ACCÉLÉRER LA TRANSITION VERS LE DOSSIER MÉDICAL NUMÉRIQUE GRÂCE À LA DÉCENTRALISATION DES DONNÉES

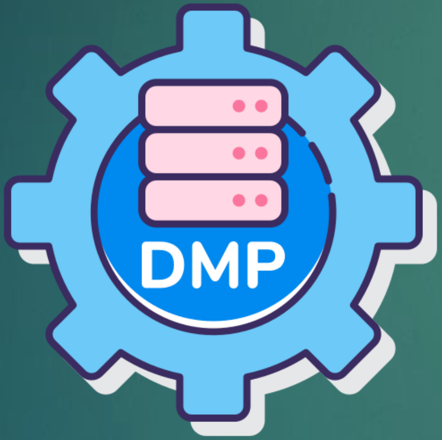
WERCK HUGO N°10512

BINÔME: CLAUDON AUBIN N°22595

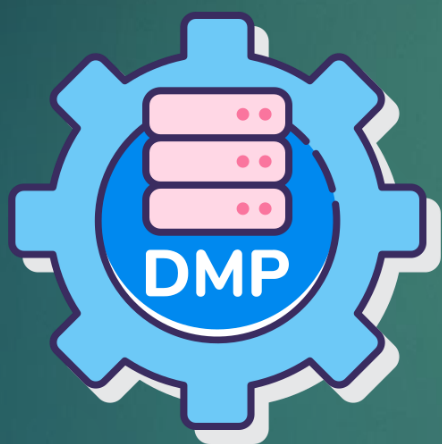
Table des matières

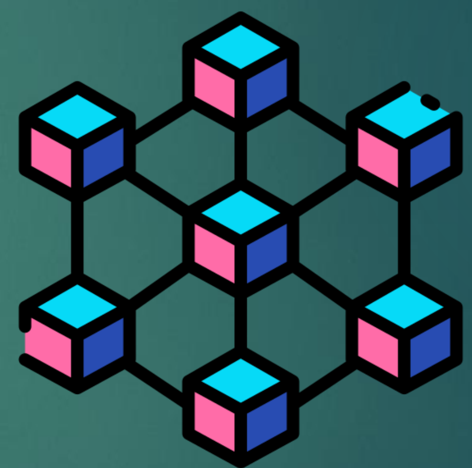
1. Introduction
2. Solution possible – la blockchain
3. Mise en place d'un réseau blockchain
4. Limites de la modélisation
5. Conclusion
6. Annexes

I) Introduction



I) Introduction





I) Introduction

- *Problématique retenue:*
- Il faut que le couple patient / médecin ait un accès confidentiel aux données médicales du sujet, éparpillées sur les différents serveurs sécurisés des professionnels de santé.
- L'enjeu est ici d'établir un réseau blockchain permettant un transfert sécurisé, entre médecin et spécialiste, des clés authentifiant le patient à son dossier médical.

II) Solution – la blockchain

- Avantages : immuable, sécurisée, décentralisation, autonomie, garantit l'intégrité et le transfert des données

II) Solution – la blockchain

- Avantages : immuable, sécurisée, décentralisation, autonomie, garantit l'intégrité et le transfert des données
- Inconvénients : données en clair, en quantité limitée

