Achtatou Sean
Sergei Tichomirov

University of Luxembourg

Bachelor in Computer Science

The Block Chain

Semester 2 - 2017-2018

1.Introduction

This research document represents the second Bachelor Semester Project made by the student Sean Achtatou; and his tutor Sergei Tichomirov, this last helping his student in his tasks and work. Both working in the University of Luxembourg offices, this second Bachelor Semester Project, is related to the educational needing for the Bachelor in Computer Science degree directed by the Doctor Nicolas Guelfi, and is focused on one of the many side of the computer science security comprehension that is hard to understand, and why it is important for the future of humanity to get to known with what they will deal with every day.

For this project, we worked on the Block Chain, an algorithm beginning to get known, that could provide a better security for all of us in a near future, mainly all aspects which are related to the digital money. Today, this algorithm is being used in several organization and companies for prevention to any security breach or attacks. We worked along based with the operation of a digital money that is well known today: The Bitcoin.

From the several researches on this algorithm by other researchers, the Block chain has been considered as un-computationally breakable, since it is using one really well implemented and conceived. This project focus on the explanation of how works the Block Chain algorithm, providing an explanation of its high security rate. For this, we worked in ascent, and divided the components of the body of the Block Chain algorithm in several parts to retrieve at the end the final algorithm.

We began with several functions such as the Hash Function, that is one important aspect without the one the Block Chain wouldn't have any sense, that is used to create the different hashes of the data. Then carry on with the Digital signature, that is as important as the Hash Function is, and is used for creating and checking the data. After this, we worked on the Merkle Tree algorithm, a nice way to see if an information belongs or not in a certain block. At the end, those parts put together would lead us to the possibility to create a local Block Chain, and observe how it would work in a real situation.

2.Project Description

2.1. Domain

The domain of this project concerns one of the many interesting world of the computer science: the cyber-security for computer scientist. The cyber-security was a long time ago, when the first computers appeared, not even regarded as the main issue for the computer science. Ever since, with the evolution of technology and their difficulties, today, the cyber-security is now considered as well important for the start of any system or program to be secured, according to its level of importance, or if the user wish to secure it better than it already is.

Due to the rising of hackers and bad programs, the cyber-security is everyday getting stronger to counter these attackers, and grant the possibilities to create new security programs that could potentially counter all types of attacks. Everything suggests that the cyber-security is the future of the humanity for the computer science, by reason of the spread of the technology around the people in the world wide. Without a proper security on these technologies, we could potentially have a lot of issues with many systems put in place, putting people's life in danger.

For this project, the cyber-security is more globalized around the Block Chain. This last is used as an effective way to establish secured transaction between several users using peer to peer connections. The Block Chain could be considered as one of the best security system nowadays.

2.2. Objectives

The main objective of this project is to provide to the people unaware, a new way of thinking about the security of the computer science. For this project, we worked on the security of the Block Chain system, that is well used for many programs since several years, such as the Bitcoin does (this is a money transfer worldwide based on the Block Chain algorithm working by peer to peer connections).

The Block Chain is, as said below in the domain, could be considered as one of the best security system nowadays. Indeed, for some computer scientists having provided research on it, they concluded that it was computationally impossible to break the Block Chain's security. By computationally, the computer scientists attest, even by using a computer, it would take a very long time to be able to break the Block Chain algorithm.

Accordingly, the objective of this project is to show, with proofs, this information they provided, is surely correct, and how we could concretely be so sure about its integrity. We would have the possibility to work along with the Bitcoin to provide an example for our work, if possible.

As conclusion for this project, we should all be convinced, thanks to several implemented programs simulating the Block Chain operation, that it is undoubtedly one of the best security system nowadays, and we can trust that system for now to secure the transactions.

2.3. Constraints

As for any project, some constraints have to be considered. Nonetheless, having some basics in computer science and the love of mathematics, those constraints should not be a problem, otherwise it could be more difficult for the others.

The first of the constraints that has to be considered is the understanding of the average mathematics. We are not in the basics of the security, but a level higher, consequently several notation has to be learnt to be able to follow the formula used for the algorithm, and for at least understand what we are attempting to do in this project.

The second constraint to be aware of, is the capability to implement some basics algorithm for a given language. Indeed, for this project all the programs are being implemented on Python 3. It is a simple language that is well used today by programmer of different level, but if you have already programmed a little, you should be able to follow this project and get pass easily this constraint.

The last constraint but not the least, is that the project is about the security domain of the computer science. This domain is one of the most difficult to master since it uses a lot of mathematics as much practical as theoretical. Therefore, if you have an adoration for security and mathematics, it should not be that problematic.

3.Background

3.1. Scientific

3.1.1. Security in Computer Science

The security in Computer has become the number one important notion in the conception of any applications or network. Today most of the systems are evaluated based on their security strength, but why has the security become so important? The security is helping the user to be prepared for an eventual attempt of security breach, like a program or someone trying to get or modify some data that the one is not allowed to modify or/and even see. Most of the system, to provide efficient protection, supply one to multiple anti-virus, or, in extreme cases, the intervention of a human or an Artificial Intelligence live surveillance. But, still nowadays, none system has the capability to be considered as hundred percent's secured. Would it be able to change in a near future?

3.1.2. Secured Algorithms

From days to days, the technology is evolving in all the domains, and for the computers, the different types of attacks are evolving as well. To provide better security, the computer scientist in security, create nonstop different types of counter-attack to protect against those malicious software or adversaries and lower the possibility of a security-breach. In those protection, there exists:

- Anti-virus
- Surveillance
- Network shield

Still, there exist others ways to protect our data such as using a Secured Hash Algorithm(SHA). The SHA, is a hashing function concealed by the National Security Agency(NSA) that contains the SHA-256(32 bits words) providing a hash of length 256 bits and SHA-512(64 bits words) providing a hash of length 512 bits. This function is mainly used to secure by encrypting and transforming the input data in a fixed bit length according to the SHA used.

3.1.4. Notion of Cryptography

The Cryptography, meaning hidden(crypt) writing(graphy) in Greek, is a technique used to secure the communication between users against any potential adversaries. The main goal is to secure the information by respecting different notion such as:

- Data confidentiality
  (no one can read the data)
- Data integrity
  (the data has not been modified by any adversaries)
- Data authenticity
  (the data has been proven to be the real one)
- And Non-repudiation
  (once the data has been sent, the sender cannot claim he has never send the data)

The cryptography is used in multiples domains:

- Commerce
- Payments cards
- Computer passwords
- Military
- ...

