Department of Electronic & Telecommunication Engineering

University of Moratuwa

EN4720: Security in Cyber-Physical Systems
Course Project

Milestone 2: Cryptographic API Implementation

Group Name: Decryptors

Pasqual A.C.	200445V
Gunatilake P.T.B.	200439G
Madhushan R.M.S.	200363R
Madusan A.K.C.S.	200366E

March 23, 2025

Contents

1 Introduction

This cryptographic API provides functionalities related to encryption and hashing. Both symmetric and asymmetric key encryption are supported as well as a variety of hash functions.

1.1 Overall Implementation of the API

The API supports the following features:

- Generate a key with a unique key ID for a given encryption scheme and key size
- Encrypt a message using a given key ID
- Decrypt a message using a given key ID
- Hash a message using a given hash function
- Verify if a given hash matches the data

The API is built using the Flask Python library with POST endpoints for each of the above features. It uses an SQLite database for persistent storage of the generated keys. The database is integrated into the application using Flask-SQLAlchemy.

The API is hosted at http://entcdecryptors.pythonanywhere.com, and the complete API documentation can be viewed here. The source code can be found at https://github.com/cyber-decryptors/cryptography-api.

1.2 API Usage Instructions

This section contains details on how to access the API endpoints, input parameters, and the response structure.

1.2.1 Key Generation

Generates a new encryption key.

- Request: `POST http://entcdecryptors.pythonanywhere.com/generate-key`
- Body (JSON): { `"key_type": "AES", "key_size": 256` }
- Parameters:
  - `key_type`: Type of encryption. (AES and RSA are supported)
  - `key_size`: Size of key in bits. (128, 192, 256 for AES and 1024, 2048, 3072, 4096 for RSA)
- Response: { `"key_id": "1b34e457", "key_value": "base64-encoded-key"` }
  - `key_id`: A unique hexadecimal identifier for the generated key.
  - `key_value`: The generated key encoded in Base64. (Only public key for RSA)

1.2.2 Encryption

Encrypts plaintext using a previously generated key.

- Request: `POST http://entcdecryptors.pythonanywhere.com/encrypt`
- Body (JSON): { `"key_id": "1b34e457", "plaintext": "text to encrypt", "algorithm": "AES"` }

- Parameters:
  - `key_id`: Unique identifier of a previously generated key.
  - `plaintext`: Text to be encrypted.
  - `algorithm`: Encryption algorithm to use. (AES or RSA)
- Response: { `"ciphertext": "base64-encoded-encrypted-data"` }
  - `ciphertext`: The encrypted data encoded in Base64.

1.2.3 Decryption

Decrypts ciphertext using a previously generated key.

- Request: `POST http://entcdecryptors.pythonanywhere.com/decrypt`
- Body (JSON): { `"key_id": "1b34e457", "ciphertext": "base64-encoded-encrypted-data", "algorithm": "AES"` }
- Parameters:
  - `key_id`: Unique identifier of a previously generated key.
  - `ciphertext`: Ciphertext to be decrypted.
  - `algorithm`: Decryption algorithm to use. (AES or RSA)
- Response: { `"plaintext": "decrypted text"` }
  - `plaintext`: The decrypted plaintext.

1.2.4 Hashing

Generates a hash value from input data.

- Request: `POST http://entcdecryptors.pythonanywhere.com/generate-hash`
- Body (JSON): { `"data": "data to hash", "algorithm": "SHA-256"` }
- Parameters:
  - `data`: Data to be hashed.
  - `algorithm`: Hashing algorithm to use. (SHA-224, SHA-256, SHA-384, SHA-512, MD5, BLAKE2b or BLAKE2s)
- Response: { `"hash_value": "hash-string", "algorithm": "SHA-256"` }
  - `hash_value`: Generated hash value.
  - `algorithm`: Hashing algorithm used.

1.2.5 Hash Verification

Verifies if a hash value matches the original data.

- Request: `POST http://entcdecryptors.pythonanywhere.com/verify-hash`
- Body (JSON): { `"data": "original data", "algorithm": "SHA-256", "hash_value": "hash-to-verify"` }
- Parameters:

  - `data`: Original data before hashing.
  - `algorithm`: Hashing algorithm used. (SHA-224, SHA-256, SHA-384, SHA-512, MD5, BLAKE2b or BLAKE2s)
  - `hash_value`: Hash value to verify against.