II) Solution – la blockchain

N°10512

II) Solution – la blockchain

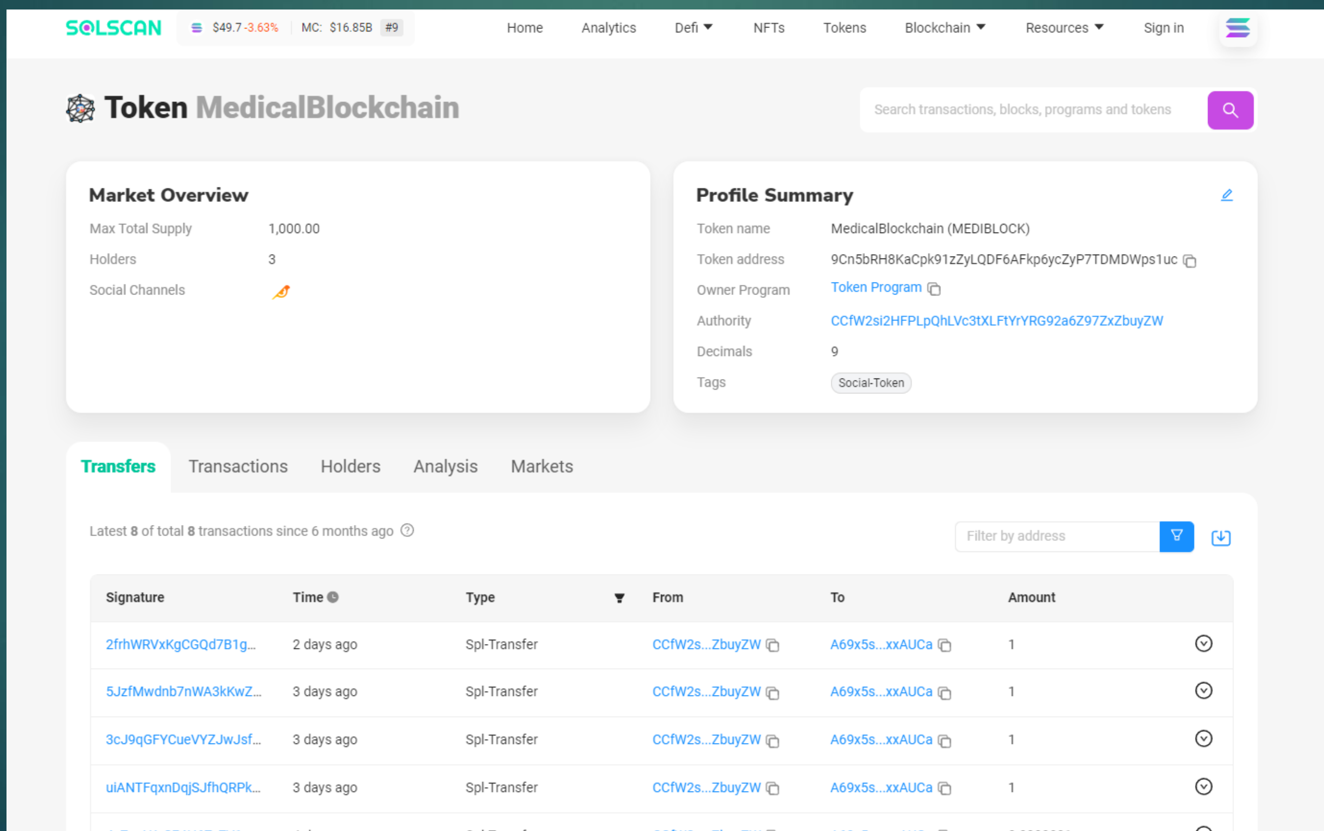
II) Solution – la blockchain

II) Solution – la blockchain




II) Solution – la blockchain

- Création d'un token hébergé par Solana



II) Solution – la blockchain

- Réseau blockchain codé en python

- `class Block:`
  - `Attributs:`
    - *`Index`*
    - *`Nonce`*
    - *`Data`*
    - *`Timestamp`*
  - `Calcul du hachage en SHA256`

II) Solution – la blockchain

- Réseau blockchain codé en python

- `class Blockchain:`
  - `Attributs:`
    - *Chain*
    - *Current_data*
    - *Nodes*
    - *Genesis_block()*

II) Solution – la blockchain

- class Blockchain:
  - Genesis_block()
  - Add_block()
  - Check_validity()
  - New_data()
  - Proof_of_work()
  - Verify_proof()
  - Last_block()
  - Block_mining()
  - New_node()

II) Solution – la blockchain

- Algorithme - Proof_of_work() :
  - Tant que verify_proof() est faux
    - Incrémenter nonce par pas de 1

II) Solution – la blockchain

- Algorithme – Verify_proof() :
  - Faire hachage de {dernier nonce-nouveau nonce}
  - Vérifier que ce hachage débute par le nombre de 0 imposé : la difficulté

II) Solution – la blockchain

- Algorithme – block_mining():
  - Ajouter des nouvelles données au réseau
  - Recalculer le hachage du dernier bloc
  - Vérifier que proof_of_work() est vraie pour ce hachage

III) Mise en place du réseau blockchain

- Comment faire contre cette quantité de données limitée et en clair ?

III) Mise en place du réseau blockchain

- Comment faire contre cette quantité de données limitée et en clair ?

Transfert de fichiers lourds en pair à pair et d'une clé sécurisée par la blockchain

III) Mise en place du réseau blockchain

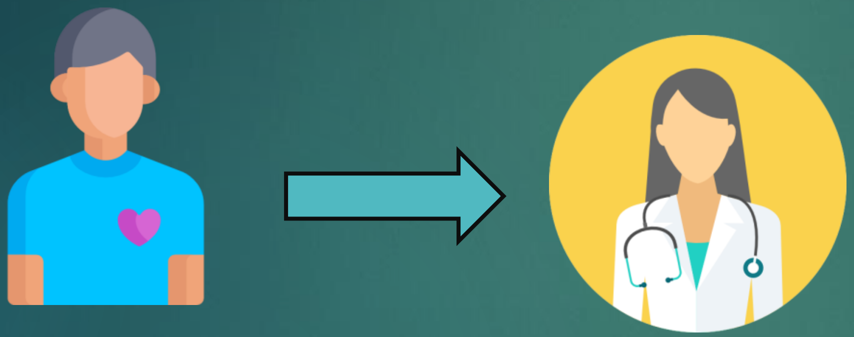
- Chiffrement symétrique (AES):

→ Transfert de fichiers en pair à pair

- Chiffrement asymétrique (RSA):

