Tutorial #7

Security in Computing COSC2356/2357

Q1: Why the blockchain is considered as a trustworthy network?

Answer:

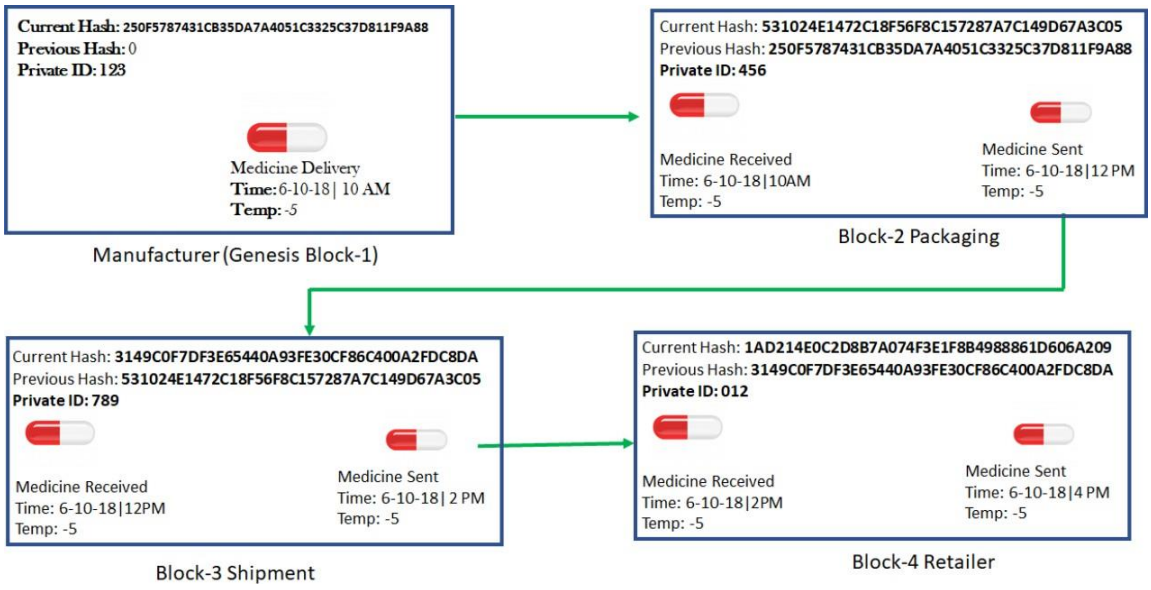
Because blockchains are replicated across a peer-to-peer network, the information they contain is very difficult to corrupt or extinguish

• With a blockchain in place, applications that could previously run only through a trusted intermediary, can now operate in a decentralized fashion, without the need for a central authority, and achieve the same functionality with the same amount of certainty.

• We say that the blockchain enables *trustless* networks, because the parties can transact even though they do not trust each other. The absence of a trusted intermediary means faster reconciliation between transacting parties.

• The heavy use of cryptography, a key characteristic of blockchain networks, brings authoritativeness behind all the interactions in the net- work.

Q2: What happens if anybody wants to tamper block 2? discuss

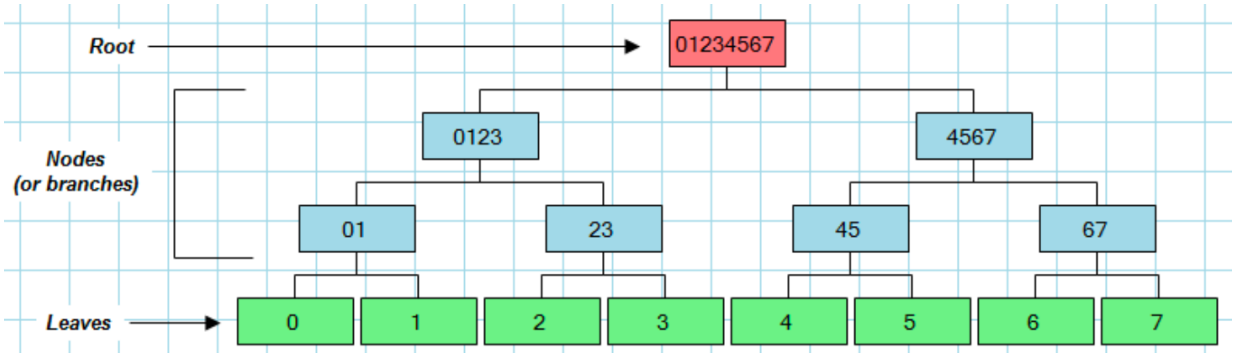


Answer:

The current hash value of block 2 will be changed. Therefore, following hashes should be different. However, the person who is trying to temper the value in block 2 will not be able to

compute the current hash value for block 2. As a result, the shipment will be able to realize that some value has been tempered and discard the transaction.