- Response: { `"is_valid": true, "message": "Hash verified successfully"` }
  - `is_valid`: Boolean indicating whether the hash matches the original data.
  - `message`: Additional verification message.

1.2.6 Error Responses

All endpoints return a consistent error format with response code 400 if an invalid request is given. (eg. Unsupported algorithm, invalid key size, nonexistent key ID)
{ `"error": "Error message description"` }

2 Design and Functionality of Cryptographic Functions

This section details the features and implementation steps of each cryptographic function provided by the API. It also demonstrates successful API operations executed through Postman. The `cryptography` Python library was used for cryptographic operations.

2.1 Symmetric Key Cryptography

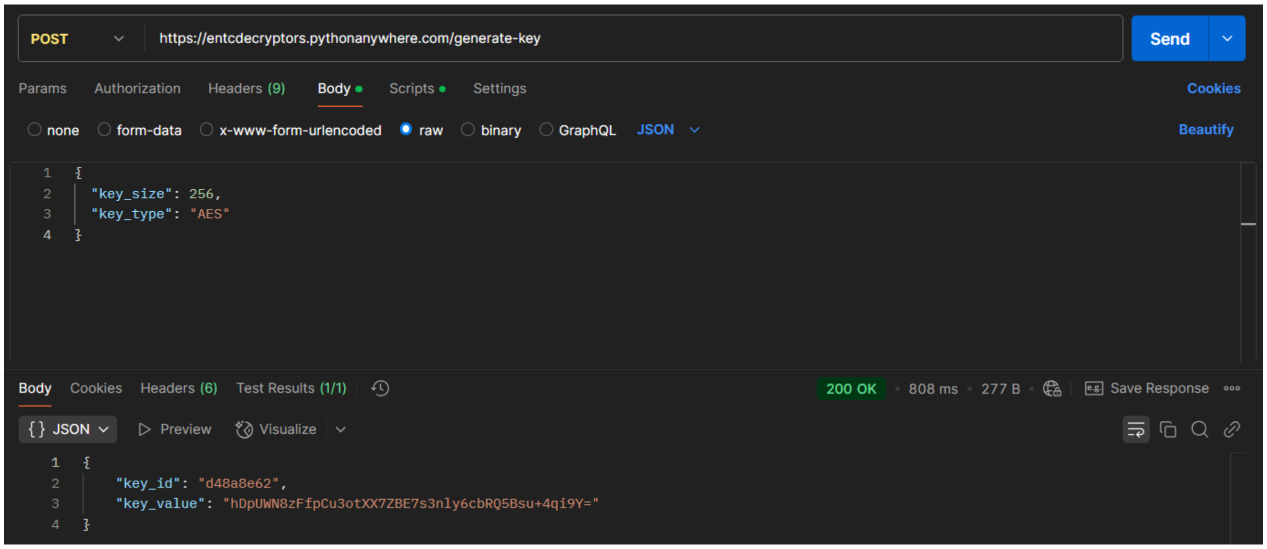
This API supports AES (Advanced Encryption Standard) for symmetric key encryption. It provides endpoints for key generation, encryption, and decryption. Features of the symmetric key cryptographic functions are as follows:

Key Sizes	$128, 192, 256$ bits
Mode of Operation	CBC (Cipher Block Chaining)
Padding Scheme	PKCS7 Padding
Initialization Vector (IV)	Random 16-byte IV unique for each encryption operation
Output Encoding Format	Base64 for keys and ciphertexts

2.1.1 Key Generation

For AES, the key generation endpoint performs the following steps:

- Validates the key size.
- Generates an random key of the specific size.
- Encodes the key in Base64 and stores it in the database with a unique 32-bit key ID.