→ Clé sécurisée associée au fichier transféré

III) Mise en place du réseau blockchain



III) Mise en place du réseau blockchain

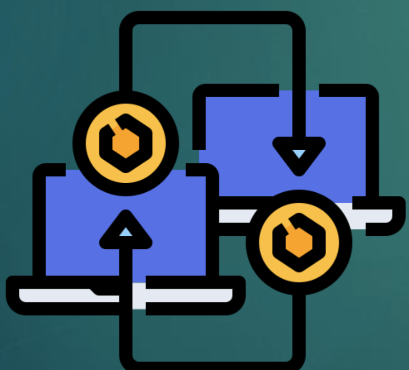
III) Mise en place du réseau blockchain

Nom
Clé publique
Numéro portefeuille

III) Mise en place du réseau blockchain

Données : IPP,
IP médecin, clé
publique médecin

Chiffrage asymétrique



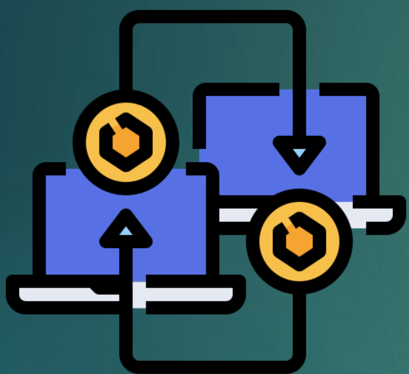
III) Mise en place du réseau blockchain

Partie de mon binôme

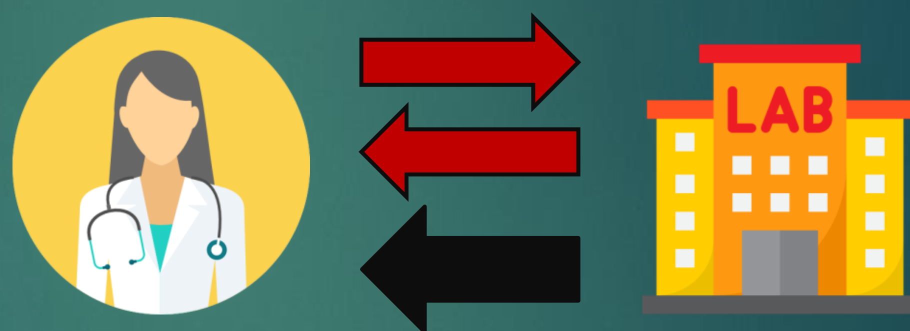
III) Mise en place du réseau blockchain

Partie de mon binôme

III) Mise en place du réseau blockchain



<u>Retour</u> : clé symétrique chiffrée asymétriquement



III) Mise en place du réseau blockchain

III) Mise en place du réseau blockchain

- Base de données accessible au médecin

cle	nom	portefeuille
Filtre	Filtre	Filtre
48271	Médecin	Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dS...
2147483647	Laboratoire	9xcCpz4Y3BVSFiM7ZWZvS3uT5tUo9xsqpLJadCm...

III) Mise en place du réseau blockchain

- Demande d'information sous python

Instruction Details

#1 - Token Transfer

Interact With	Token Program - TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA
Input Accounts	#1 - Source - Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi Writable
	#2 - Destination - AfzPooLwjpmhjjo7KR44fSivhwuxCsgHiEPAuhNgakvw Writable
	#3 - Authority - CCfW2si2HFPLpQhLVc3tXLFtYrYRG92a6Z97ZxZbuyZW Writable Signer Fee Payer
	#4 - Amount - 1 MEDIBLOCK
	#5 - Mint - 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc

#2 - Unknown

Interact With	Memo Program V2 - MemoSq4gqABAXKb96qnH8TysNcWxMyWCqXgDLGmfcHr
Instruction Data	{'minstd': 2147483647, 'data': b'\x9ac\xc1\xae\x1f7e\xef%\xd3n!n...x0e\xab\x9dY\xa4Ia\xd69?'}
Input Accounts	

III) Mise en place du réseau blockchain

- Envoi de la clé symétrique sous python

Instruction Details

#1 - Token Transfer

Interact With	Token Program - TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA
Input Accounts	#1 - Source - AfzPooLwjpmhjjo7KR44fSivhwuxCsgHiEPAuhNgakvw Writable
	#2 - Destination - Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi Writable
	#3 - Authority - CCfW2si2HFPLpQhLVc3tXLFtYrYRG92a6Z97ZxZbuyZW Writable Signer Fee Payer
	#4 - Amount - 1 MEDIBLOCK
	#5 - Mint - 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc

#2 - Unknown

Interact With	Memo Program V2 - MemoSq4gqABAXKb96qnH8TysNcWxMyWCqXgDLGmfcHr
Instruction Data	{'minstd': 2147483647, 'data': b'sE\x08\x04\x92C&\x8f}{k$\...\xfc\xc7Ca\x9bu\x8b\x00\x0fH'}
Input Accounts	

III) Mise en place du réseau blockchain

- Lien des deux parties:
- Méthodes :
  - Request_info()
  - Decrypt_info()
  - Send_symmetric_key()
  - Send_transaction()
  - Read_transaction()
  - Last_minstd()

III) Mise en place du réseau blockchain

- Transaction **aller :** send_transaction()
- Message : request_info()
- *objetRequete*DEL*ipp*DEL*rpps*DEL*ipMedecin *SEP*signatureMedecin**

III) Mise en place du réseau blockchain

- Transaction **aller :** send_transaction()
- Message : request_info()
- *objetRequete*DEL*ipp*DEL*rpps*DEL*ipMedecin*SEP*signatureMedecin**
- Encrypté asymétriquement par le médecin avec ajout d'une signature RSA

III) Mise en place du réseau blockchain

- Transaction **aller :** send_transaction()
- Message : request_info()
- *objetRequete*DEL*ipp*DEL*rpps*DEL*ipMedecin *SEP*signatureMedecin**
- Encrypté asymétriquement par le médecin avec ajout d'une signature RSA
- Requête lue avec read_transaction() : dans dernières transactions non lues, décryptage avec decrypt_info() du message par spécialiste

III) Mise en place du réseau blockchain

- Transaction **retour :** send_transaction()
- Message : send_symmetric_key()
- *retour*DEL*cléSymetrique*SEP*signatureSpécialiste*
- Encrypté asymétriquement par le spécialiste avec ajout d'une signature RSA
- Requête lue avec read_transaction() : dans dernières transactions non lues, décryptage avec decrypt_info() du message par médecin

III) Mise en place du réseau blockchain

- Transaction **aller**

```
Index: 0
Nonce: 0
Previous hash: 0
Data: []
Timestamp: 1654590832.5868871
,
Index: 1
Nonce: 69732
Previous hash: 367c280ec7e805f089772e5ea0ef94eba79480c9470cc1e75e9e59b3f5163688
Data: [{'sender': 'doctor_wallet', 'recipient': 'specialist_wallet', 'quantity': 1, 'message': {'minstd': 2147483647, 'data': b'%\x94\x9e\x99\xe9x
Timestamp: 1654590832.7038894
]


MESSAGE DECRYPTE PAR LE SPECIALISTE: ['Demande analyse sanguine de M.Dupont', '2147483647', '48271', '127.0.0.1']
```

