Simulation

December 9, 2019

1 Simulation of the Healthcare BlockChain

Let's move to the `src` folder where the code is stored, and let's import the libraries that we created and that we are going to use.

[1]:
```
cd ../src
```

```
/home/matteo/PycharmProjects/HealthcareBlockchain/src
```

[2]:
```
from patient import Patient
from event import Event
from doctor import Doctor
```

We have created some fake patients, doctors, diseases and prescriptions to run this simulation.

[3]:
```
# Import patients
from simulations.simulation import Ann, Bob, Charlie, Daniel, Eveline, Frank,␣
 ↪George, Hilary, Ingrid, Juliet

# Import doctors
from simulations.simulation import DrHouse, DrZhivago, DrJD, DrTurk, DrKelso,␣
 ↪DrCox, DrEspinosa, DrAdams, \
DrFrankenstein, DrLecter

# Import diseases
from simulations.simulation import arthritis, bulimia, celiac, diabetes, ebola,␣
 ↪flatulence, gastroenteritis, \
hemorrhoids, insomnia, labyrinthitis

# Import prescriptions
from simulations.simulation import prescription_1, prescription_2,␣
 ↪prescription_3, prescription_4, prescription_5, \
prescription_6, prescription_7, prescription_8, prescription_9, prescription_10
```

Last thing: let's import all the functions that patients and doctors will have at their disposal.

[4]:
```
from utils import get_block, get_chain, add_event, load_private_key, verify,␣
 ↪get_chain_size
```

1.1 A new patient is born

Let's suppose that a new patient, Silvio, is born. A new blockchain related to him is created.

[5]:
```
Silvio = Patient("W852XY")
```

The patient is identified by a unique ID which makes him uniquely identifiable, still preserving his privacy.

Silvio has now his personal blockchain. When a Patient blockchain is created, private and public keys related to that patient are also generated. The patient is provided with a pair of permanent private and public keys and a pair of temporary private and public keys. These keys do not really encrypt anything (there are no messages to be encrypted in this world), but they are needed to give to the doctors permissions to deal with the patient's blockchain.

The permanent private key should always be kept secret by the patient. He or she will always be able to access his or her data by using that key. The temporary key, on the other hand, can be given to the doctor when the physician has to have a look at the patient blockchain or when he or she has to add a new block to the chain. In fact, although the patient is the owner of his or her blockchain, the miner is the doctor. The doctor is the only one who can mine blocks in the chain of the patient. When the physician does so, the temporary keys that were used to give the permissions to the doctor are destroyed and replaced by new temporary keys that only the patient can see. The next time the patient meets the doctor, he or she will have to provide the doctor with the new permanent key, in order to let him or her access the patient's blockchain.

1.2 A new drug is discovered

Let's suppose that medical research works well and a new medicine is discovered. In our simulation, we already have 10 drugs, this is our eleventh. In order to avoid misinformation, we will refer to this drug as drug_11, without using a real name. Physicians can now prescribe drug_11 to their patients, therefore a new blockchain related to this prescription is generated.

[6]:
```
prescription_11 = Event(event="prescription", name="drug_11",␣
↪incompatibilities=["flatulence", "ebola"], loc=locals())
```

Medical research also knows that, if a patient is flatulent or has ebola, this drug will hurt him or her. As we can see from the code, these two diseases are in the list of incompatibilities of the medicine. If we print the incompatibilities of the diseases flatulence and ebola, we will see that their incompatibilities have been contextually updated.

[7]:
```
print(f"Flatulence incompatibilities: {flatulence.incompatibilities}")
print(f"Ebola incompatibilities: {ebola.incompatibilities})")
```

```
Flatulence incompatibilities: ['drug_11']
Ebola incompatibilities: ['drug_4', 'drug_5', 'drug_11'])
```

1.3 How this technology works

Let's consider patient Ann. She has never had any medical issue in her life, so her blockchain only has the genesis block. One day, Ann meets doctor Adams, and the doctor certifies that Ann has contracted celiac disease.

Celiac disease has to be added to Ann's blockchain. Ann gives her temporary key to doctor Adams, who will use it to do it.

[8]:
```
key = load_private_key('../private_keys/R945MU_temporary.pem')
```

[9]:
```
add_event(doctor=DrAdams, event=celiac, patient=Ann, key=key)
```

```
celiac disease added successfully to patient R945MU
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

As we can see, the temporary key has been refreshed. If the doctor tries to use the same key to check Ann's blockchain, he will get an error.

[10]:
```
get_chain(Ann, key=key)
```

```
␣
↪---------------------------------------------------------------------------

PermissionError                           Traceback (most recent call␣
↪last)

<ipython-input-10-71bd96168845> in <module>
----> 1 get_chain(Ann, key=key)


~/PycharmProjects/HealthcareBlockchain/src/utils.py in get_chain(chain,␣
↪key)
     44     """
     45     if isinstance(chain, Patient):
---> 46         check_keys(key, chain)
     47         refresh_keys(chain)
     48     return chain.get_chain()