Most of the time, the encryption is done by using two different keys own to the user:

- The Public key
  (used to decrypt by the others the data)
- And the private key
  (used to encrypt the data).

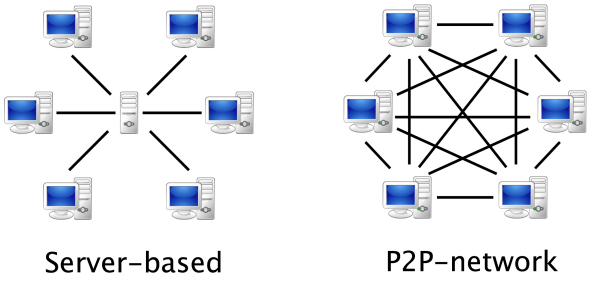
3.1.5. Why get interested in the Block Chain?

Most of the actual networks, such as the well-known Internet, until now are working with a client-server protocol. Indeed, for many years now, those are used to give or receive information from the others users from all around the world. However, the Block Chain system, working almost on the same principles, could be bringing an another advantage to those connections between the users. The Block Chain, contrary to the Internet, lies on a connection from peer-to-peer, meaning there doesn't exist a centralized server that control the network, all the users are interconnected between them and are aware of each other's. Some computer scientist opinion affirm that the Block Chain system could become the future of the money transaction thanks to the reliable security it could provide to the users.

3.2. Technical

►3.2.1. Visualization of the Block Chain

Since the beginning of the interconnected computers, the computer scientist has used different ways to communicate, one of the most known and used is Internet, mainly based on sharing data between client-server, so there exist a central between the exchange. Counter to that, the Block Chain, as its name suggests, a chain of blocks, is a storage and transmitting technology without any control by peer to peer. The users are exchanging transaction between them using blocks containing multiples transactions. We can compare the Block Chain as a big spider web with nodes connected between them and communicating every time. It is nowadays used by multiples companies, agencies and applications worldwide for their security system.



Picture 1. Graph of a Server-based and a Peer to Peer network.

3.2.2. Types of Hash Functions

The hash function can be split in two classes:

- Unkeyed
  (single input parameter and a message – MDC)
  - ► The MDC (Modification Detection Codes) provide a hash of a message with additional mechanisms to facilitate integrity assurances
- Keyed
  (two input, a message and a secret key – MAC).
  - ► The MAC (Message Authentication Codes) provide a hash of a message without any mechanisms to facilitate integrity assurances

There exist two types of cryptography for hash function depending on the usage:

- Cryptographic
  - ► For the cryptographic hash function (Example: SHA), it will provide a way to guarantee multiples securities properties, those being :

    • Deterministic
    (same message result in the same hash)
    • One-way function, called as well preimage -resistant
    (impossible to generate a message from its hash value)
    • Second preimage-resistant
    (for a message we can't find two different hash value)
    • Collision resistant
    (for two messages we can't find the same hash value)
- Non-cryptographic
  - ► For the non-cryptographic hash function (Example: Seahash), it will just try to avoid possible collisions without taking in consideration the security. It could be used to detect data corruption for example.

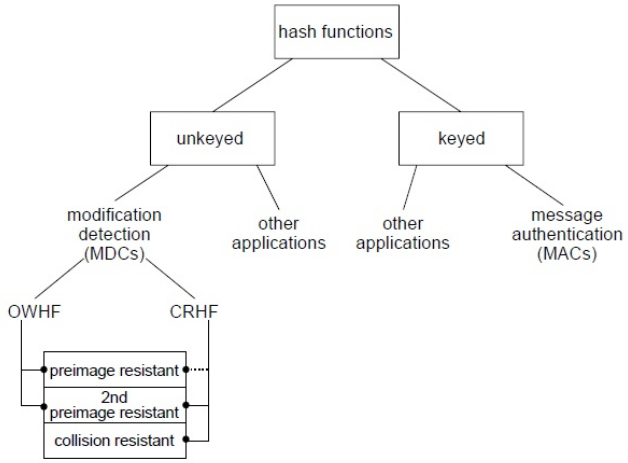


Figure 9.1: *Simplified classification of cryptographic hash functions and applications.*

3.2.3. What is the Digital Signature?

The digital signature is a mathematical scheme to demonstrate the authenticity of a digital message. It is based on three principles:

- Authentication
- Non-repudiation
- Integrity

And it consists of three different algorithms:

- A key generation
  (create a private key and a corresponding public key)
- A signing
  (with a message and the private key, it produces a signature)
- And the signature verification

(given the message, the public key and the signature, it checks the authenticity of the message)

There exist two types of signature schemes:

- Appendix (Example: DSA)
  (require original message as input in the verification)
- Message recovery (Example: RSA)
  (the message is recovered from the signature itself).

There exist variant of those signature. The ECDSA (Elliptic Curves Digital Signature Algorithm), for purpose, is a variant for DSA, the difference come from the manner to create the three algorithms. The DSA and RSA are based on simple logarithms for these tasks, while the ECDSA will use elliptic curves instead of equation to achieve it. The main advantage of ECDSA would be the use of smaller length for the

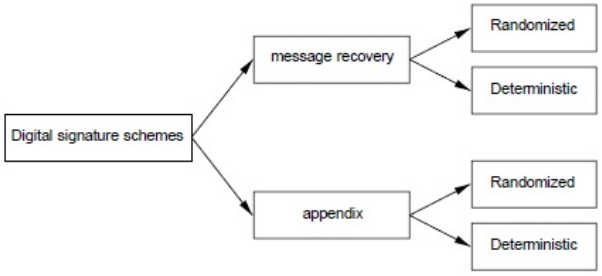


Figure 11.1: *A taxonomy of digital signature schemes.*

3.2.4. What is The Merkle Tree?

The Merkle Tree is a simple algorithm, used in the Block Chain, and applying multiples a hash function, is used to provide a way to verify if a transaction is in a set or not, that is called the membership or not membership (for the Block Chain the set would be a block). Called as well hash tree, the Merkle Tree can be represented as a tree with multiples leaf node labelled to the hash of a data block. It is an efficient and secured verification for the content of large data structure. However, it is not the only existing tree hash, some other are working the same way but with minor modification, such as the Tiger tree hash that is using the Tiger hash.