III) Mise en place du réseau blockchain

- Transaction **retour**

```
Index: 0
Nonce: 0
Previous hash: 0
Data: []
Timestamp: 1654590832.5868871
,
Index: 1
Nonce: 69732
Previous hash: 367c280ec7e805f089772e5ea0ef94eba79480c9470cc1e75e9e59b3f5163688
Data: [{'sender': 'doctor_wallet', 'recipient': 'specialist_wallet', 'quantity': 1, 'message': {'minstd': 2147483647, 'data': b'%\x94\x9e\x99\xe9x\xa1a`\t\x1
Timestamp: 1654590832.7038894
,
Index: 2
Nonce: 23263
Previous hash: 1ec22b731a889d09c67c3e54cd0034db26ce53ac71210bf6dffee238db880f03
Data: [{'sender': 'specialist_wallet', 'recipient': 'doctor_wallet', 'quantity': 1, 'message': {'minstd': 2147483647, 'data': b'\xdey\x9d\xeb\xd8\xa63:\x0007
Timestamp: 1654590832.7538576
]


MESSAGE DECRYPTE PAR LE MEDECIN: ['retour', "b'\\xa3^\\x11\\x1a%F\\xde\\xa4\\xdfK\\xbd;\\xb7\\xf1\\xa1-e%h'", "b'\\x03\\xa9\\xd50\\xb0\\xc8-I\\x97\\xfb\\xa4\
```

III) Mise en place du réseau blockchain

- Transfert du fichier encrypté symétriquement

Analyses M.Dupont - Bloc-notes

Fichier Edition Format Affichage Aide

```
*Resultats de l'analyse sanguine de M.Dupont*
```

```
*Resultats de l'analyse sanguine de M.Dupont*
----------
b'_lp\xda\x80C2D\xd2\xda\x01\xb8Q%vCz\x0c\xab\x9dm\x12%\x94%BH\x06\xef\xae\x82\xf0mT\xd8/\
----------
*Resultats de l'analyse sanguine de M.Dupont*
```

IV) Limites de la modélisation

- Réseau à échelle locale

IV) Limites de la modélisation

- Réseau à échelle locale
- Temps requis pour que les instances médicales rejoignent le réseau

IV) Limites de la modélisation

- Réseau à échelle locale
- Temps requis pour que les instances médicales rejoignent le réseau
- Choix modélisation : payant pour patient

IV) Limites de la modélisation

- Réseau à échelle locale
- Temps requis pour que les instances médicales rejoignent le réseau
- Choix modélisation : payant pour patient
- Algorithmes de validation de la blockchain encore à optimiser

V) Conclusion

- Notre solution - une application possible de cette nouvelle technologie révolutionnaire

V) Conclusion

- Notre solution - une application possible de cette nouvelle technologie révolutionnaire
- Apports du TIPE:

V) Conclusion

- Notre solution - une application possible de cette nouvelle technologie révolutionnaire
- Apports du TIPE:
  - Projet informatique d'une plus grande ampleur

V) Conclusion

- Notre solution - une application possible de cette nouvelle technologie révolutionnaire
- Apports du TIPE:
  - Projet informatique d'une plus grande ampleur
  - Travail de groupe

V) Conclusion

- Notre solution - une application possible de cette nouvelle technologie révolutionnaire
- Apports du TIPE:
  - Projet informatique d'une plus grande ampleur
  - Travail de groupe
  - Cahier des charges et temps imparti à respecter

VI) Annexes – sources des images

- Icones https://www.flaticon.com/fr/

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```
def _createDataBase(self, dbName: str) -> (object,object):
    """
    Create a database, or open it if already existing

    :param str dbName: name of the database
    :return: c the cursor and conn the connection
    :rtype: tuple of cursor and connection sqlite objects
    """
    conn = sqlite3.connect(dbName)
    c = conn.cursor()
    return c, conn
```

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```python
def _createTable(self, c: object, conn: object, tableName: str) -> None:
    """
    Create a table in given databse, or raise an error

    :param cursor c: cursor of the given database
    :param connection conn: connection of the given databse
    :param str tableName: name of the table
    :raise: Error if table already exists
    """
    try:
        with conn:
            comm = "CREATE TABLE " + tableName + " (cle integer, nom text, portefeuille text)"
            # this syntax prevents SQL injections
            c.execute(comm)

    except Exception as e:
        print(e)
```

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```python
def _insertValue(self, c: object, conn: object, tableName: str, publicKey: str, personName: str, wallet: str) -> None:
    """
    Insert value into given table

    :param cursor c: cursor
    :param connection conn: connection
    :param str tableName: name of the table
    :param publicKey: public RSA key of the person
    :param personName: person's name
    :param str wallet: wallet of the person for the given token of blockchain
    """
    with conn:
        comm = "INSERT INTO " + tableName + " VALUES ( '" + publicKey + "', '" + personName + "', '" + wallet + "' )"
        c.execute(comm)
```

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```
def _showTable(self, c: object, conn: object, tableName: str) -> None:
    """
    Show the given table

    :param object c: cursor
    :param object conn: connection
    :param str tableName: name of the table
    """
    with conn:
        comm = "SELECT * FROM " + tableName
        c.execute(comm)
        print(c.fetchall())
```