~/PycharmProjects/HealthcareBlockchain/src/utils.py in check_keys(key,␣
↪patient)
    165                                                 format=serialization.
↪PublicFormat.SubjectPublicKeyInfo)
    166     if key != public_key and key != permanent_key:
--> 167         raise PermissionError("You do not have access to the␣
↪BlockChain of this patient.")
    168
    169
```

```
        PermissionError: You do not have access to the BlockChain of this␣
↪patient.
```

If Ann gives the new temporary key to the doctor, then he will be able to check her blockchain.

[11]:
```
key = load_private_key('../private_keys/R945MU_temporary.pem')
get_chain(Ann, key=key)
```

```
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

[11]:

Index	Name	Doctor	Timestamp	Event	Kind	Previous hash	Hash
0	R945MU	None	2019-12-09 16:26:04.112335	None	None	0000000000000000000000000000000000000000000000000…	694950a87a5999e44d10ccd9412e8e654f2562b607709b…
1	R945MU	D-Y553CX	2019-12-09 16:26:06.444703	disease	celiac	694950a87a5999e44d10ccd9412e8e654f2562b607709b…	f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…

Again, the keys have been refreshed and the old ones cannot be used. What the function `get_chain` returned is Ann's blockchain. We can see that the event "celiac disease" has now been added.

We can also see how the blockchain of celiac disease and the blockchain of doctor Adams have been updated.

[12]:
```
get_chain(celiac)
```

[12]:

Index	Name	Patient	Doctor	Timestamp	Previous hash	Hash
0	celiac	None	None	2019-12-09 16:26:05.802650	0000000000000000000000000000000000000000000000000…	890b0416bd1e1d81feabb73d161ed999541e6a489c6f98…
1	celiac	R945MU	D-Y553CX	2019-12-09 16:26:06.444762	890b0416bd1e1d81feabb73d161ed999541e6a489c6f98…	4ad2129dbd4af332ac225367cae90017809aad912d1cc2…

[13]:
```
get_chain(DrAdams)
```

[13]:
```
            Name Patient                   Timestamp    Event    Kind  \
Index
0       D-Y553CX    None 2019-12-09 16:26:05.803028     None    None
1       D-Y553CX  R945MU 2019-12-09 16:26:06.444789  disease  celiac

                                              Previous hash  \
Index
0       0000000000000000000000000000000000000000000000000…
1       f5f7fabd1150d2e020a56d74582454ca82a91f1f07561f…

                                                      Hash
Index
0       f5f7fabd1150d2e020a56d74582454ca82a91f1f07561f…
1       6d17089c6f9e39a069baac0ab69914d1b0070b12b5ec6b…
```

[14]:
```
celiac.incompatibilities
```

[14]:
```
['drug_9', 'drug_10']
```

One day, Ann catches a fever. She meets another doctor, doctor Cox, who does not know (or does not remember) that Ann has celiac disease (or does not remember that a certain drug is not compatible with that disease).

Doctor Cox tries to make a prescription for drug_9 to Ann (prescription_9), but drug_9 is not compatible with her celiac disease. The blockchain knows that and will throw an error, preventing the mistake.

[15]:
```
key = load_private_key('../private_keys/R945MU_temporary.pem')
add_event(doctor=DrCox, event=prescription_9, patient=Ann, key=key)
```

```
      ␣
↪---------------------------------------------------------------------------

       IncompatibilityError                      Traceback (most recent call␣
↪last)

       <ipython-input-15-d37796f89d9f> in <module>
         1 key = load_private_key('../private_keys/R945MU_temporary.pem')
   ----> 2 add_event(doctor=DrCox, event=prescription_9, patient=Ann, key=key)


       ~/PycharmProjects/HealthcareBlockchain/src/utils.py in add_event(doctor,␣
↪event, patient, key)
        67         raise PermissionError("Only doctors can add events.")
        68     check_keys(key, patient)
```

```
---> 69     doctor.add_event(event, patient)
     70     print(f"{event.name} {event.event} added successfully to patient␣
↪{patient.name}")
     71     refresh_keys(patient)


~/PycharmProjects/HealthcareBlockchain/src/doctor.py in add_event(self,␣
↪event, patient)
     33             The BlockChain relative to the patient that faces the␣
↪event.
     34         """
---> 35         patient.add_block(event, doctor=self)
     36         event.add_block(patient, doctor=self)
     37         self.add_block(event, patient)


~/PycharmProjects/HealthcareBlockchain/src/patient.py in add_block(self,␣
↪event, doctor)
     42             for ev in self.get_chain().Kind:
     43                 if ev in event.incompatibilities:
---> 44                     raise IncompatibilityError(f"Past event {ev} is␣
↪not compatible with {event.name} {event.event}")
     45
     46         self.blocks.append(Block(len(self.blocks), player=self.
↪player, name=self.name, doctor=doctor.name,


IncompatibilityError: Past event celiac is not compatible with drug_9␣
↪prescription
```

Doctor Cox will then prescribe a different medicine, for example drug_8, which is compatible with celiac disease.

