritm simetric de cifru pe blocuri cu o dimensiune a blocului de 128 de biti. Converteaza aceste blocuri individuale folosind chei de 128, 192 si 256 de biti. Odata ce cripteaza aceste blocuri, le uneste pentru a forma textul cifrat.



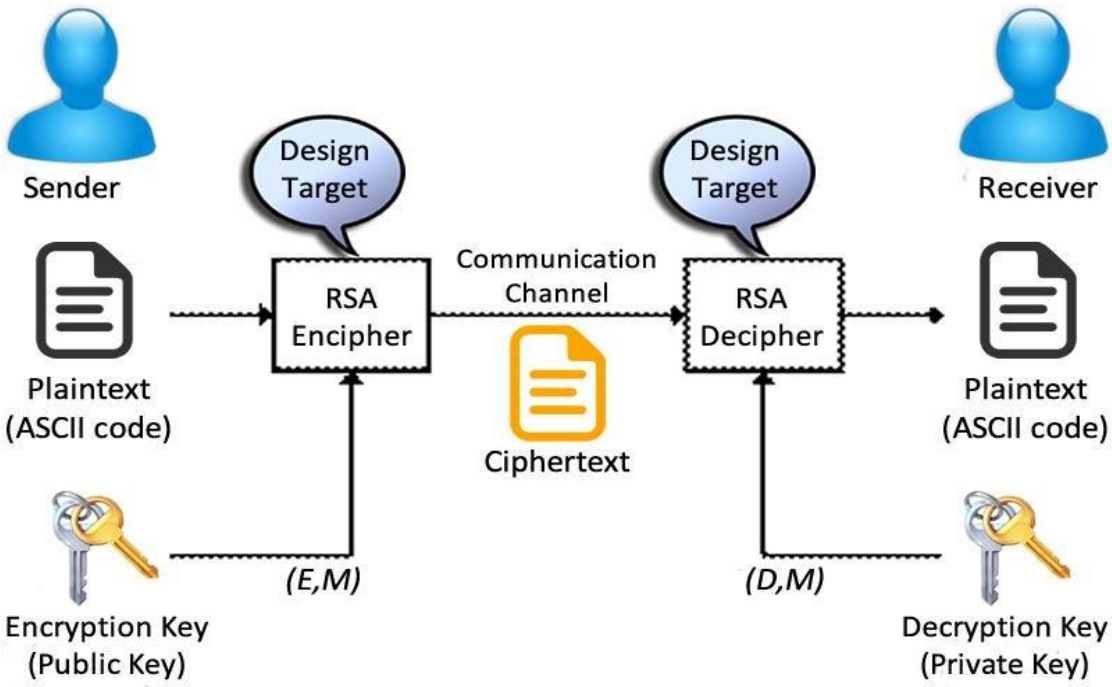
Este bazat pe o retea de substitutie-permutare, cunoscuta si sub numele de retea SP (substitutie-permutare). Constă într-o serie de operații interconectate, inclusiv inlocuirea intrarilor cu iesiri specifice (substitutii) si alte operații care implica amestecarea bitilor (permutari).

```
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
encryptor = cipher.encryptor()

padder = primitives.padding.PKCS7(algorithms.AES.block_size).padder()
padded_data = padder.update(plaintext) + padder.finalize()

encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
```

```
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
decryptor = cipher.decryptor()

decrypted_data = decryptor.update(ciphertext) + decryptor.finalize()

unpadder = primitives.padding.PKCS7(algorithms.AES.block_size).unpadder()
data = unpadder.update(decrypted_data) + unpadder.finalize()
```

Algoritmul RSA (Rivest-Shamir-Adleman) este baza unui criptosistem - o suita de algoritmi criptografici folositi pentru anumite servicii sau scopuri de securitate specifice - care permite criptarea cu chei publice si este folosit pe scara larga pentru securizarea datelor sensibile, in special atunci cand acestea sunt trimise peste o retea nesigura, cum ar fi internetul.

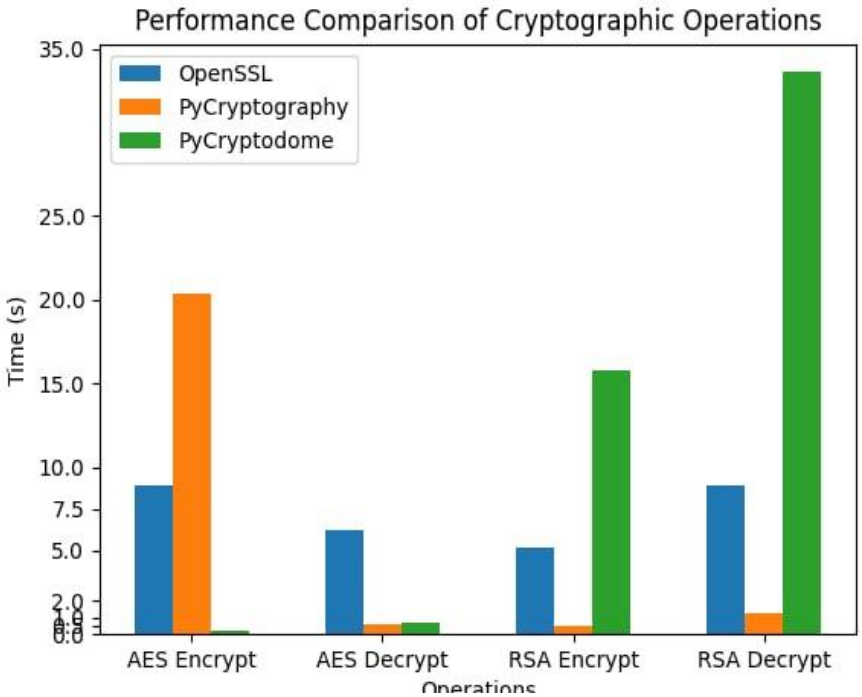
Criptografia cu cheie publica, cunoscuta si sub numele de criptografie asimetrica, foloseste doua chei diferite, dar matematic conectate - una publica si una privata. Cheia publica poate fi partajata cu toata lumea, in timp ce cheia privata trebuie pastrata secreta.