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```
def _deleteRow(self, c: object, conn: object, tableName: str, nameToDelete: str) -> None:
    """
    Delete one row for a given name

    :param object c: cursor
    :param object conn: connection
    :param str tableName: name of the table
    :param str nameToDelete: name to delete
    """
    with conn:
        comm = "DELETE FROM '" + tableName + "' WHERE nom = '" + nameToDelete + "'"
        c.execute(comm)
```

VI) Annexes – création base de données

- **Cryptographie/code/src/data_manager.py**

```
if __name__ == "__main__":
    initializeLogger()

    try:
        manager = DataManager()

        # TEST DATABASE
        c, conn = manager.createDataBase("ListeSpecialistes.db")
        manager.createTable(c, conn, "Liste")

        manager.insertValue(c, conn, "Liste", 48271, "Médecin", "Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi")
        manager.insertValue(c, conn, "Liste", 2147483647, "Laboratoire", "9xcCpz4Y3BVSFiM7ZWZvS3uT5tUo9xsqpLJadCmEAM7Y")
        manager.showTable(c, conn, "Liste")

        manager.deleteRow(c, conn, "Liste", "Patient")
        manager.showTable(c, conn, "Liste")

        conn.close()


    except Exception as e:
        log.error(e)
```

VI) Annexes – mise en place de Solana

- Prérequis:

```
root@DESKTOP-68774F0:~# sh -c "$(curl -sSfL https://release.solana.com/v1.10.17/install)"
downloading v1.10.17 installer
```

VI) Annexes – mise en place de Solana

- Création d'une paire de clés
- *solana-keygen new*

```
BIP39 Passphrase (empty for none):
Wrote new keypair to /home/hugow/.config/solana/id.json
================================================================================
pubkey: CCfW2si2HFPLpQhLVc3tXLFtYrYRG92a6Z97ZxZbuyZW
================================================================================
Save this seed phrase and your BIP39 passphrase to recover your new keypair:
```

VI) Annexes – mise en place de Solana

- Création d'un token hébergé par Solana

```
hugow@DESKTOP-68774F0:~$ spl-token create-token
Creating token 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc

Signature: 3zZwNfrjqYynFm4ud7fW9Mcr1qr8NG1XAEPGvZUiLRa6kdkVPmmXE4MEYdKqCWD
vDpLKBvRnoNTfG5remzZtmnQp

hugow@DESKTOP-68774F0:~$
```

VI) Annexes – mise en place de Solana

- Création d'un compte pour ce token, associé à la paire de clés précédente

```
hugow@DESKTOP-68774F0:~$ spl-token create-account 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc
Creating account Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi

Signature: 3EJ67Do1xDakvj8rvXUs6ac9fxqxQLyihfEy9uke8XH7NoMc2WMpRfJet8W9BodL7ZKYWvjsLFsfZNQEtDUDiauN
```

VI) Annexes – mise en place de Solana

- Ajout de fonds à ce compte

```
hugow@DESKTOP-68774F0:~$ spl-token mint 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc 1000 Chup9he48sSAuPKG
F7M2eXNVJXYbxDWFnrC6dSbDqkgi
Minting 1000 tokens
  Token: 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc
  Recipient: Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi

Signature: 5Dd9N2LUMRWyuVdBxLrjxCstTxQgAbQFXarzAHcz9H7qrKrVWCbcxBVGnDWi6XEFHvM9XBQBjmGZXDM6kGpnf2xN
```

VI) Annexes – mise en place de Solana

- Première transaction grâce à l'invite de commandes

```
hugow@DESKTOP-68774F0:~$ solana balance
0.20147912 SOL
hugow@DESKTOP-68774F0:~$ spl-token transfer --fund-recipient 9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc
200 EEos62nKnBxwxyxhisTs22jZpa4TArcP4ddemC2RPSEq
Transfer 200 tokens
  Sender: Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi
  Recipient: EEos62nKnBxwxyxhisTs22jZpa4TArcP4ddemC2RPSEq
  Recipient associated token account: 9xcCpz4Y3BVSFiM7ZWZvS3uT5tUo9xsqpLJadCmEAM7Y
  Funding recipient: 9xcCpz4Y3BVSFiM7ZWZvS3uT5tUo9xsqpLJadCmEAM7Y (0.00203928 SOL)

Signature: 3xYvYTTcdxnsMGc4Js1HSmTTmjFwpGyyNcRnxnhgRifJ4Xmge3jgunb5GkLw6gneZqnwt1pdN2xdH7HAeebEhkqK

hugow@DESKTOP-68774F0:~$ solana balance
0.19943484 SOL
```

VI) Annexes – mise en place de Solana avec python


```python
import os

# /!\ Having Solana CLI installed is required

def send_transaction(token: str, quantity: float, recipient: str, memo: str) -> object:
    """
    Send a transaction through the Solana blockchain, for a desired token and with a message

    :param str token: desired token
    :param float quantity: quantity of token to send
    :param str recipient: receiver of the transaction
    :param str memo: message to attach to the transaction
    :return: transaction request through the command line
    :rtype: exit status on Linux, returned value by shell on Windows
    """

    return os.system(f'spl-token transfer {token} {quantity} {recipient} --fund-recipient --with-memo "{memo}"')
```


```
{'jsonrpc': '2.0', 'result': 'QALJKEbzVhSe7SzvXVtFENbWdacMGUnsJ9w4woPmbghzYPZoHRfcqm8wGWdiS7GUktiSrR7HVt8sGGGDuiq32zF', 'id': 5}
{'jsonrpc': '2.0', 'result': {'context': {'slot': 134134697}, 'value': 2039280}, 'id': 32}
{'jsonrpc': '2.0', 'result': {'context': {'slot': 134134697}, 'value': 2039280}, 'id': 33}
{'jsonrpc': '2.0', 'result': {'context': {'slot': 134134697}, 'value': 197375560}, 'id': 34}