POST https://entcdecryptors.pythonanywhere.com/generate-key

Body: raw JSON

```
{
  "key_size": 256,
  "key_type": "AES"
}
```

Body {} JSON

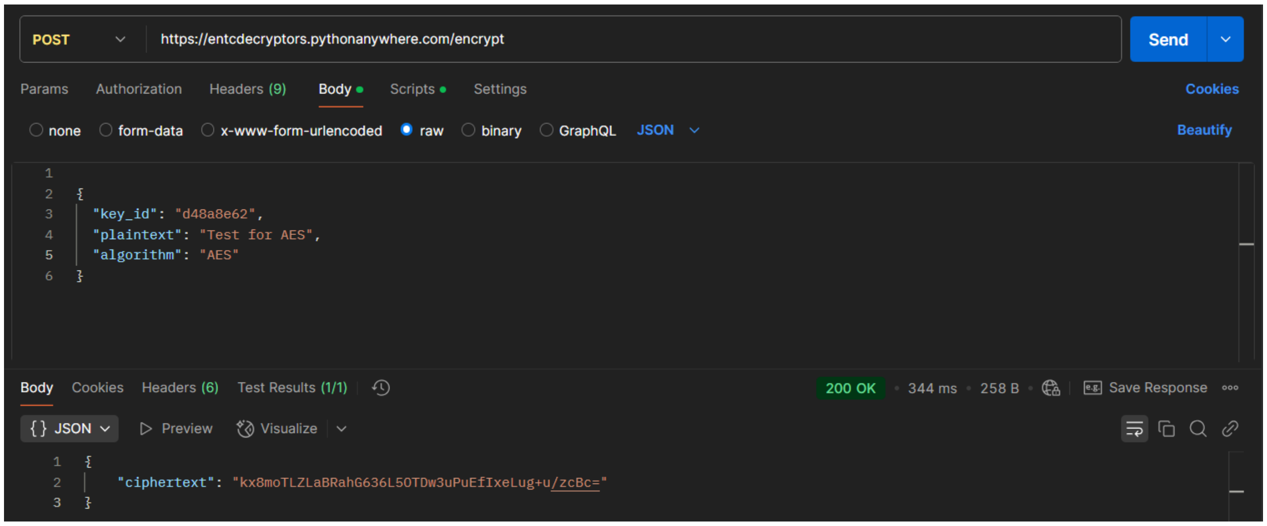
```
{
    "key_id": "d48a8e62",
    "key_value": "hDpUWN8zFfpCu3otXX7ZBE7s3nly6cbRQ5Bsu+4qi9Y="
}
```

Figure 1: AES key generation with key size 256

2.1.2 Encryption

The encryption endpoint performs the following steps:

- Retrieves the AES key corresponding to the key ID from the database.
- Pads the plaintext using PKCS7.
- Generates a random 16-byte IV and encrypts the padded data in CBC mode.
- Combines IV and ciphertext, encoding the result in Base64.



POST https://entcdecryptors.pythonanywhere.com/encrypt

Body: raw JSON

```

{
  "key_id": "d48a8e62",
  "plaintext": "Test for AES",
  "algorithm": "AES"
}
```

Body {} JSON

```
{
    "ciphertext": "kx8moTLZLaBRahG636L5OTDw3uPuEfIxeLug+u/zcBc="
}
```