4.The Block Chain

4.1. Requirements

4.1.1. The Hash Function

For the Hash function, we will talk about how it is working in detailed with an example of algorithm that can be found behind it. The objective of this project is to provide a way to know if the different parts of the Block Chain are secured, so we will need to introduce all the security issues that could happen with this function.

The Hash function, is a complex algorithm that has been designed by expert computer scientist decades ago. As said below, it is used to transform any type of data, in an encrypted fixed data length according to the type of Hash we are using. It is also called a compression function. Here, we will focus on the SHA-256(Recall: 256bit length of encrypted data). To be able to compress any data in a secured manner, the algorithm had to be concealed such a way to protect from adversaries. To achieve this, the algorithm is implemented with multiples functions:

- XOR
  (this a function which can be compared to a sum)
- AND
- OR
- COMPLEMENT
  (for a set A, that is all that is not in the set A)
- MODULO
  (the rest of a division)
- BITS SHIFT LEFT AND RIGHT
  (used to manage calculus with bits).

Fact: Most of the people who has tried to create their own hash function algorithm as performant as SHA-256, has given up at some time.

Adversaries might want attempt to attack this algorithm. There exist different possible attacks, those are first depending on the types of Hash function the adversary is dealing with, MAC or MDC.
For the MDC, the adversary has two possible ways to break the algorithm:

- Attack OWHF [one-way hash function]
(given a hash value Y the adversary would find a pre-image X such that Y = h(X) or for a pair (X,h(X)) would find X' such that h(X') = h(X) . Clearly, with a

hash value, he could find back the decrypted data)

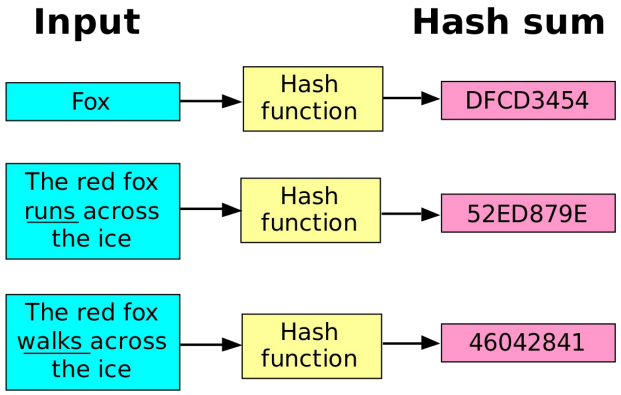
- Attack CRHF [collision-resistant hash function] (given two different input, the adversary would find the same hash value)

For the MAC hash function, the adversary would have others possibilities. Here he would try, without knowing the key K , to compute a new text-MAC pair for some text given one or more pair. For this situation, the adversary could attempt three different attacks:

- Known-text attack
  (the adversary knows one or multiples text-MAC from the victim)
- Chosen-text attack
  (same than Known-text attack, except the adversary can choose the text-MAC)
- Adaptive chosen text attack
  (the adversary use Known or Chosen-text attack but now will attack based on the results he would retrieve from the text-MAC)

If an attack is successful, what would happen? If such occurs, it is called a forgery due to the adversary penetrating the algorithm. There exists different level of consequences depending on the degree of control of the adversary:

- Selective forgery (most of the damage)
  (the attacker can produce any text-MAC pair for a text of his choice)
- Existential forgery
  (same than for selective forgery, but the adversary cannot control the value of the text-MAC)



Picture 2. Examples of some input in a Hash Function giving a Hashed output

4.1.2. The Digital Signature

Here, we will concentrate on the algorithm of the DSA since it is used for any arbitrary length message, while the RSA is used for short messages. Such as the Hash function, the algorithm is quite complex to understand at first side. Let us remember that the digital signature must provide three different algorithms for: keys, signature, verification.

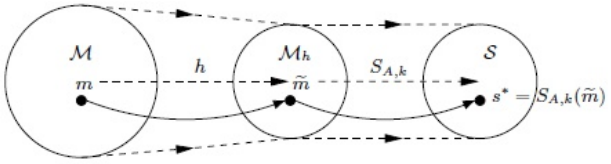
- First the Digital Signature algorithm has to do the key generation by creating a public and private key:
  - ► The private key would be defined as : $\mathcal{S}_A = \{S_{A,k} : k \in \mathcal{R}\}$ of transformations.
  - ► Each $S_{A,k}$ is a one-one mapping from $\mathcal{M}_h$ to S and called a signing transformation.
  - ► Each $\mathcal{S}_A$ defines a mapping V_A from $\mathcal{M}_h \times \mathcal{S}$ such that :

$$V_A(\tilde{m}, s^*) = \begin{cases} \text{true}, & \text{if } S_{A,k}(\tilde{m}) = s^*, \\ \text{false}, & \text{otherwise}, \end{cases}$$

for all $\tilde{m} \in \mathcal{M}_h, s^* \in \mathcal{S}$, $\tilde{m} = h(m)$, $m \in \mathcal{M}$.

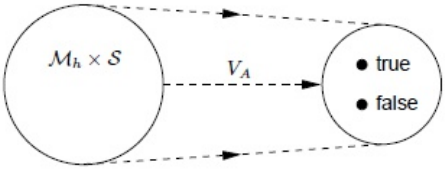
  - ► V_A is called a verification transformation and might be computed without knowing the signer's private key.
  - ► At the end, it will have created a public key and a set of private keys $\mathcal{S}_A$.

- After the creation of the keys, a Digital Signature algorithm has to create the signature for the signer, for this it will:
  - ► Select an element K (select one private key)
  - ► Compute $\tilde{m} = h(m)$ and $s^* = S_{A,k}(\tilde{m})$
  - ► The signature for the signer for m is S* .



(a) The signing process

- The last step is the verification. If any other entities would like to verify the signature, a Digital Signature algorithm have to these steps :
  - ► Obtain the authentic public key V_A .
  - ► Compute $\tilde{m} = h(m)$ and $u = V_A(\tilde{m}, s^*)$.
  - ► Accept the signature if and only if u is "True". Otherwise, the signature is invalid.