Process finished with exit code 0
```

VI) Annexes – blockchain en Python

- **Blockchain/solana_blockchain.py**

```python
import json

# pip install solana

from spl.token.constants import TOKEN_PROGRAM_ID
from spl.token.instructions import transfer_checked, TransferCheckedParams

from solana.rpc.commitment import Confirmed
from solana.rpc.api import Client
from solana.rpc.types import TxOpts
from solana.keypair import Keypair
from solana.publickey import PublicKey
from solana.transaction import Transaction


client = Client(endpoint="https://api.mainnet-beta.solana.com", commitment=Confirmed)
```

VI) Annexes – blockchain en Python

- **Blockchain/solana_blockchain.py**

```
# import here your id.json file containing your keypair (mine is hidden)
# or generate a new one using Keypair()
f = open('id_auth.json')
secret_key = json.load(f)
f.close()
owner = Keypair.from_secret_key(bytes(secret_key))

# Creation a transaction object, then addition of parameters for our request
transaction = Transaction()

transaction.add(
   transfer_checked(
      TransferCheckedParams(
   program_id=TOKEN_PROGRAM_ID,
   source=PublicKey("Chup9he48sSAuPKGF7M2eXNVJXYbxDWFnrC6dSbDqkgi"),
   mint=PublicKey("9Cn5bRH8KaCpk91zZyLQDF6AFkp6ycZyP7TDMDWps1uc"),
   dest=PublicKey("AfzPooLwjpmhjjo7KR44fSivhwuxCsgHiEPAuhNgakvw"),
   owner=PublicKey("CCfW2si2HFPLpQhLVc3tXLFtYrYRG92a6Z97ZxZbuyZW"),
   amount=100,
   decimals=9,
   signers=[])))

print(client.send_transaction(transaction, owner, opts=TxOpts(skip_confirmation=False,
preflight_commitment=Confirmed)))
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```
from time import time
from hashlib import sha256


class Block:

    def __init__(self, index: int, nonce: int, prev_hash: str, data: list, timestamp=None) -> None:
        """
        Initialisation of the Block class

        :param int index: index of the block
        :param int nonce: number of tries to find a hash that satisfies the difficulty
        :param str prev_hash: previous block hash
        :param list data: attached data to the block
        :param float timestamp: given timestamp or generated one with time module
        """

        self.index = index
        self.nonce = nonce
        self.prev_hash = prev_hash
        self.data = data
        self.timestamp = timestamp or time()
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
@property
def hash_calculation(self) -> str:
    """
    Calculation of a SHA256 hash

    :return: hash of the block
    :rtype: str
    """

    str_block = f'{self.index}{self.nonce}{self.prev_hash}{self.data}{self.timestamp}'
    return sha256(str_block.encode()).hexdigest()


def __repr__(self) -> str:
    """
    Printable presentation of one block

    :return: presentation of a Block
    :rtype: str
    """
    return f'\nIndex: {self.index}\nNonce: {self.nonce}\nPrevious hash: {self.prev_hash}\nData: {self.data}' \
        f'\nTimestamp: {self.timestamp}\n'
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
class Blockchain:
   global difficulty
   difficulty = 4

   def __init__(self) -> None:
      """
      Initialization of the Blockchain class
      """

      self.chain = []
      self.current_data = []
      self.nodes = {}
      self.genesis_block()


   def genesis_block(self) -> None:
      """
      First block added to the blockchain, with arbitrary value of 0 for both nonce and previous hash
      """

      self.add_block(nonce=0, prev_hash='0')
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
def add_block(self, nonce: int, prev_hash: str) -> Block:
    """
    Addition of a block to the blockchain

    :param int nonce: number of tries to find a hash that satisfies the difficulty
    :param str prev_hash: previous block hash
    :return: the block added
    :rtype: Block
    """

    block = Block(
        index=len(self.chain),
        nonce=nonce,
        prev_hash=prev_hash,
        data=self.current_data)
    self.current_data = []

    self.chain.append(block)
    return block
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
@staticmethod
def check_validity(prev_block: Block, block: Block) -> bool:
    """
    Check the validity of 2 given blocks according to their hash, timestamp, proof and index

    :param Block prev_block: previous block
    :param Block block: new block
    :return: True if blockchain is valid, False otherwise
    :rtype: bool
    """

    if prev_block.hash_calculation != block.prev_hash:
        return False

    elif block.timestamp <= prev_block.timestamp:
        return False

    elif not Blockchain.verify_proof(prev_block.nonce, block.nonce):
        return False

    elif prev_block.index + 1 != block.index:
        return False

    return True
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
def new_data(self, sender: str, recipient: str, quantity: float, message: str) -> bool:
    """
    Attach new data to the current one

    :param str sender: sender of the transaction
    :param str recipient: receiver of the transaction
    :param float quantity: quantity of tokens to send
    :param str message: message attached to the transaction
    :return: True
    :rtype: bool
    """

    self.current_data.append({
        'sender': sender,
        'recipient': recipient,
        'quantity': quantity,
        'message': message})

    return True
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
@staticmethod
def proof_of_work(last_nonce: int) -> int:
    """
    Proof of work algorithm : count the attempts to verify the proof with the nonce variable