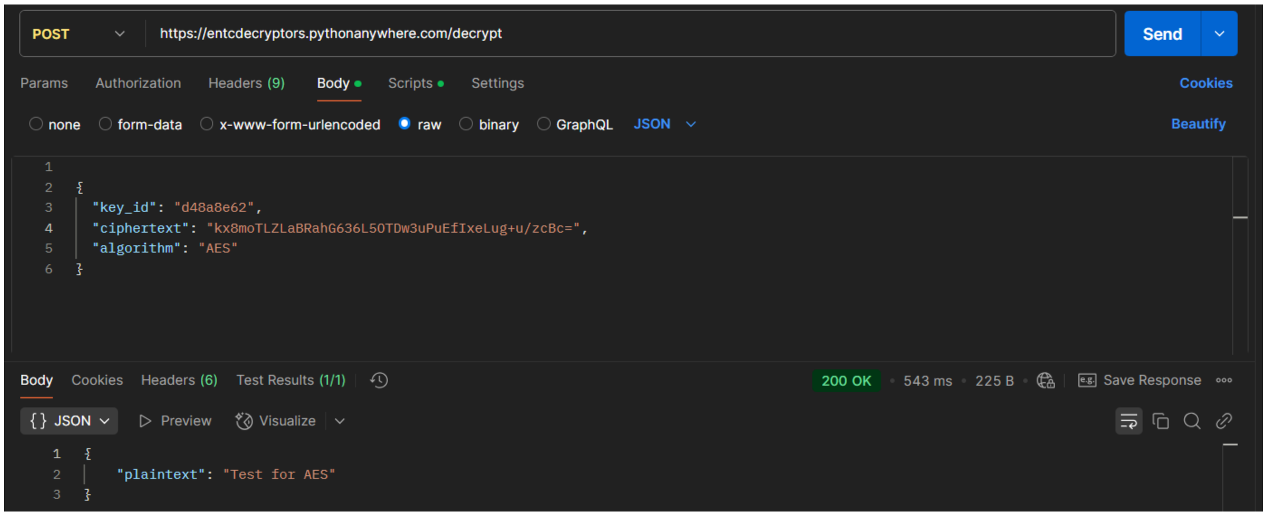
Figure 2: AES encryption

2.1.3 Decryption

The decryption endpoint performs the following steps:

- Retrieves the AES key corresponding to the key ID from the database.
- Decodes the Base64 ciphertext and extracts the IV.

- Decrypts the data in CBC mode and removes PKCS7 padding.
- Returns the original plaintext.



```

{
  "key_id": "d48a8e62",
  "ciphertext": "kx8moTLZLaBRahG636L5OTDw3uPuEfIxeLug+u/zcBc=",
  "algorithm": "AES"
}
```

```
{
    "plaintext": "Test for AES"
}
```

Figure 3: AES Decryption

2.2 Asymmetric Key Cryptography

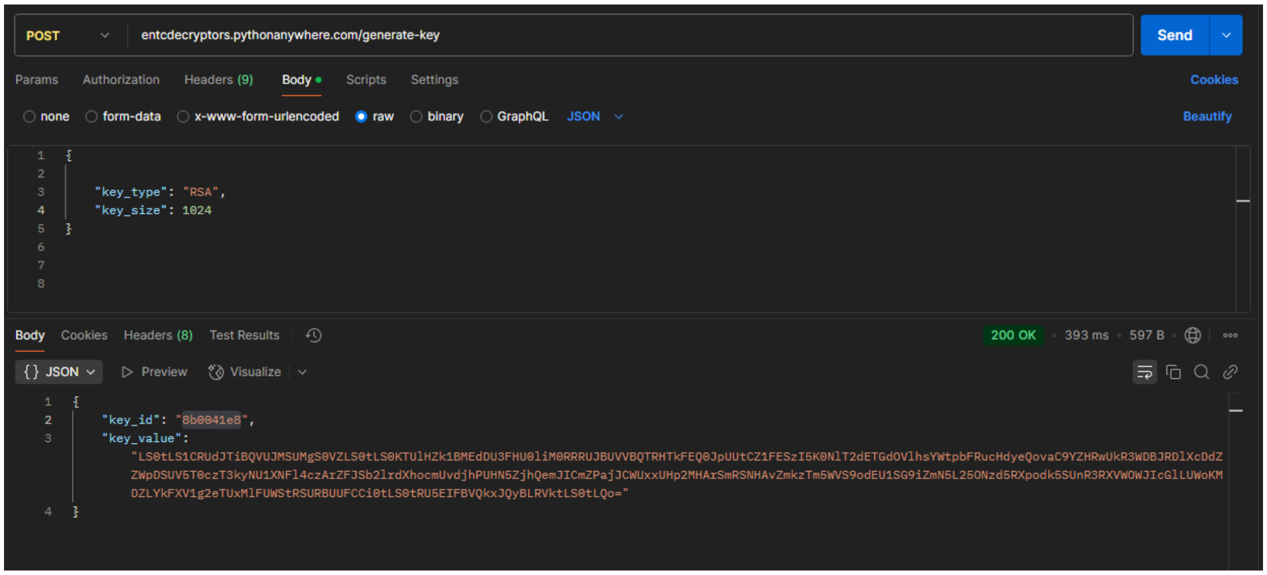
This API supports the RSA (Rivest–Shamir–Adleman) asymmetric key encryption algorithm for secure data transmission. It provides API endpoints for key generation, encryption, and decryption of messages using asymmetric key cryptography. Features of the cryptographic functions are as follows:

