RESEARCH ARTICLE OPEN ACCESS

Proof of Storage: A comprehensive framework for Secure Surveillance Storage using Blockchain

Soujanya Duvvi*, Kondapalli Venkata Ramana**
**(Department of Computer Science and Systems Engineering, AU College of Engineering(A), Andhra University, Visakhapatnam -03*
***(Department of Computer Science and Systems Engineering, AU College of Engineering(A), Andhra University, Visakhapatnam -03*

ABSTRACT
With the increase of security concerns all over the world, there is an enormous amount of surveillance data that is being produced by various security devices. This enormous information being populated day by day, makes it a monotonous task for the organization to store as well as to process. Here, distributed environment has a vibrant impact on storing and analysing the data. But sharing of information between various devices raises concerns about the security of the information being shared as any foe can intervene and may mislead the surveillance system by either apprehending or meddling the information. There by we propose a secure blockchain based model for making the information unassailable. In this context, we save the files in a distributed ledger technology called IPFS (Inter Planetary File System). IPFS uses encryption for storing data in the form of a merkle tree. Our proposed model includes various strategies and conservation mechanisms, certificate ascendancies, authentication and overturning of certificates to access the stored data, by using various smart contracts.

***Keywords* -** IPFS, Blockchain, Cryptography, Surveillance, Smart Contract.

Date of Submission: 15-12-2020 Date of Acceptance: 30-12-2020

I. INTRODUCTION

Over the last few years, the procedure of safe guarding the information from illegitimate users to restrict the unauthorized access to valuable data. In this digital world the growth of information is very drastic. Thus, making personal, critical and sensitive information more prone to vulnerabilities, thereby increasing the chances for manipulation or misuse of information. The main source of such information comes from centralized surveillance systems which exist in every place. These centralized systems provide better bureaucracy, but flinches down in offering pliable, assured, computationally competent method of processing critical information.

Blockchain is a distributed ledger technology working towards the decentralization of information stored in it. The storage and organization of data is perceived of by Inter Planetary File System (IPFS) [1]. The problems in centralization are overcome, together by blockchain and IPFS. A smart contract which digitally validates and imposes safekeeping of stored information binds the user for registration and authorization.

In this paper we addressed about the decentralized storing strategies that are unified with blockchain and certificate ascendancy. Authentication of user credentials is taken care of by the smart contract as an initial and mandatory step. The certificate ascendancy is entrenched with the planned system to observe the behavioral pattern of various users using the system by carrying out various operations like Policy substantiation, issuing of certificates, creation and cancellation of certificates, maintaining enrolment authority and substantiation of services.

II. RELATED WORK

M. N. Asghar et. al [3] proposed a model and give solution for visual surveillance data protection based on General Data Protection Regulations (GDPR). In the context of GDPR, the roles of machine learning, image processing, cryptography and blockchain are explored as a way of deploying Data protection by design solution for visual solution data. Nikouei et. al [4] suggested a blockchain enabled scheme to protect the big data generated by numerous surveillance devices, where the data is indexed by encrypting the path between the nodes, thereby reducing the attacks on the small edges and devices. Wong et. al [5] implemented a new video surveillance system with permissioned blockchain, IPFS, CNNs and edge computing to achieve large scale wireless sensor information acquisition and data processing.

Nagothu et. al [6] developed microservice - enabled architecture for smart surveillance system in which the video analysis algorithms are encapsulated into each microservice. Kerr et. al. [7] demonstrates the participation of prototype camera for secured distributed ledger of video streaming. This scheme combines blockchain and digital watermarking for providing trustworthy evidence protection in distributed environments. Jeong et. al [8] created a blockchain network where internal managers play the trusted role. The metadata of the video is recorded in the distributed ledger of blockchain and that further generates a license of the videos. The license stored in the private database of the blockchain restricted to access others. However, internal managers can export the videos by exporting the license.

III. PRELIMINARIES

In this section we present the related knowledge and background for better understanding of our proposed system.