(b) The verification process

Such as for the Hash function, multiples attacks on the Digital Signature algorithm must happen and be evoked. Here the goal of the adversary would be to produce signatures being accepted by others entities. Any adversary that would be able to break the algorithm would result in a:

■ Total break: adversary is able to compute the private key information of a signer or an efficient signing algorithm equivalent to the valid one.

■ Selective forgery: adversary is able to create valid signature for a particular message chosen.

■ Existential forgery: adversary is able to forge signature for at least one message, but has little or no control over the message whose signature is obtained.

To be able to break the Digital Signature algorithm, the adversary has multiples attack choice:

■ Key-only attacks: adversary knows the signer's public key.

■ Message attacks: adversary is able to examine signatures corresponding to known or chosen messages. This type of attack is subdivided into three classes:

► Known-Message attack: adversary has signature for a set of message known by him but not chosen.

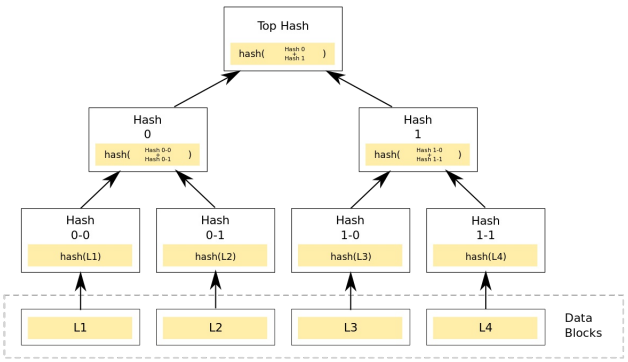
► Chosen-Message attack: adversary obtains valid signature from a chosen list of messages before attempting to break the signature scheme (non-adaptive attack)

► Adaptive chosen-message attack: adversary may request signatures of messages depending on the signer's public key or on previously obtained signatures or messages.

4.1.3. The Merkle Tree

Here is the Merkle Tree, we will talk more in detailed about its operation and why it is used for. On the contrary to the others two function, the Merkle Tree is not really used for security but more for verification.

The Merkle Tree has said far below is used in the Block Chain algorithm. Concretely it is located in each block of the set of Blocks of the Block Chain. The algorithm of the Merkle Tree contrary of the Hash function and the Digital Signature is quite simple. For a set of hashed data (transactions) in a block, the Merkle Tree will sum up in pair each transaction. These last will after be hashed resulting in a new hashed data of the same length. By doing this the numbers of hashed data before has been divided by two. The Merkle Tree will continue to do this for the new set of hashed data until eventually arrived to a single hashed data in the set. This single hashed data would be the Merkle Root of the block. When we would want to see the block identity, the Merkle Root will be put in first followed by the transaction. But what to do if the number of hash data is odd? To solve this problem, just add a new hash data of value zero to the set. So each time you would have an odd number of hashed data you would end up with an even number and carry on the Merkle Tree algorithm. The purpose of the Merkle Tree is just to be able to prove that a certain transaction is a membership of a block or not. What it would do is go all down until reach a transaction that is similar to the one looked for and hash to the top, if the value is equal it is a membership of the block, otherwise it is not.



Picture 3. Graph of the Merkle Tree operation on multiples Data Blocks

4.2. Design

4.2.1. Working with the Hash Function

As we said below in the introduction and our objectives, we have to break the Block Chain in several parts to understand and test each one. Since the Hash function is one of the first algorithm to be used by the Block Chain, we begun to get interested to this one. For this project, we decided to work with the SHA-256 hash function. The reason, is because the SHA-256 algorithm is the most used in the Block Chain nowadays, and on which we can rely according to the researches of PhDs. Before beginning to work

with the SHA-256, to be able to create and test the Hash Function security, we have to implement it on a certain language. For this project, we used Python3, more precisely PyCharm because it allows us to debug our code. We used this language, because it is one of the easiest language to learn if we are novice, and the most understandable by the crowd. Furthermore, it can be downloaded and ran on any operating system Windows or Linux.

What is the point of this? As we said in the security of the Hash Function, one of the property of MDC has to provide is the collision-resistance, meaning that for two different input we can't find the same Hash value. Here, we will try to see with different input if we can find an equality between them at a certain time.

To test it, once we have opened PyCharm, we began by creating a module called "SHA-256". In this module, the first thing we did is to import the library of "hashlib". Indeed, the library of PyCharm does contain the module "hashlib" containing already several different types of Hash Function. Consequently, we do not need to download any external module or implement ourselves the complicated algorithm that the Hash Function is. For example, if we have to use the Hash function SHA-256 on the word "Hello", the code would be: hashlib.sha256(« Hello ».encode()).hexdigest()

What we have to do after is to see for two different inputs, if they provide the same hash value. For this, we decide to do a loop from zero to one billion for which we do the Hash Function of SHA-256 to each one. We begin by looking for an equality between two input if the first index of the hash value is zero for both. Then, we do the same for the index of the hash value being two zeros for both, and again until we try to find an equality between two input if their hash value has ten consecutives zeros.

4.2.2. The use of a Digital Signature

For this part we do not need to implement anything on Python. Indeed, the Digital Signature is an algorithm used in the Block Chains systems which are evolving in an interconnected network, such as the Bit Coin does. It means multiples users on this network are interconnected and are interchanging between them by peer to peer. Since at this point, we are not working on the network of the Block Chain, but more on its security, the implementation of the Digital signature is then optional. On the contrary, we need to see the security application of it, and an explanation in detail of its operation is needed to understand the rough security behind the Digital Signature algorithm.

In the creation process:

- The entire Document of Alice is hashed using a hash function (SHA-256) giving a fingerprint to the document.

- Then the fingerprint is encrypted with the secret key of Alice which gives the Digital signature of the document.

- Since we are dealing with an appendix scheme signature, the document and the Digital Signature are both being send to Bob. The difference with a message recovery, is that in the message recovery it would send a digital signature containing the document inside it which would need to be recovered. And the creation process is done.
On the other side, for Bob to verify the signature. The Digital signature algorithm needs to do the verification of the signature.