Q3:



Let's say you are the owner of the record "6" in the above diagram. You also have, from a trusted authority, the root hash, which in our simulation is "01234567". You ask the server to prove to you that your record "6" is in the tree. How does it work?

Answer:

§ What the server returns to you are the hashes "7", "45", "0123" as illustrated here:
§ Using this information (including the right-left flags that are sent back along with the hashes), the proof is that:
•6+ 7 from which you compute 67
•45 + 67 from which you compute 4567
•0123 + 4567 from which you compute 01234567
§ Since you know the root hash from your trusted authority, the proof validates that "6" exists in the tree.

Q4:

Say you want to hash the string "1034". You take a random number and hash it using md5 (usually it is by appending the number to the end of content). You keep trying different random numbers until you find a hash that starts with four 0s. How many iterations do you need to get the first solution?

[Hints: You should start with the number '1' as random number. After trying up to the number '5', start trying from the number '595']

Answer:

You start with say the number 1:

md5(10341) = 859b755563f548d008f936906a959c8f

But this does not generate hash with four leading 0s. So, we increment the number and Try again:

md5(10342) = 7b3564b05f78b6739d06a2ea3187f5ca

We keep trying by incrementing the number:

md5(10343) = 859b755563f548d008f936906a959c8f

md5(10344) = 7b3564b05f78b6739d06a2ea3187f5ca

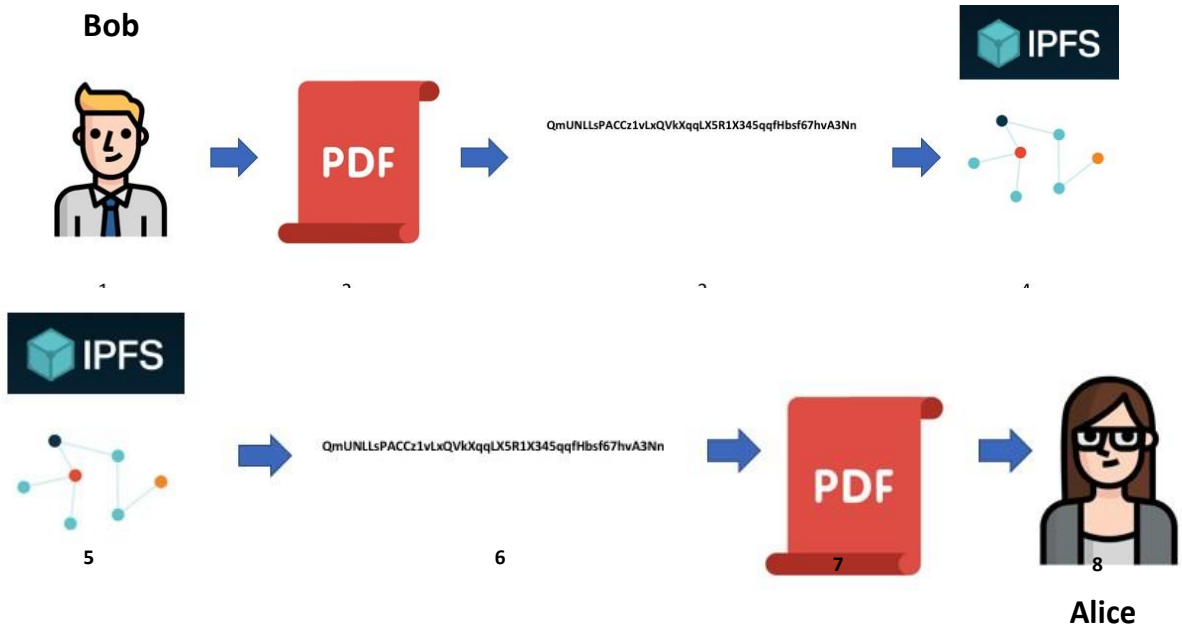
and finally when we append nonce 599 it conforms to our requirement

md5(1034599) = 0000fc70ad0a307d08f88a484dd99cb4

So, after 599 hashes we have a proof of work.