    :param int last_nonce: previous number of tries required to find the hash
    :return: nonce
    :rtype: int
    """

    nonce = 0

    while not Blockchain.verify_proof(last_nonce, nonce):
        nonce += 1

    return nonce
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
@staticmethod
def verify_proof(last_nonce: int, nonce: int) -> int:
    """
    Verify that the given hash match the desired one, with the difficulty required

    :param int last_nonce: previous nonce
    :param int nonce: current nonce
    :return: True if sha256 of last_nonce & nonce match the difficulty required, False otherwise
    :rtype: bool
    """

    to_find = f'{last_nonce}{nonce}'.encode()
    hash_to_find = sha256(to_find).hexdigest()

    return hash_to_find[:difficulty] == '0'*difficulty
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```
@property
def last_block(self) -> Block:
    """
    Return the last block of the chain

    :return: last block of the chain
    :rtype: Block

    """
    return self.chain[-1]
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
def block_mining(self, miner_details: str) -> Block:
    """
    Add a new block with the given miner details when validations are completed

    :param str miner_details: details of the miner
    :return: the last block when validations are completed
    :rtype: Block
    """

    self.new_data(
        sender='0', # chosen value 0 for a new block
        recipient=miner_details,
        quantity=1, # arbitrary value of 1
        message='***Mining new block***')

    last_block = self.last_block
    last_nonce = last_block.nonce
    last_hash = last_block.hash_calculation

    nonce = self.proof_of_work(last_nonce)
    block = self.add_block(nonce, last_hash)

    return block
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
def new_node(self, address:  str) -> bool:
    """
    Add a new node address

    :param str address: address of the new node
    :return: True
    :rtype: bool
    """

    self.nodes.add(address)
    return True
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```python
if __name__ == '__main__':

    '''Let's test our functions:
    -at first, we create a blockchain with 1 node, the genesis block and add 1 block
    -then, we mine another one'''

    global difficulty
    difficulty = 4  # number of 0 required at the beginning of the hash
    print('Difficulty:', difficulty)


    # test 1
    blockchain = Blockchain()
    blockchain.nodes = {'PC 1', 'PC 2', 'PC 3'}
    print('Nodes:', blockchain.nodes, '\n')
    print('Initial blockchain:\n', blockchain.chain)

    prev_block = blockchain.last_block
    last_nonce = prev_block.nonce
    nonce = blockchain.proof_of_work(last_nonce)

    blockchain.new_data(
        sender='PC 1',
        recipient='PC 2',
        quantity=1,
        message='First Test')
```

VI) Annexes – blockchain en Python

- **Blockchain/python_blockchain.py**

```
last_hash = prev_block.hash_calculation
block = blockchain.add_block(nonce, last_hash)

print('\nBlockchain with 1 validated block:\n', blockchain.chain)
print('\nValidity of the blockchain:', blockchain.check_validity(prev_block, block))


# test 2
blockchain.new_data(
    sender='PC 2',
    recipient='PC 3',
    quantity=2,
    message='Second Test')

print('\n\nData to mine:', blockchain.current_data)

block = blockchain.block_mining('PC 3')
print("Let's directly use the block_mining() function:", block)
print("Finally, our blockchain:\n", blockchain.chain)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
import python_blockchain
from cryptographie.code.src.encryption import *


def request_info(minstd: int, requestObject: str, ipp: int, rpps: int, ip: str,
                 recipientPublicKey: object, senderPrivateKey: object) -> dict:
    """
    Give a dictionary with minstd and encrypted message.

    :param minstd: last minstd number generated
    :param requestObject: object of the request
    :param ipp: unique patient id (=identifiant permanent du patient)
    :param rpps: unique specialist id (= répertoire partagé des professionnels de santé)
    :param ip: ip address of the sender
    :param recipientPublicKey: RSA public key of the recipient
    :param senderPrivateKey: RSA private key of the sender
    :return: dictionary which serves as message through the blockchain transaction
    :rtype: dictionary
    """

    data = f'{requestObject}*DEL*{ipp}*DEL*{rpps}*DEL*{ip}'
    signature = RSASignature(data, senderPrivateKey)
    message = asymmetricRSAEncryption(data.encode(), recipientPublicKey) + b"*SEP*" + signature

    return {'minstd': minstd, 'data': message}
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
def decrypt_info(data: bytes, senderPublicKey: object, recipientPrivateKey: object) -> list:
    """
    Decrypt the given message.

    :param data: given message
    :param senderPublicKey: public key of the sender
    :param recipientPrivateKey: private key of the recipient
    :return: decrypted message if valid signature, else value error
    :rtype: list
    :raise: error if RSA signature isn't valid
    """
    dataReceived, signatureReceived = data.split(b"*SEP*")
    dataReceived = asymmetricRSADecryption(dataReceived, recipientPrivateKey)
    dataReceived = dataReceived.decode()

    try:
        verifyRSASignature(dataReceived, signatureReceived, senderPublicKey)
        return dataReceived.split("*DEL*")

    except Exception as e:
        print(e)
        return [e]
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
def send_symmetric_key(minstd: int, symmetricKey: object, tag: bytes, nonce: bytes,
                       recipientPublicKey: object, senderPrivateKey: object) -> dict:
    """
    Message format to send symmetric key of the file, asymmetrically encrypted.