Called the Digital Signature verification process:

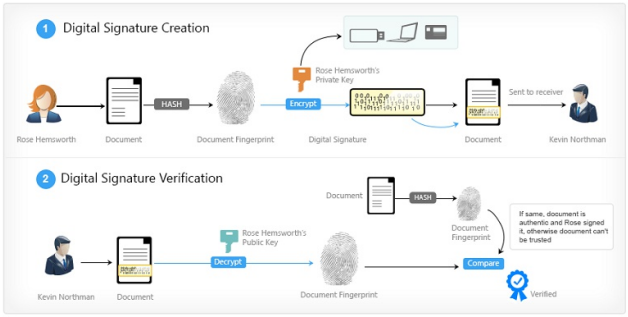
- Bob have to take the Digital signature he received, and applies the Public key of Alice so it decrypts it. It would give Bob a document fingerprint.

- Then he will take the document and apply a hash function (SHA-256) on it, which gives an another document fingerprint.

- Bob will simply have to compare the two fingerprints he got, and see if they are the same. If they are actually the same, the signature is verified and trusted, otherwise it is rejected.

We can see that the three properties of the Digital Signature are respected:

- Authentication
- Non-repudiation
- Integrity



4.2.3. Provide a simple Merkle Tree

For the Merkle Tree algorithm we use as well the language Python3 with PyCharm. As recall, the only purpose of the Merkle Tree is to provide a way to verify if a certain transaction is a membership or not of a block. It is not about security of the Block Chain, but just a verification of the hold of a block.

In the code, since we are using a random and the hash function, we need the importation of the libraries “hashlib” and “random”. To implement the Merkle Tree, we have to create a random number of hashed data which will be used as a base for our Merkle Tree. Here, we can use the hash for the numbers from zero to hundred. What we do after, is implementing the hash of the sum of each pair of the hashed data, if the number of hashed data is odd, we add a hash of the value zero to the numbers of hashed data. We end up with twice less hashed data. And we redo it by implementing a loop, until we end up with one hashed data that will be our Merkle Root of the block. To provide a way to see if a transaction is a membership of our block, we need to implement a search function that will go all down the our Merkle Tree until it reaches the same number or some near it. Since we are not dealing with a Block Chain, we can simply create a list regrouping all our based hashed data, and look through it.

In a real Block Chain, the algorithm would go back at the bottom of the Merkle Tree, taking each time the hash data used to go further down. For example, in a Merkle Tree with eight data, it would need six hash data. For sixteen hash data, eight. The formula is the numbers of data converted in based two, the power is doubled and gives the number of data needed : 2^n→n.2 = number of hash data needed.

4.2.4. Create a local Block Chain

For the last part, since we are not working with the network, we are creating a local Block Chain simulating some transactions. It is implemented on Python and uses JavaScript Object Notation (JSON file) which helps us to structure the data of the Block Chain.

The local Block Chain is combining the Hash Function, the Merkle Tree and the mining of a block. The mining of a block in a network, is normally the job of the miners, which is to calculate complicated algorithm to provide a verification of the transactions done and put them on the chain of the Block.

To create a Block Chain, we need:

■ To create two arrays, unconfirmed which will hold the transactions that are not confirmed and the block which will hold all the blocks in the block chain .

■ To create the JSON file which will update the data of our Block Chain (data.txt)

■ The Block Chain will provide different functions:

► AddToUnconfirmed(): It will append a transaction to the unconfirmed.
► Mine_Block(): It reads the transaction from the unconfirmed. Since a transaction cannot be included in two different block(double-spend), it checks if it doesn’t already exist in a block. Mining is calculating, for each new block put in the block chain, several properties such as :

o Height (Position of the Block among the others)

o Hash (It is a calculation of the hash of the block based on a target. To find it, we need to hash the sum of the prev_hash, the merkleroot and the nonce. While the result is greater than a given target, it will loop by adding each time one to the nonce value)

o Nonce (It is found by calculating the Hash of the block)

o Prev_Hash (Hash of the previous block)

o Merkle Tree (MerkleTree algorithm to the transactions)

o Transactions (All the transactions of the block displayed)

Note: In a real Block Chain, the Mining operation is done by the Miners. The miners are the computers of people chosen to do the mining operation of a block because the power of calculation or algorithm of their computer is satisfying for the Block Chain timeout. It is mainly because the calculation of the Hash and the Nonce are taking a lot of time and without a proper computer, it would take too much time. Of course, those chosen do this are reward each time with digital money.

► Get_block: Return the string representation of the block contents we are looking for, otherwise None.
► Find_tx: Return the hash of the block which includes the transaction we are looking for, otherwise None.

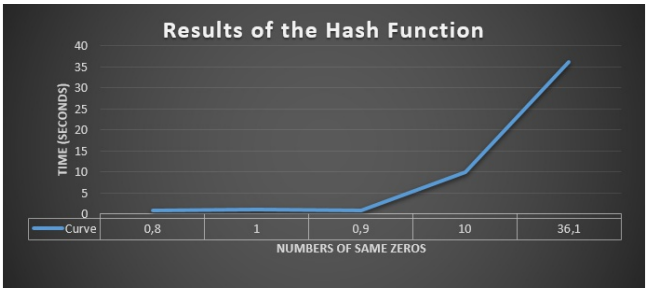
4.3. Production

4.3.1. Hash Function result

For the Hash Function, let us see what result we got from our testing of the function on Python. Recall that we are trying to see if the Hash function provide a collision-resistance, two different input cannot give a same hash value. Here we are trying to find two different inputs having a same number of consecutives zero from zero to ten, for the integers from one to one million. Here the result for each hash of the integers is of two hundred and fifty-six bits since we are using SHA-256. For the hash function to break, all the two hundred and fifty-six bits should be the same for two different input.