Key Sizes	1024, 2048, 3072, 4096 bits
Padding Scheme	OAEP (Optimal Asymmetric Encryption Padding)
Output Encoding Format	Base64 for keys and ciphertexts

2.2.1 Key Generation

For RSA, the key generation endpoint performs the following steps:

- Validates the key size.
- Generates an RSA public-private key pair using the `cryptography` library.
- Serializes the keys to PEM format and encodes them in Base64.
- Stores them in the database with a unique 32-bit key ID.



entcdecryptors.pythonanywhere.com/generate-key

```
{

    "key_type": "RSA",
    "key_size": 1024
}
```

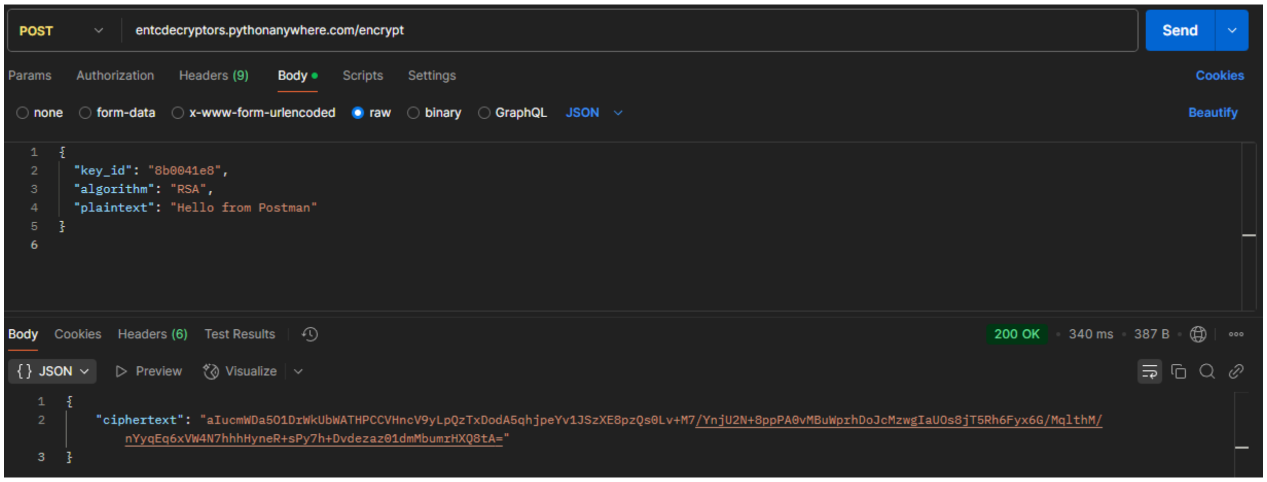
```
{
    "key_id": "8b0041e8",
    "key_value":
        "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlHZk1BMEdDU3FHU0liM0RRRUJBUVVBQTRHTkFEQ0JpUUtCZ1FESzI5K0NlT2dETGdOVlhsYWtpbFRucHdyeQovaC9YZHRwUkR3WDBJRDlXcDdZ
        ZWpDSUV5T0czT3kyNU1XNFl4czArZFJSb2lrdXhocmUvdjhPUHN5ZjhQemJICmZPajJCWUxxUHp2MHArSmRSNHAvZmkzTm5WVS9odEU1SG9iZmN5L25ONzd5RXpodk5SUnR3RXVWOWJIcGlLUWoKM
        DZLYkFXV1g2eTUxMlFUWStRSURBUUFCCi0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo="
}
```

Figure 4: RSA key generation for key size 1024

2.2.2 Encryption

The encryption endpoint performs the following steps:

- Retrieves the RSA public key corresponding to the key ID from the database.
- Deserializes the key from PEM format.
- Encrypts the plaintext using the public key and OAEP padding.
- Returns the encrypted message in Base64 format.