Task-1 (IPFS)

Say, Bob wants to upload a PDF file to IPFS. He puts his PDF file in his working directory. He tells IPFS he wants to add this file, which generates a hash of the file. Now, his file is available on the IPFS network. Now, suppose Bob wants to share this file with his colleague Alice through IPFS. He simply tells Alice the hash of his file. Alice calls the hash from IPFS and she gets a copy of the PDF file. Consider the following figure and demonstrate the above scenario using IPFS.



Answer:

Uploading a file "myfile.txt" to IPFS server by Bob

Step-1: Download IPFS in your computer from the following link according to your operating system and install it:

https://ipfs.io/docs/install/

Follow the instructions for installation given in the above link.

Step-2: Once you have installed IPFS, go to the home directory of IPFS. Now, initialize IPFS with the following command in the command prompt (in Windows OS) or terminal (in Mac OS or Linux):

```
ipfs init
```

You will see the following output:

```
H:\go-ipfs>ipfs init
initializing IPFS node at H:\.ipfs
```

Step-3: On your computer, start your ***daemon*** with the following command:

```
ipfs daemon
```

You will see the following output:

```
H:\go-ipfs>ipfs daemon
Initializing daemon...
Swarm listening on /ip4/127.0.0.1/tcp/4001
Swarm listening on /ip4/131.170.43.242/tcp/4001
Swarm listening on /ip4/192.168.182.1/tcp/4001
Swarm listening on /ip4/192.168.225.1/tcp/4001
Swarm listening on /ip4/192.168.56.1/tcp/4001
Swarm listening on /ip6/::1/tcp/4001
Swarm listening on /p2p-circuit/ipfs/QmQXW8iMNsCHTyv3RoUsExoCJT2xpcY1zchCFdjStiJyke
Swarm announcing /ip4/127.0.0.1/tcp/4001
Swarm announcing /ip4/131.170.43.242/tcp/4001
Swarm announcing /ip4/192.168.182.1/tcp/4001
Swarm announcing /ip4/192.168.225.1/tcp/4001
Swarm announcing /ip4/192.168.56.1/tcp/4001
Swarm announcing /ip6/::1/tcp/4001
API server listening on /ip4/127.0.0.1/tcp/5001
Gateway (readonly) server listening on /ip4/127.0.0.1/tcp/8080
Daemon is ready
```

Step-4: Open another command prompt or terminal and run the following command:

```
ipfs add myfile.txt
```

You will see the following output:

```
H:\go-ipfs>ipfs add myfile.txt
256 B / ? [-------=                                                ] ←
added QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP cipher-text.txt
256 B / 256 B [============================================================] 100.00%
```

Here, QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP is the hash value of the file "myfile.txt"

Step-5: Bob should double check to make sure our file is available on IPFS with the following commands:

```
ipfs pin ls
```

Bob should see the following output:

```
H:\go-ipfs>ipfs pin ls
QmQ5vhrL7uv6tuoN9KeVBwd4PwfQkXdVVmDLUZuTNxqgvm indirect
QmS4ustL54uo8FzR9455qaxZwuMiUhyvMcX9Ba8nUH4uVv recursive
QmUNLLsPACCz1vLxQVkXqqLX5R1X345qqfHbsf67hvA3Nn recursive
QmY5heUM5qgRubMDD1og9fhCPA6QdkMp3QCwd4s7gJsyE7 indirect
QmYCvbfNbCwFR45HiNP45rwJgvatpiW38D961L5qAhUM5Y indirect
QmZTR5bcpQD7cFgTorqxZDYaew1Wqgfbd2ud9QqGPAkK2V indirect
QmejvEPop4D7YUadeGqYWmZxHhLc4JBUCzJJHWMzdcMe2y indirect
QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP recursive
QmPZ9gcCEpqKTo6aq61g2nXGUhM4iCL3ewB6LDXZCtioEB indirect
QmXgqKTbzdh83pQtKFb19SpMCpDDcKR2ujqk3pKph9aCNF indirect
```

The hash exists!