Numbers of same zeros	Integer 1	Integer 2	Hash of the integer 1	Hash of the integer 2	Time taken (seconds)	Observation
One	39	49	0b9189...	0e17da...	0.8	The difference between the two integers is of 10.
Two	286	671	00328c...	00bebc...	1	The difference between the two integers is of 485.
Three	886	1039	000f21...	00037f...	0.9	The difference between the two integers is of 153.
Four	88484	172608	0000a4...	0000f7...	10	The difference between the two integers is of 84124.
Five	596138	665782	00000691...	000000039...	36.1	The difference between the two integers is of 69644.
Six	/	/	/	/	Still running after 5 minutes ...	/
Seven	/	/	/	/	Still running...	/
Eight	/	/	/	/	Still running...	/
Nine	/	/	/	/	Still running...	/
Ten	/	/	/	/	Still running...	/

We can see by the observation and the time taken that the results are exponentially rising. Trying to find similitude for only six digits is already taking a lot of time compared to the previous results. And we can eventually presume that the next results, if we get one, will be each time much higher than the previous one. What we can conclude by that is, that the hash function is indeed well implemented against attacks. Indeed, since when we are trying to find only 6 same digits is taking a long time, what would it be for 256 digits? Since it is proportional, it would probably take years, decades, may be centuries to find a same hash value for two different integers. Indeed the possibilities of hash value are about 2^256.



4.3.2. Digital Signature result

For the Digital Signature result, let us explain why the three properties are satisfied first:

1. Authentication
   ► This is respected when Bob is doing the verification of the document, by using the Public Key of Alice, if the two fingerprint are similar, Bob will know that the document is coming from Alice and no one else.
2. Non-repudiation
   ► Since the Block Chain is normally operating on a network, the verification of the document is not only done by one user, it is done by all the network which will check the document signature. Since it is send in all the network, Alice cannot can say she hasn't send anything since everyone will be aware of her transaction.
3. Integrity
   ► This is once more done by Bob, by using a hash function on the document for encrypting and decrypting, the little modifications will be seen as a big difference at the comparison of the two fingerprints.

Let see now why this method is efficient against adversaries by recalling all the possible attacks:

■ Total break: For this the adversary must be able to compute the private key of Alice or the algorithm creating it.

► The private key of a user is most of the time conserved in the computer files or in an external physical card to avoid adversaries for looking in your system files. It is then hard to impossible to get the private key of a user.

► If the adversary would like to be able to create the algorithm for creating the private key, he needs to get the right mapping and transformations as the user does. Moreover, the digital signature creates a set of private keys, in which the user will pick one, so the adversary might need to pick the right one again.

■ Selective/Existential forgery: For this, adversary is able to create valid signature for a particular or random message.

► To create a valid signature, the adversary need to have a private or public key, which with the ones the verification of a document will give the same fingerprint after a hash function. Since we saw that the hash function is computationally taking a long time trying to break it, the adversary might wait as well for a long time.

In conclusion, the digital signature is well protected against several types of attacks, unless we want to be attacked, the algorithm should be efficient against adversaries.

4.3.3. Merkle Tree result

Once again, since the Merkle Tree is not provided for security but for simple verification, its utility is not mandatory in the Block Chain, it is just used to prove the membership of a transaction in a block.
To be sure to follow the result, let's not use a random, but numbers from zero to one thousand twenty-four. Recall that the Merkle Tree is using a hash function, here we use SHA-256.

The results we get are:

Numbers of elements in the Merkle Tree	Number of hashes done	Number of hash needed for membership	Results
1	0	1	5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
2	1	2	fa13bb36c022a6943f37c638126a2c88fc8d008eb5a9fe8fcde17026807feae4
4	2	4	862532e6a3c9aafc2016810598ed0cc3025af5640db73224f586b6f1138385f4
5	3	6	2065493a65fc5f33e1612819b95dccb28d6ad5de6cc7ee34f17de4dceae2fa61
10	4	8	c44da581adcd4f45398fd2a4be492814ffa9fced3111d442532c102000aad4c2
16	4	8	a901f842b0016f1e350d20b751851a7179e26dfbb74b213c7a92d37f3c4fbb6c
32	5	10	39e61dd46d3a471055b2502a178098c023ade5f61468eb646b3380f0958eb708
64	6	12	c7fa97592f2ae198f85e930e7cd18decf5f3dd1354b58618c44bf3bfd1e4ca18
128	7	14	7e36da2cbc09bb20ab02cf8ce67936f195e6f0c2352cd5eaab51cd28e98f62cf
256	8	16	ca5034d539d0939709b61df50a67f3a4271601eb46102b902805e918c29fe1da
512	9	18	8c017e717e6e1f175721f25c36488a50fe316afd4f14f8d42ccf6517495a4db5
1024	10	20	ef263f22ed8d4c0847770cac29762a088320fa287993b68d0060f04148cb8e0c

We can observe that according to the numbers of elements in the Merkle tree, the number of hashes needed is rising. Moreover, we can as well see that for the membership, the number of hashes needed are equal two times the number of hashes done. Indeed, for a membership, the Merkle Tree will check if a transaction is in the Block by going to the bottom and looking for it. Once at the bottom, it needs to redo the hashes to check if we end up with the same result, but since to get a hash we the sum of two others hash, each time it goes down it takes two hashes.

4.3.4. Block Chain result

In a real Block Chain, such as the Bit Coin, the Digital Signature add a high security between the users, and is then needed.

In our Block Chain, since we are working offline, the Digital Signature is not done because no verification between users are done, this is in local.

So, for this local Block Chain, we look at the result we are supposed to get for each function enumerated in the design. For more understanding, let's say we already created five blocks inside the Block Chain with transactions inside each one(App3).

How each of the functions of the local Block Chain should behave:

■ AddToUnconfirmed()(Picture1-4)

► For now, the data of unconfirmed must be empty. Now, we add a new unconfirmed transaction such as "Hello World".

► The JSON file should now be updated so we can see the in the unconfirmed the hash of the transaction:
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"

► We redo the operation by adding "I love cheese" giving:
9ba411b1858c20e388c8a167ed0cd8d31b07491952013ccbe4c2b0b523336ca5

► The unconfirmed data of the JSON file should be like this at the end:
"Unconfirmed": [
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824",
"9ba411b1858c20e388c8a167ed0cd8d31b07491952013ccbe4c2b0b523336ca5"]

■ Mine_Block()(Picture 8)

► Here, the function firstly has to check if a transaction inside the unconfirmed is not already in the transaction of the Blocks composing the Block Chain. If it is, it will be erased, otherwise, the transactions left will be put in a new block added to the chain of blocks.