entcdecryptors.pythonanywhere.com/encrypt

```
{
  "key_id": "8b0041e8",
  "algorithm": "RSA",
  "plaintext": "Hello from Postman"
}
```

```
{
    "ciphertext": "aIucmWDa5O1DrWkUbWATHPCCVHncV9yLpQzTxDodA5qhjpeYv1JSzXE8pzQs0Lv+M7/YnjU2N+8ppPA0vMBuWprhDoJcMzwgIaUOs8jT5Rh6Fyx6G/MqlthM/
        nYyqEq6xVW4N7hhhHyneR+sPy7h+Dvdezaz01dmMbumrHXQ8tA="
}
```

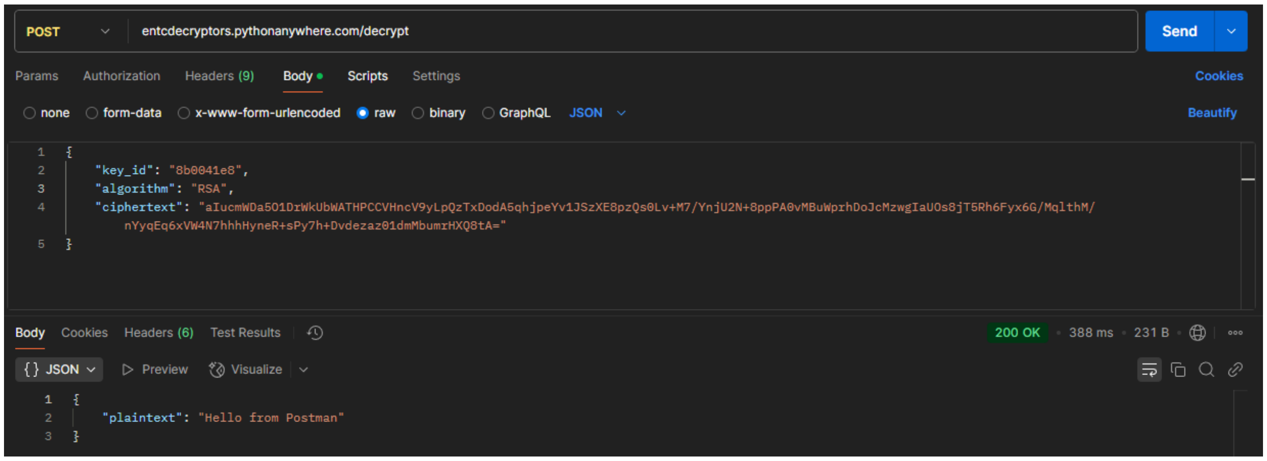
Figure 5: RSA Encryption

2.2.3 Decryption

The decryption endpoint performs the following steps:

- Retrieves the RSA private key corresponding to the key ID from the database.
- Deserializes the key from PEM format.

- Decrypts the ciphertext using the private key and OAEP padding.
- Returns the original plaintext.



entcdecryptors.pythonanywhere.com/decrypt

```
{
    "key_id": "8b0041e8",
    "algorithm": "RSA",
    "ciphertext": "aIucmWDa501DrWkUbWATHPCCVHncV9yLpQzTxDodA5qhjpeYv1JSzXE8pzQs0Lv+M7/YnjU2N+8ppPA0vMBuWprhDoJcMzwgIaUOs8jT5Rh6Fyx6G/MqlthM/
        nYyqEq6xVW4N7hhhHyneR+sPy7h+Dvdezaz01dmMbumrHXQ8tA="
}
```

```
{
    "plaintext": "Hello from Postman"
}
```