3.1. ETHEREUM: Blockchain is “an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way”.[7] This is a peer-to-peer network of decentralized systems where each and every node hold the replica of the original data in a chained manner. The size of the chain keeps on increasing as more blocks are attached to it. Each block has its own unique id called the hash pointer which identifies the block uniquely. The initial block is called the genesis block. Any operation performed in the network is stored in the blockchain in the form of transactions. These transactions are stored in the form of cryptographically hashed values. The cryptography function employed in the blockchain is a one-way hash function which prevents the data in the blockchain from being tampered, as the blocks are all arranged in a tree structure called merkle tree and any changes made to the data will collapse the entire tree and it has to be built again.

3.2. ETHEREUM: Ethereum is an open source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality.[8] It provides users with various types of accounts to perform the operations on the blockchain. Any operation performed in Ethereum is calculated in terms gas units. Gas can be purchased using ether which is cryptocurrency used in Ethereum or can be mined by the miner. Whenever any transaction is triggered by the user the transaction cost can be calculated as sum of execution cost and execution cost.

$$P_oT = (EC + T\,C) * P \qquad (1)$$

Where P_oT is the price of a transaction, EC is the Execution cost, TC is the Transaction cost and P is the price of 1 gas unit.

3.3. IPFS: IPFS is protocol and also a network designed to create a content addressable, peer to peer method of storing and sharing hypermedia in a distributed file system [3]. This works on the content it stores in the network. Identifiers are generated depending on the content in the file i.e., a document, audio, image, video or any other kind of information that is stored in it. These identifiers are arranged in the form of a merkle dag. IPFS as a network model uses PKI based identity. An IPFS Node is a program that can find, publish, and replicate merkledag objects. Its identity is defined by a private key.

$$\text{KeyGen} \rightarrow \text{PUKey, PRKey} \qquad (2)$$

$$\text{NodeId} = \text{MultiHash(PUKey)} \qquad (3)$$

All hashes in IPFS are encoded with multihash, a self-describing hash format. All IPFS nodes support sha2-256, sha-512, sha3 algorithms. The generated hash values should be deterministic, uncorrelated, unique and one-way. The main functionality of IPFS depends on the Content Identifiers which possess the following structure <cidv1>::=<multibase-prefix><cid-version><multicode-content-type> <multihash-content-address> where <multibase-prefix> is a code (1 or 2 bytes), to ease encoding CIDs into various blocks. <cid-version> is a varint representing the version of CID for upgradability. <multicode-contenttype> is used to represent the content type or format of the data being addressed. <multihash-content-address> represents the cryptographic hash of the content being addressed in the format base58(<varint hash function code><varint digest size in bytes ><hash function output>).

3.4. SMART CONTRACT: A smart contract is computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism. They render transactions which are traceable, transparent, and irreversible.

IV. SMART CONTRACT FOR SECURE SURVEILLANCE STORAGE MODEL

Our model consists of Grilling Team (GT), Enrolment Authority (EA), Certificate Ascendancy (CA) modules. The primary role of GT is to collect all the source files, validate them from all sources. EA's responsibility is to authenticate all the files sent by GT and to store them securely. The role of CA is to administer Issuing Authority (IA), Receiving Authority (RA) and Unbiased Observer Group. The CA is also responsible for maintaining IPFS and governing the flow of information in and out from a blockchain. The operations performed on the proposed model are shown in figure 1 and explained as follows:

- The Grilling Team (GT) collects all the evidences from various sources.
- GT sends source files and UserId to GnuPG.
- GnuPG generates session key and individual key pair for the user.
- Performs encryption on message using the keys generated in step 3
- Adds digital signature to encrypted message.
- Sends back the message generated in 5 and public keys to GT.
- The Chief Investigator (CI) sends his unique credentials to Enrolment Authority (EA) for validation.
- EA authenticates the credentials of CI and communicates back the same to GT.
- CI sends the received digitally signed encrypted documents to EA.
- The EA verifies the validity of the document hash.
- Upon verifying the document hash EA sends these files to IPFS.
- EA updates the transactions to blockchain.
- IPFS encodes the received files, thereby generating a MerkleDag for each file.
- If the document is found valid, then it will be added to virtual file system.
- The CEI places a request to IA to register themselves with the system to issue certificates.
- Upon validating the request, the IA accepts/rejects the request.
- The CErtificate Issuer (CEI) places a request to the Issuing Authority (IA) for registering their certificates.
- The IA sends the files and user details to GnuPG.
- GnuPG generates unique key pairs and session keys.
- Performs encryption on source files using the generated keys in 19.
- GnuPG sends back the required digital signed files and keys to IA.
- The IA sends these files to CA for further processing.
- CA checks if these files are already existing in IPFS and also for their validity.
- CA stores the files to IPFS, if they are not present already.
- CA updates the transaction status to blockchain.
- Recipients register themselves with EA to receive certificates.
- Upon verifying the details, the EA conveys them back.
- Recipients place request to EA for various certificates.
- EA requests CA to issue the requested certificates.
- CA checks with repository. If requested files are available, they will be issued otherwise convey a failure message to EA.
- CA updates the transaction statuses to blockchain.
- Anyone belonging to Unbiased Observer Group wishing to verify the recipient's address, places a request to CA.
- CA upon verifying and validating the credentials provides them the requested details.