► Then, for this new block added, since the data of the chain has been changed, it will need to do all the functions enumerated earlier:

- o Hash
- o Nounce
- o Prev_Hash
- o MerkleTree

■ Get_block()(Picture 6)

► Here, we try to get the data of a certain block. We want the data of the Block 2. So, it must display all the properties that the Block 2 has:

- o Height (must be of 2)
- o Hash
- o Nounce
- o Prev_Hash (must be of 1)
- o Merkle Tree (depend on the transaction inside the block)
- o Transactions

■ Find_tx()(Picture 7)

► Here, we try to look for the transaction “Bachelor in Computer Science”. Since it doesn’t exist in any Block, it returns us nothing.

► Now, we try with “I love cheese”. If this transaction has been mined from the unconfirmed and put in a Block, it will return the Hash of the Block.

4.4. Assessment

4.4.1. Hash function

For the Hash function, the results for the collision-resistance testing are satisfying and convincing. Indeed, from those results, we can clearly observe that this function has a powerful implemented algorithm to provide an efficient protection against the eventual adversaries trying to break it. Here, the main problem for the adversaries would be the time. Indeed, has shown in the results, it would take a really long time before being able to find a same Hash value. We see in the graph that the time taken would grow exponentially to the number of correspondence.

This is already enough to say this function is well secured to be used in the Block Chain. Still it would have been great to work on the others aspects of the Hash function, such has the MAC to provide different point of view compared to the MDC. But here, we were focusing on the Bit Coin operation using exactly MDC hash function system.

4.4.2. Digital Signature

For the Digital Signature observation, those are as well satisfying and understandable. Indeed, it has been done, by showing first that the properties of the Digital Signature were valid with the examples of Alice and Bob evolving in the Digital Signature algorithm. Moreover, we provide an explanation on each possible attacks on the Digital Signature, and how it would be possible to do it. We observed afterwards that for each attacks enumerated, it is really hard nay impossible to break the algorithm since it is likewise based on the using of the Hash Function algorithm already shown has secured. The Digital Signature is consequently a secured way to create signature in the Block Chain and protect from adversaries’ attacks.

4.4.3. Merkle Tree

For the Merkle Tree result, we didn’t really show any security aspect of it for the Block Chain but more its utility. Recall, the only purpose of the Merkle Tree is to provide an way to verify if a transaction is a membership of a certain block. Nevertheless, the project to represent how works the algorithm is satisfying. We can observe that, more we have transactions, more it needs hash value back to verify the membership of a transaction. Still, the number of hash value used for membership are lesser then the total of hash value of the Merkle Tree. We can consider the Merkle Tree has a proof of a transaction in a block and can be not used if wanted.

4.4.4. The local Block Chain

The main project, the local Block Chain. For the result of the implementation of the local Block Chain using all the functions enumerated below, except the Digital Signature; it is really satisfying. Indeed, the local Block Chain is providing all the results we were expecting for in the production. Although, the creation of the Block Chain at the beginning is taking a certain time since we need to do the Mining for each block. Afterwards, the functions are operating properly.

Nevertheless, we were not working with the Digital Signature algorithm since we weren't focusing on the network side, it would have been great yet to implement it since it raises a lot of the security in the Block Chain and at least see when it happens during the implementation. Of course, we know when it would happen in the code since we know how works the Digital Signature algorithm. Each time a new transaction would be created by the user, it would create a private and public key for the user and used on the transaction. Afterwards, in the Mining process, it would verify the transaction and put it in block if it is considered as correct, otherwise it would have been rejected. Here we encounter this by only verifying if the transaction wasn't already in a block, we didn't verify the correctness of it.

5.Conclusion

We can clearly see that for each function, which are used in the Block Chain, those are well protected and provide together an efficient secured implemented algorithm against adversaries for the Block Chain. Although, there exists multiples attacks against it, most of theses are way to hard for the adversaries to deal with because of the time it would take most of the time. Since, it is getting trusted, the Block Chain could eventually then be used in a near future everywhere around the world in all the industries and companies to protect their transactions of money or information between users without the fear of being attacked. Today, some banks and industries are implementing their block chain and using it everyday. None efficient security fault has been yet found nowadays which would break the Block Chain algorithm, but will it happen one day?

6.Appendix

```
unconfirmed = {}
unconfirmed["Unconfirmed"] = []

blocks = {}
blocks["Blocks"] = []
```

Picture1. Creation of the Unconfirmed and the blocks lists.

```
for i in range(nbB): # Create the blocks#
    block = {"Block" + " " + str(i): {"Height": i, "Hash": b, "Prev_Has":
c, "Nonce": d, "MerkleRoot": e,"Txs":[]}}
    blocks["Blocks"].append(block)
```

Picture2. Creation of five block in the blocks list.

```
blocks["Blocks"][0]["Block 0"]["Txs"].append(hashlib.sha256("Test".encode()).hexdigest())
blocks["Blocks"][0]["Block 0"]["Txs"].append(hashlib.sha256("Hello".encode()).hexdigest())
blocks["Blocks"][0]["Block 0"]["Txs"].append(hashlib.sha256("World".encode()).hexdigest())

blocks["Blocks"][1]["Block 1"]["Txs"].append(hashlib.sha256("Bye".encode()).hexdigest())
blocks["Blocks"][1]["Block 1"]["Txs"].append(hashlib.sha256("Have fun".encode()).hexdigest())
blocks["Blocks"][1]["Block 1"]["Txs"].append(hashlib.sha256("See you".encode()).hexdigest())

blocks["Blocks"][2]["Block 2"]["Txs"].append(hashlib.sha256("This".encode()).hexdigest())
blocks["Blocks"][2]["Block 2"]["Txs"].append(hashlib.sha256("is".encode()).hexdigest())
blocks["Blocks"][2]["Block 2"]["Txs"].append(hashlib.sha256("sparta".encode()).hexdigest())

blocks["Blocks"][3]["Block 3"]["Txs"].append(hashlib.sha256("I".encode()).hexdigest())
blocks["Blocks"][3]["Block 3"]["Txs"].append(hashlib.sha256("am".encode()).hexdigest())
blocks["Blocks"][3]["Block 3"]["Txs"].append(hashlib.sha256("groot".encode()).hexdigest())

blocks["Blocks"][4]["Block 4"]["Txs"].append(hashlib.sha256("Hello".encode()).hexdigest())
blocks["Blocks"][4]["Block 4"]["Txs"].append(hashlib.sha256("is it me".encode()).hexdigest())
blocks["Blocks"][4]["Block 4"]["Txs"].append(hashlib.sha256("you looking
for".encode()).hexdigest())

blocks["Blocks"][5]["Block 5"]["Txs"].append(hashlib.sha256("It".encode()).hexdigest())
blocks["Blocks"][5]["Block 5"]["Txs"].append(hashlib.sha256("doesn't
work".encode()).hexdigest())
blocks["Blocks"][5]["Block 5"]["Txs"].append(hashlib.sha256("that way".encode()).hexdigest())
```