Figure 6: RSA Decryption

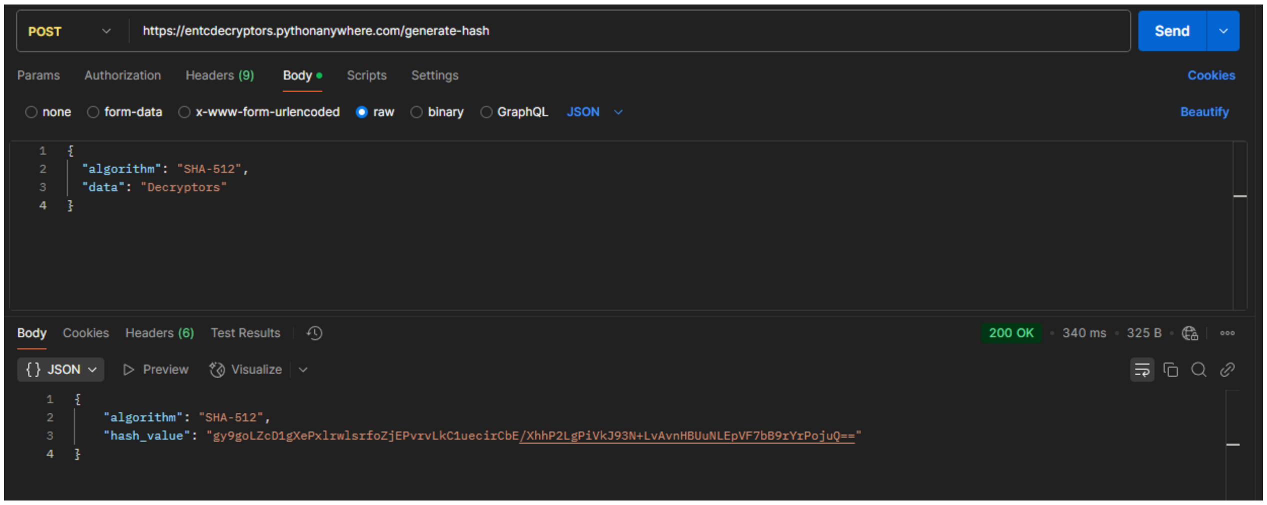
2.3 Hashing and Hash Verification

The API provides endpoints for generating and verifying hashes using various cryptographic hash functions. The supported algorithms are SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, MD5, BLAKE2b, and BLAKE2s.

2.3.1 Hash Generation

The hash generation endpoint performs the following steps:

- Validates that the given hashing algorithm is supported.
- Computes the hash for the message using the specified algorithm from the `cryptography` library.
- Converts the hash to Base64 encoding for portability.



https://entcdecryptors.pythonanywhere.com/generate-hash

```
{
    "algorithm": "SHA-512",
    "data": "Decryptors"
}
```

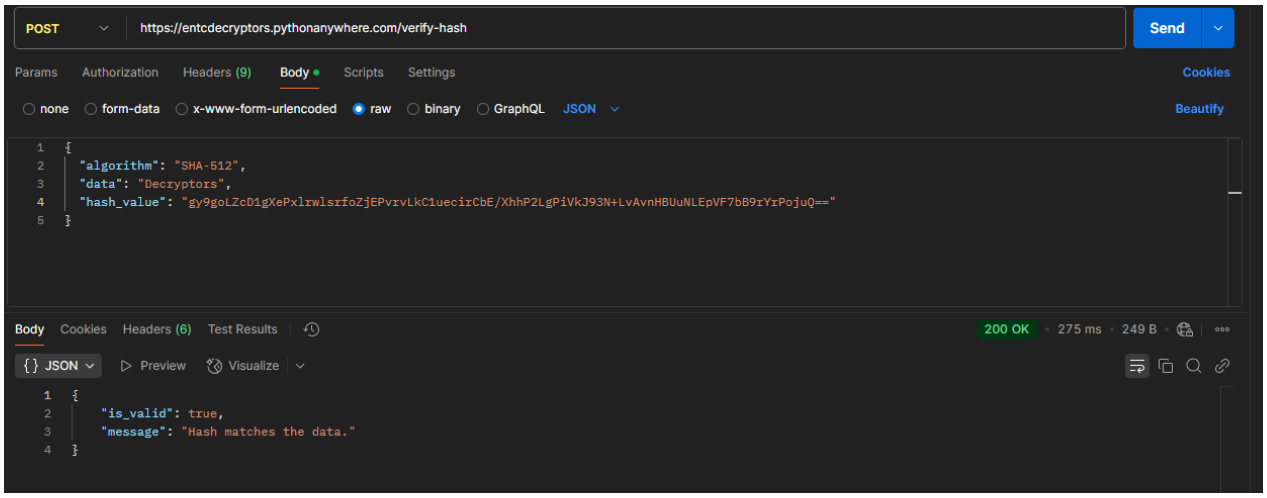
```
{
    "algorithm": "SHA-512",
    "hash_value": "gy9goLZcD1gXePxlrwlsrfoZjEPvrvLkC1uecirCbE/XhhP2LgPiVkJ93N+LvAvnHBUuNLEpVF7bB9rYrPojuQ=="
}
```