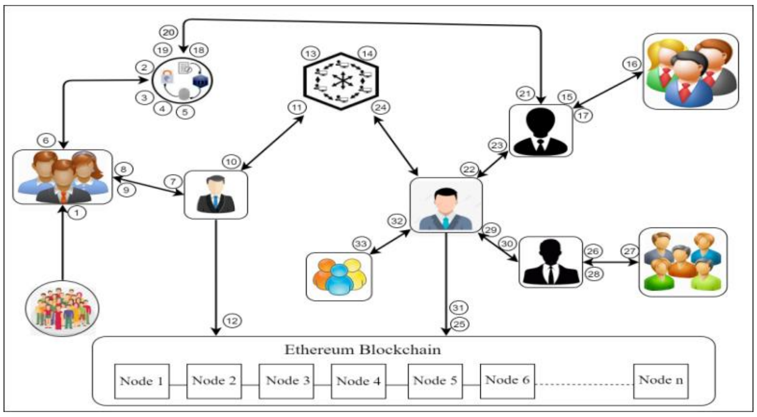

FIGURE 1: Secure Surveillance Storage Architecture

4.1. GNU PRIVACY GUARD: This OpenPGP standard is defined by RFC4880 (PGP). This is an encryption algorithm used to encrypt the data and communications. GnuPG algorithm also possess a versatile key management system which is mainly used in our model. It generates a unique key pair (public and private key) differently for every user, also depending on the input file. The operations of GnuPG can be visualized as in figure 2 and explained as follows:

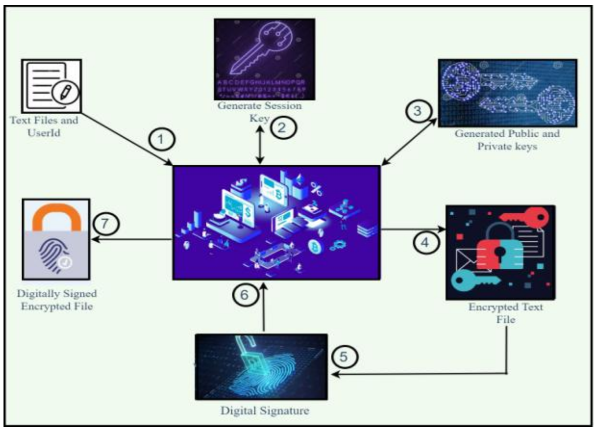

FIGURE 2: GNUPG Architecture

- GT sends the required files and user id to hybrid encryption software.
- HES generates a session key for the corresponding user.
- HES also generates unique key pair for the user in communication with HES.
- Using the keys generated in steps 2, 3 HES encrypts the message.
- Digital Signature and encrypted text file are sent to HES.
- Digital Signature will be applied on the encrypted text file.
- HES sends the digitally signed encrypted file along with the public key of the user back.

TABLE I: Notations used in algorithms

U_{id}	User id value
CT_x	Cipher Text
SC	Encrypted session key
$P_u$key	Public key
CID	Content Identifier
BLOB	Binary Large Object
F_{id}	Unique identity of source file
C_{Hash}	Secure hash value of certificate
S_{addr}	Sender Address
$User_{addr}$	User Address
P_{id}	Peer id
R_{addr}	Recipient Address
I_{addr}	Issuer address
$C_{HashIssuer}$	Secure Hash of Certificate Issuer
H_x	Secure hash of source file
R_{id}	Recipient Address
C_{id}	Unique Certificate Identifier