Picture3. Creation of the five first blocks in the block chain

```
with open("data.txt", "w") as i:  # Put all the data in a data.txt file#
    json.dump(BlockChain.unconfirmed, i, indent=4)
    json.dump(BlockChain.blocks, i, indent=4)
    i.close()
```

Picture4. Function used to create the data file JSON and put the list of Unconfirmed and blocks.

```
def New_Transaction_Unconfirmed(a):
   #Append a new transaction unconfirmed to the Unconfirmed#
   BlockChain.unconfirmed["Unconfirmed"].append(a)
   print("[Success] Your transaction has been add to the unconfirmed
transactions.")
```

Picture5. Function used to add a transaction to the Unconfirmed list.

```
def Search_Block_Contents(a):
    print("Informations for the Block",a,":")
    try:
        print(BlockChain.blocks["Blocks"][int(a)]["Block"+" "+a])
        print("[Success] The contents for the Block", an, "has been
displayed.")
    except:
        print("[Error] The Block Height you entered doesn't exist.")
    #Read for each block the height, once found, print the block contents#
    return
```

Picture6. Function used to display the contents of a block in the block chain.

```
def Search_Transaction_Hash_Block(a):
    for i in range(nbB):
        for k in BlockChain.blocks["Blocks"][i]["Block" + " " +
str(i)]["Txs"]:
            if k == hashlib.sha256(a.encode()).hexdigest():
                print("[Success] Transaction found in the Block",i)
                print("Hash of the
Block",i,":",BlockChain.blocks["Blocks"][i]["Block"+" "+str(i)]["Hash"])
                return
            else:
                continue
    print("[Error] The transaction hasn't been found in any Blocks.")
    return None
```

Picture7. Function used to display the hash of a block containing a certain transaction.

```
def Mine_Block():
    global nbB,Target

    for i in range(nbB): #Look in the unconfirmed if a transaction doesn't
already exist in the blocks#
        for k in BlockChain.blocks["Blocks"][i]["Block" + " " +
str(i)]["Txs"]:
            count = -1
            for j in BlockChain.unconfirmed["Unconfirmed"]:
                count += 1
                if k == j:
                    print("[Warning] An unconfirmed transaction contains a
transaction already found in a block, this transaction is being deleted
from the unconfirmed.")
                    del BlockChain.unconfirmed["Unconfirmed"][count]
                else:
                    continue
    if BlockChain.unconfirmed["Unconfirmed"] == []:
        print("[Error] No transactions in the Unconfirmed, no block has
been created.")
    else:
        block = {"Block" + " " + str(nbB): {"Height": nbB, "Hash": b,
"Prev_Has": c, "Nonce": d, "MerkleRoot": e, "Txs": []}}
        BlockChain.blocks["Blocks"].append(block)

        for i in BlockChain.unconfirmed["Unconfirmed"]:
            BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["Txs"].append(i)
        BlockChain.unconfirmed["Unconfirmed"] = []
        # Read previous block hash#
        BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["Prev_Has"] = BlockChain.blocks["Blocks"][nbB-1]["Block" + " " +
str(nbB-1)]["Hash"]
        # Calculate MerkleRoot of transaction#
        BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["MerkleRoot"] =
Merkle_Tree_Function.Merkle_Tree(BlockChain.blocks["Blocks"][nbB]["Block" +
" " + str(nbB)]["Txs"])
        # Find nounce#
        Cst = str(BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["MerkleRoot"]) + str(BlockChain.blocks["Blocks"][nbB]["Block" + "
" + str(nbB)]["Prev_Has"]) + str(BlockChain.blocks["Blocks"][nbB]["Block" +
" " + str(nbB)]["Nonce"])
        Tcs = hashlib.sha256(Cst.encode()).hexdigest()
        Bts = abs(hash(Tcs))
        while Bts > Target:
            BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["Nonce"] += 1
            Cst = str(BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["MerkleRoot"]) + str(BlockChain.blocks["Blocks"][nbB]["Block" + "
" + str(nbB)]["Prev_Has"]) + str(BlockChain.blocks["Blocks"][nbB]["Block" +
" " + str(nbB)]["Nonce"])
            Tcs = hashlib.sha256(Cst.encode()).hexdigest()
            Bts = abs(hash(Tcs))
            BlockChain.blocks["Blocks"][nbB]["Block" + " " +
str(nbB)]["Hash"] = Tcs
        nbB +=1
```

Picture8. Function that will do the Mining of the Unconfirmed transaction. Check if the transaction doesn't already exist in a block, otherwise erase it. Will create a new block if some transaction is in the Unconfirmed. Calculate the Merkle Root, take the Hash of the previous block. Establish a loop to find the Hash of the new block and the Nonce.

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

7.References

https://fr.wikipedia.org/wiki/Blockchain
https://en.wikipedia.org/wiki/Hash_function
https://en.wikipedia.org/wiki/Digital_signature
https://fr.wikipedia.org/wiki/Digital_Signature_Algorithm
https://en.wikipedia.org/wiki/Merkle_tree
https://fr.wikipedia.org/wiki/Chiffrement_RSA
http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf
http://cacr.uwaterloo.ca/hac/
http://cacr.uwaterloo.ca/hac/about/chap9.pdf
http://cacr.uwaterloo.ca/hac/about/chap11.pdf
https://en.wikipedia.org/wiki/Tiger_(cryptography)
http://stackabuse.com/reading-and-writing-json-to-a-file-in-python/