Figure 7: Hash generation (using SHA-512)

2.3.2 Hash Verification

The hash verification endpoint performs the following steps:

- Validates that the given hashing algorithm is supported.
- Computes the hash for the message using the specified algorithm from the `cryptography` library.
- Converts the hash to Base64 encoding and compares with the given hash.
- Returns whether the hash matches the message.



POST https://entcdecryptors.pythonanywhere.com/verify-hash

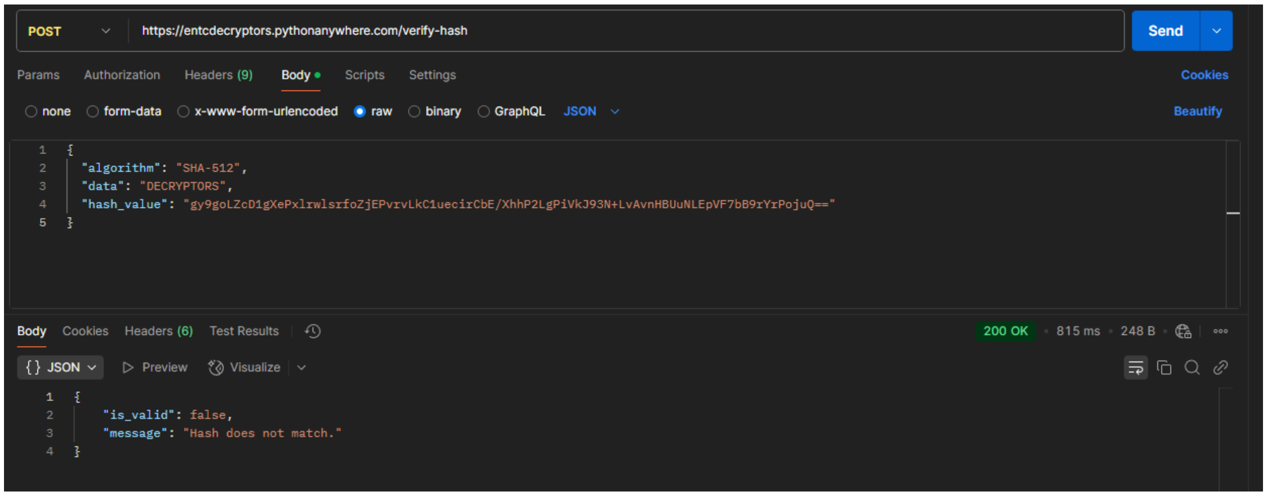
Body: raw, JSON

```
{
  "algorithm": "SHA-512",
  "data": "Decryptors",
  "hash_value": "gy9goLZcD1gXePxlrwlsrfoZjEPvrvLkC1uecirCbE/XhhP2LgPiVkJ93N+LvAvnHBUuNLEpVF7bB9rYrPojuQ=="
}
```

200 OK · 275 ms · 249 B

```
{
    "is_valid": true,
    "message": "Hash matches the data."
}
```

Figure 8: Hash verification where the hash matches the message



POST https://entcdecryptors.pythonanywhere.com/verify-hash

Body: raw, JSON

```
{
  "algorithm": "SHA-512",
  "data": "DECRYPTORS",
  "hash_value": "gy9goLZcD1gXePxlrwlsrfoZjEPvrvLkC1uecirCbE/XhhP2LgPiVkJ93N+LvAvnHBUuNLEpVF7bB9rYrPojuQ=="
}
```

200 OK · 815 ms · 248 B

```
{
    "is_valid": false,
    "message": "Hash does not match."
}
```

Figure 9: Hash verification where the hash does not match the message