Step-6: Bob sends the hash and file type information (here, "txt") to Alice using email or some other ways of communication so that she can download it from IPFS.

Downloading the file from IPFS server by Alice

Step-1: Alice collects the hash value from email.

Step-2: Using command prompt or terminal, Alice downloads the file from IPFS with the hash value "QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP":

```
ipfs get QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
```

Alice gets the following output:

```
H:\go-ipfs>ipfs get QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
Saving file(s) to QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
 267 B / 267 B [=============================================] 100.00% 0s
```

Step-3: Alice renames the file with extension "txt" and opens the file to use.

Task-2 (IPFS and OpenSSL)

Assume that an owner of a particular file, say Alice, wants to share the file to her friend, say Bob, in a secret way. Alice and Bob use OpenSSL for file encryption and decryption (similar to Task-2 of Tutorial-4), and IPFS for file sharing (similar to Task-1 of this tutorial). For sharing other information (public key and file names), they use emails for the sake of simplicity. Before the communication starts, Bob generates his **RSA public-key** and **private-key** using OpenSSL, and emails the **public-key** to Alice.

Now, Alice encrypts the file with **Bob's RSA public-key** using OpenSSL to generate an **encrypted file**. Next, she uploads the encrypted file in the IPFS-based repository using IPFS command, and receives a **Unique Hash Identifier(UHI)**. Finally, Alice emails the **UHI** to Bob using her email account.

Bob collects the **unique hash identifier (UHI)** for the file from his email. Next, Bob downloads the encrypted file from IPFS-based repository with the file's UHI. Bob uses IPFS commands to download the file. Finally, Bob decrypts the encrypted file with his **RSA private-key** using OpenSSL for using.

Demonstrate the above scenario using OpenSSL and IPFS.

Answer:

Key Generation by Bob

Step-1: Bob generates 2048-bit **RSA private key** using the following command. The private key is stored in a file called **private-key.pem**.

```
genrsa -out key.pem 2048
```

Step-2: Bob extracts **RSA Public key** from the private key using the following command. The public key is stored in a file called **public-key.pem**.

rsa -in private-key.pem -pubout -out public-key.pem

Step-3: Bob emails **public-key.pem** to Alice for encrypting files.

File Sharing by Alice

Step-1: Alice encrypts the plain-text file "**file.txt**" with Bob's public key **public-key.pem** using the following command:

rsautl -in file.txt -out encrypted_file.txt -pubin -inkey public-key.pem –encrypt

An encrypted file called **encrypted_file.txt** is generated.

Step-2: Alice starts IPFS in another command prompt or terminal. Please refer to the instructions in ***Task-1*** for starting IPFS. Next, Alice runs the following command:

ipfs add encrypted_file.txt

Alice finds the following output:

```
H:\go-ipfs>ipfs add myfile.txt
256 B / ? [-------=---------------------------------- ] ←
added QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP encrypted_file.txt
256 B / 256 B [=====================================================] 100.00%
```

Here, **QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP** is the unique hash Identifier (UHI) of the file "**encrypted_file.txt**".

Please note that the hash value of your file may be different.

Step-3: Alice emails the **UHI** to Bob for downloading the file.

Downloading the file from IPFS server by Bob

Step-1: Bob collects the UHI from email.

Step-2: Using command prompt or terminal, Bob downloads the encrypted file from IPFS with the hash value "QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP":

```
ipfs get QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
```

Bob gets the following output:

```
H:\go-ipfs>ipfs get QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
Saving file(s) to QmNx8ckHFRxPSsoHcwsjt3AThUrnfGJfWPWjHbxMXu9sNP
 267 B / 267 B [=================================================] 100.00% 0s
```

Step-3: Bob renames the file as "**encrypted_file. txt**" and decrypts with his **RSA private-key** using the following command:

rsautl -in encrypted_file. txt -out decrypted_file.txt -inkey private-key.pem -decrypt

A file called **decrypted_file.txt** is generated, which is the decrypted file.