In criptografia RSA, atat cheile publice, cat si cele private pot cripta un mesaj. Cheia opusa celei folosite pentru criptarea unui mesaj este folosita pentru decriptarea acestuia. Aceasta caracteristica este unul dintre motivele pentru care RSA a devenit cel mai utilizat algoritm asimetric: furnizeaza o metoda pentru a asigura confidentialitatea, integritatea, autenticitatea si necontestarea comunicarilor electronice si a stocarii datelor.

```
rsa_public_key = RSA.import_key(public_key)
cipher = PKCS1_OAEP.new(rsa_public_key)
encrypted_data = cipher.encrypt(plaintext)
```

```
rsa_private_key = RSA.import_key(private_key)
cipher = PKCS1_OAEP.new(rsa_private_key)
decrypted_data = cipher.decrypt(ciphertext)
```



Performanta Comparativa a Operatiunilor Criptografice

Introducere

Graficul de mai sus ilustreaza comparatia performantelor pentru trei framework-uri criptografice diferite: OpenSSL, PyCryptography si PyCryptodome. Timpurile de executie au fost masurate pentru patru operatiuni criptografice: criptare AES, decriptare AES, criptare RSA si decriptare RSA.

Pentru a asigura consistenta si acuratetea testelor, toate masuratorile au fost efectuate pe acelasi fisier, cu o dimensiune constanta. Testele au fost realizate pe acelasi laptop, fara alte aplicatii deschise in afara celor necesare pentru rularea testelor.

Aceasta comparatie are scopul de a oferi o perspectiva asupra performantei acestor framework-uri in scenarii reale de utilizare.

Observatii si Statistici

OpenSSL:

AES Encrypt: 8.94 s; AES Decrypt: 6.18 s
RSA Encrypt: 5.22 s; RSA Decrypt: 8.87 s

Observatie:

OpenSSL are performante consistente intre diferitele operatiuni, cu timpi de executie relativ apropiati. Criptarea si decriptarea RSA sunt putin mai lente in comparatie cu operatiunile AES.

PyCryptography:
AES Encrypt: 20.36 s; AES Decrypt: 0.62 s
RSA Encrypt: 0.5 s; RSA Decrypt: 1.24 s

Observatie:

PyCryptography prezinta o discrepanta semnificativa intre performantele pentru AES si RSA. Desi criptarea AES este relativ lenta, RSA se descurca mult mai bine, cu timpi de executie foarte mici.

Performanta AES poate fi influentata de modul in care biblioteca interactioneaza cu librariile C subiacente.

PyCryptodome:
AES Encrypt: 0.22 s; AES Decrypt: 0.66 s
RSA Encrypt: 15.82 s; RSA Decrypt: 33.62 s

Observatie:

PyCryptodome exceleaza in operatiunile AES, avand cei mai mici timpi de executie dintre toate framework-urile analizate. Cu toate acestea, pentru RSA, in special decriptarea, timpii de executie sunt semnificativ mai mari.

Performanta RSA poate fi afectata de complexitatea algoritmului si de marimea cheilor utilizate.

Concluzii

- **OpenSSL** ofera performante echilibrate si stabile pentru toate operatiunile analizate, fiind o alegere buna pentru un set variat de operatiuni criptografice.
- **PyCryptography** este extrem de eficient pentru operatiunile RSA, dar nu si pentru AES, sugerand ca ar putea fi mai potrivit pentru aplicatii care folosesc predominant RSA.
- **PyCryptodome** este ideal pentru operatiunile AES datorita timpilor de executie foarte mici, insa performantele sale la RSA, in special decriptarea, sunt sub cele ale celorlalte framework-uri, indicand ca este mai potrivit pentru scenarii unde AES este predominant utilizat.

Factori Externi si Consideratii

Performantele masurate pot fi influentate de mai multi factori externi:

Hardware:

Specificatiile calculatorului pe care sunt rulate testele (CPU, memorie, suport hardware pentru instructiuni criptografice) pot afecta semnificativ timpii de executie.