[16]:
```
key = load_private_key('../private_keys/R945MU_temporary.pem')
add_event(doctor=DrCox, event=prescription_8, patient=Ann, key=key)
```

```
drug_8 prescription added successfully to patient R945MU
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

Let's have a look at Ann's blockchain now.

[17]:
```
key = load_private_key('../private_keys/R945MU_temporary.pem')
get_chain(Ann, key=key)
```

```
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

[17]:
```
          Name    Doctor                   Timestamp         Event    Kind  \
Index
0       R945MU      None 2019-12-09 16:26:04.112335          None    None
1       R945MU  D-Y553CX 2019-12-09 16:26:06.444703       disease  celiac
2       R945MU  D-Y553BX 2019-12-09 16:26:15.748361  prescription  drug_8

                                              Previous hash  \
Index
0       0000000000000000000000000000000000000000000000000…
1       694950a87a5999e44d10ccd9412e8e654f2562b607709b…
2       f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…

                                                      Hash
Index
0       694950a87a5999e44d10ccd9412e8e654f2562b607709b…
1       f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…
2       812c0a9d8a33caa9d819c6a1c0d3dffb039265af508856…
```

Everything has been stored correctly. We can verify that the chain is not corrupted by running the appropriate function.

[18]:
```
verify(Ann)
```

[18]:
```
(True, 'The BlockChain is not corrupted.')
```

One last thing we can do is, for example check how many people in the world have contracted celiac disease.

[19]:
```
get_chain_size(celiac)
```

[19]:
```
1
```

Only one person (Ann) has celiac disease. Let's suppose that another person, say Bob, contracts celiac disease (this event will be certified by doctor JD). We will see that the `get_chain_size` will print 2.

[20]:
```
key = load_private_key('../private_keys/C901UL_temporary.pem')
add_event(doctor=DrJD, event=celiac, patient=Bob, key=key)
```

```
celiac disease added successfully to patient C901UL
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

[21]:
```
get_chain_size(celiac)
```

[21]:
```
2
```

It is worth noting that, if a patient tries to mine blocks of his or her own blockchain, he or she will get an error. This time, patient Charlie will try to add diabetes to his chain, using his permanent

key.

[22]:
```
permanent_key = load_private_key('../private_keys/R396UZ_permanent.pem')
```

[23]:
```
add_event(doctor=Charlie, event=diabetes, patient=Charlie, key=permanent_key)
```

```
␣
↪---------------------------------------------------------------------------

PermissionError                                Traceback (most recent call␣
↪last)

<ipython-input-23-9a89e2bdc5d9> in <module>
----> 1 add_event(doctor=Charlie, event=diabetes, patient=Charlie,␣
↪key=permanent_key)

~/PycharmProjects/HealthcareBlockchain/src/utils.py in add_event(doctor,␣
↪event, patient, key)
     65     """
     66     if doctor.player != "doctor":
---> 67         raise PermissionError("Only doctors can add events.")
     68     check_keys(key, patient)
     69     doctor.add_event(event, patient)

PermissionError: Only doctors can add events.
```

However, as we mentioned above, the permanent key is never destroyed, and any patient can use it to check his health blockchain any time he or she wants. Now Ann uses her permanent key to check her blockchain.

[24]:
```
permanent_key = load_private_key('../private_keys/R945MU_permanent.pem')
```

[25]:
```
get_chain(Ann, key=permanent_key)
```

```
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

[25]:

Index	Name	Doctor	Timestamp	Event	Kind	\
0	R945MU	None	2019-12-09 16:26:04.112335	None	None	
1	R945MU	D-Y553CX	2019-12-09 16:26:06.444703	disease	celiac	
2	R945MU	D-Y553BX	2019-12-09 16:26:15.748361	prescription	drug_8	

Previous hash	\

```
Index
0      0000000000000000000000000000000000000000000000000…
1      694950a87a5999e44d10ccd9412e8e654f2562b607709b…
2      f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…

                                                      Hash
Index
0      694950a87a5999e44d10ccd9412e8e654f2562b607709b…
1      f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…
2      812c0a9d8a33caa9d819c6a1c0d3dffb039265af508856…
```

[26]:
```
get_chain(Ann, key=permanent_key)
```

```
Temporary keys have been refreshed. Old keys have been destroyed and will not
work.
```

[26]:
```
         Name    Doctor                  Timestamp         Event    Kind  \
Index
0      R945MU      None 2019-12-09 16:26:04.112335          None    None
1      R945MU  D-Y553CX 2019-12-09 16:26:06.444703       disease  celiac
2      R945MU  D-Y553BX 2019-12-09 16:26:15.748361  prescription  drug_8

                                          Previous hash  \
Index
0      0000000000000000000000000000000000000000000000000…
1      694950a87a5999e44d10ccd9412e8e654f2562b607709b…
2      f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…

                                                      Hash
Index
0      694950a87a5999e44d10ccd9412e8e654f2562b607709b…
1      f93caef868c70739091b70d0e8879a0c7dc609145c7d1d…
2      812c0a9d8a33caa9d819c6a1c0d3dffb039265af508856…
```