    :param minstd: minstd
    :param symmetricKey: symmetric key to send
    :param tag: tag needed for decryption
    :param nonce: nonce needed for decryption
    :param recipientPublicKey: public key of the recipient
    :param senderPrivateKey: private key of the sender
    :return: dictionary which serves as message through the blockchain transaction
    :rtype: dictionary
    """
    data = f'retour*DEL*{symmetricKey}*DEL*{tag}*DEL*{nonce}'
    signature = RSASignature(data, senderPrivateKey)
    message = asymmetricRSAEncryption(data.encode(), recipientPublicKey) + b"*SEP*" + signature

    return {'minstd': minstd, 'data': message}
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
def send_transaction(blockchain: object, sender: str, recipient: str, quantity: float, data: object) -> None:
    """
    Send a transaction through the blockchain with given parameters.

    :param blockchain: blockchain where we have to have the transaction
    :param sender: wallet of the sender
    :param recipient: wallet of the recipient
    :param quantity: quantity to send
    :param data: data to send
    """
    blockchain.new_data(
        sender=sender,
        recipient=recipient,
        quantity=quantity,
        message=data)

    blockchain.block_mining(sender)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
def read_transaction(stored_blockchain: object, new_blockchain:object, address: str) -> (object, list):
    """
    Read new transaction between stored and new blockchains given

    :param stored_blockchain: old blockchain stored locally
    :param new_blockchain: new blockchain, which may have changed if a new transaction was added
    :param address: wallet of the transaction sender
    :return: new blockchain and list of new transactions
    """
    diff = len(new_blockchain.chain) - len(stored_blockchain.chain)
    L = []

    if diff != 0:
        for i in range(1, diff + 1):
            block = new_blockchain.chain[-i]
            for tx in block.data:
                if tx['recipient'] == address:
                    L.append(tx)

    else:  # another transaction in the same block
        block = new_blockchain.chain[-1]
        for tx in block.data:
            if tx['recipient'] == address:
                L.append(tx)

    return (new_blockchain, L)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```python
def last_minstd(blockchain: object) -> int:
    """
    Find last minstd stored in the given blockchain.

    :param blockchain: blockchain where minstd is to find
    :return: last minstd stored in the given blockchain
    """
    for i in range(1, len(blockchain.chain) + 1):
        for j in range(1, len(blockchain.chain[-i].data) + 1):
            mess = blockchain.chain[-i].data[-j]['message']
            if type(mess) == dict:
                return mess['minstd']
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```
if __name__ == '__main__':

    ""Example : a patient visits his doctor and wants analysis results carried out at the laboratory."""

    # Creation of keypairs for doctor & specialist
    doctorPublicKey, doctorPrivateKey = newKeyPair()
    specialistPublicKey, specialistPrivateKey = newKeyPair()

    # Simulation of 2 differents blockchains on the 2 laptops
    blockchain_doctor = python_blockchain.Blockchain()
    blockchain_specialist = python_blockchain.Blockchain()

    rpps = minstd(-1)
    ipp = minstd(rpps)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```
# Creation of 2 wallets
doctor_wallet = "doctor_wallet"
specialist_wallet = "specialist_wallet"

# Inventory
print('ETAT INITIAL \n')
print('Blockchain du médecin:', blockchain_doctor.chain, '\n')
print('Blockchain du spécialiste:', blockchain_specialist.chain, '\n\n')


# A) Request of  the analysis by the authorized doctor to the specialist
analysis_demand = request_info(ipp, 'Demande analyse sanguine de M.Dupont', ipp, rpps, '127.0.0.1',
                    specialistPublicKey, doctorPrivateKey)
send_transaction(blockchain_doctor, doctor_wallet, specialist_wallet, 1, analysis_demand)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```
# B) Message received by the specialist
blockchain_specialist, L = read_transaction(blockchain_specialist, blockchain_doctor, specialist_wallet)

print('UNE TRANSACTION PLUS TARD \n')
print('Blockchain du médecin:', blockchain_doctor.chain)
print('Blockchain du spécialiste:', blockchain_specialist.chain, '\n\n')

print('MESSAGE DECRYPTE PAR LE SPECIALISTE:',
    decrypt_info(L[-1]['message']['data'], doctorPublicKey, specialistPrivateKey), '\n\n')

baseKey = b"abcdefghijklmnop"
encrypted, key, tag, nonce = symmetricAESEncryption("test", baseKey)

# C) The specialist sends the symmetric key to the doctor
sym_key = send_symmetric_key(last_minstd(blockchain_specialist), encrypted,
                tag, nonce, doctorPublicKey, specialistPrivateKey)
send_transaction(blockchain_specialist, specialist_wallet, doctor_wallet, 1, sym_key)
```

VI) Annexes – blockchain en Python

- **Blockchain/handle_transaction.py**

```
print('CLÉ SYMÉTRIQUE ENVOYÉE \n')
print('Blockchain du médecin:', blockchain_doctor.chain, '\n')
print('Blockchain du spécialiste:', blockchain_specialist.chain, '\n\n')


# D) The doctor receives the symmetrical key from the specialist and decrypts it
blockchain_doctor, M = read_transaction(blockchain_doctor, blockchain_specialist, doctor_wallet)
print('MESSAGE DECRYPTE PAR LE MEDECIN:',
    decrypt_info(M[-1]['message']['data'], specialistPublicKey, doctorPrivateKey), '\n\n')

print('ETAT FINAL \n')
print('Blockchain du médecin:', blockchain_doctor.chain)
print('Blockchain du spécialiste:', blockchain_specialist.chain, '\n')


key = b'gX\xad\xc6~\xa5\xc9\xeeZ8RV\x14(\x0fp'

f = loadFile("Analyses M.Dupont.txt", binary=False)
print(f, end="\n----------\n")
cipher, key, tag, nonce = symmetricAESEncryption(f, key)

print(cipher, end="\n----------\n")

decryptedFile = symmetricAESDecryption(cipher, key, tag, nonce)
print(decryptedFile)
```