Build a Simple Hedera Messaging Service

import asyncio

import datetime

import hashlib

import base64

from cryptography.fernet import Fernet # For encryption

from hedera import (

Client,

AccountId,

PrivateKey,

TopicId,

TopicCreateTransaction,

TopicMessageSubmitTransaction,

TopicMessageQuery,

Timestamp,

)

# Hedera account details (REPLACE WITH YOUR CREDENTIALS)

OPERATOR\_ID = AccountId.fromString("0.0.YOUR\_ACCOUNT\_ID") # Example: 0.0.12345

OPERATOR\_KEY = PrivateKey.fromString("YOUR\_PRIVATE\_KEY") # Example: 302e02010030021c021a00...

# Topic ID (Use the provided one or create a new one)

TOPIC\_ID = TopicId.fromString("0.0.34567") # Replace with your Topic ID

# Encryption setup (using Fernet for symmetric encryption)

ENCRYPTION\_KEY = Fernet.generate\_key() # Generate a strong key (store securely!)

f = Fernet(ENCRYPTION\_KEY)

async def create\_topic(client: Client) -> TopicId:

"""Creates a new Hedera topic."""

transaction = TopicCreateTransaction()

transaction.sign\_with\_operator(client)

response = await transaction.execute(client)

receipt = await response.get\_receipt(client)

new\_topic\_id = receipt.topic\_id

print(f"New topic created with ID: {new\_topic\_id}")

return new\_topic\_id

async def send\_message(client: Client, topic\_id: TopicId, message: str):

"""Sends a message to the Hedera topic."""

# Encryption

encrypted\_message = f.encrypt(message.encode()).decode() # Encode to bytes, encrypt, decode back to string

transaction = TopicMessageSubmitTransaction(topic\_id)

transaction.set\_message(encrypted\_message) # Send encrypted message

transaction.sign\_with\_operator(client)

response = await transaction.execute(client)

receipt = await response.get\_receipt(client)

print(f"Message sent at {receipt.consensus\_timestamp}")

async def receive\_messages(client: Client, topic\_id: TopicId, since: Timestamp = None):

"""Receives messages from the Hedera topic."""

if since is None:

since = Timestamp.now() # Start from the current time if no 'since' is provided

query = TopicMessageQuery(topic\_id)

query.set\_start\_time(since)

messages = []

async for message in query.execute(client):

# Decryption

decrypted\_message = f.decrypt(message.contents.encode()).decode() # Decode from string, decrypt, decode to string

messages.append(

{

"message": decrypted\_message,

"timestamp": message.consensus\_timestamp.strftime("%Y-%m-%d %H:%M:%S"),

}

)

return messages

async def main():

client = Client.for\_testnet() # Or Client.for\_mainnet()

client.set\_operator\_with(OPERATOR\_ID, OPERATOR\_KEY)

# Uncomment to create a new topic if needed (or use the provided one)

# TOPIC\_ID = await create\_topic(client)

await send\_message(client, TOPIC\_ID, "Hello, Hedera!")

await send\_message(client, TOPIC\_ID, "Learning HCS")

await send\_message(client, TOPIC\_ID, "Message 3")

# Receive messages (example: from 1 minute ago)

# since\_time = Timestamp.now() - datetime.timedelta(minutes=1)

received\_messages = await receive\_messages(client, TOPIC\_ID) # No 'since', gets all from now onwards

print("\nMessages Received:")

for msg in received\_messages:

print(f"- {msg['message']} at {msg['timestamp']}")

# Example of Message Filtering (Bonus Challenge):

print("\nFiltered Messages (containing 'Hedera'):")

for msg in received\_messages:

if "Hedera" in msg["message"]:

print(f"- {msg['message']} at {msg['timestamp']}")

if \_name\_ == "\_main\_":

asyncio.run(main())

**Example Output (Hedera Messaging Service):**

Message sent at 2024-10-27 14:30:00.123456789 (Timestamp will be different)

Message sent at 2024-10-27 14:30:01.987654321 (Timestamp will be different)

Message sent at 2024-10-27 14:30:02.555444333 (Timestamp will be different)

Messages Received:

- Hello, Hedera! at 2024-10-27 14:30:00 (Decrypted message and formatted timestamp)

- Learning HCS at 2024-10-27 14:30:01 (Decrypted message and formatted timestamp)

- Message 3 at 2024-10-27 14:30:02 (Decrypted message and formatted timestamp)

Filtered Messages (containing 'Hedera'):

- Hello, Hedera! at 2024-10-27 14:30:00