Algorithm 1: GnuPG Algorithm
Input: Source files, UserId U_{id}
Output: Digitally signed encrypted files, Pair of keys

```
1.  Begin
2.  UserDetails ← fetch userDetails (name, email, security code)
3.  SessionKeys[] ← Generate SessionKey (userId)
4.  if userDetails exist in registeredUsers[] then
5.      dig_Signature ← fetch digitalSignature (User)
6.      keypair[] ← Generate Individual KeyPair (User)
7.  else
8.      Display "User not registered"
9.  End if
10. if keypair[] isNotNull then
11.     recipient_Keys[] ← Generate KeyPair (UserDetails)
12.     $CT_x$ ← Encrypt(files, $U_{id}$)
13.     SC ← Encrypt(SessionKey, $P_u$Key)
14. else
15.     display "Session expired!!"
16. End if
17. End
```

4.2. IPFS
This module receives input files from Enrolment Authority and Certificate Ascendancy. A hash operation is performed on the received input files using a one-way hash function. These files are further encoded and divided into blocks whose size is not more than 256kb. The generated blocks are linked in a merkle dag fashion. Upon receiving requests from Enrolment Authority and Certificate Ascendancy IPFS searches the repository. IF it finds an exact match of the files, the hash value of the file is fetched and send to the corresponding authority else displays an error message.

Algorithm 2: IPFS working Algorithm
Input: Source files
Output: Merkle dag of source files.

```
1. Begin
2. BLOB F[] ← fetch source file from user
3. CID[] ← ContentIdentifier(F)
4. while (Size _Of _Block  !> 256kb) do
5.     CID[] ← ContentIdentifier(F)
6.     GenBlocks[] ← MakeBlocks(F)
7.     IdValue[] ← hash(GenBlocks)
8.     HashCode[] ← sha256(GenBlocks)
```

```
9.      Concat(SizeOFHAshFunction DigestSize)
        to IdValue[]
10.     CID[] ← base58(IdValue)
11.     Generate MerkleDag
12. End while
13. End
```

The core modules are implemented using two smart contracts whose functionalities are discussed below.

4.3. ENROLLMENT AUTHORITY

This authority is responsible for all the operations related with files – Registering a source file, Retrieving the files, Retrieving the hash values and the Peers working on the file or who are associated with it.

- File Registration: The inputs accepted by this module are the category of file, secure hash of the file retrieved from GnuPG module and id of the author who is storing the file into the repository.

Algorithm 3: File Registration
Input: Input File details
Output: Secure hash generated by IPFS

```
1.  Begin
2.  userDetails ← fetch userID
3.  SenderDetails ← fetch SenderID
4.  DocumentDetails[] ← fetch docdetails from
    user(DocumentType, DocumentID,
    UserAddress)
5.  If UserDetails.exists == true then
6.      If UserID.isOwner == true then
7.          addDetails()
8.          addSourceFile(DocumentDetails[])
9.          Increment File Count
10.     Else
11.         Display("Only owner can store
            files")
12. Else
13.     Display("User not Enrolled")
14. End
```

- Retrieving Registered Files: This function enables the users to fetch the files that are registered and stored in the repository. It takes in the unique identification of the file that was generated during the addition of file to repository and fetches the file for the user.

Algorithm 4: Algorithm for fetching registered file details
Input: Unique Identity of the Source file, F_{id}
Output: Details_of_Source (Type, UniqueId, SecureHash)

```
1.  Begin
2.  User_Addr ← fetch User_Address()
3.  If(user ∈ Registered_Users) then
4.      Fetch_Details_of_file($F_{id}$)
5.  Else
6.      Display("Not a registered user")
7.  End
```

- Fetching Hash value of Registered File: This module accepts the unique identification value of the registered file and fetches the hash value of the corresponding document stored in the repository. This hash value can further be communicated among other peers or authorities wishing to verify the details in the source file.

Algorithm 5: Algorithm for Retrieving Evidence Hash
Input: Unique Identity of the Evidence file, F_{id}
Output: Secure hash of the document in IPFS, CT_x

```
1.  Begin
2.  $F_{id}$ ← fetch_unique_Id()
3.  userAddr ← fetch_user_addr()
4.  if(userAddr ∈ Valid_Registered_Users)
    then
5.      if($U_{id}$ ∈ Valid_Registered_
            Evidence_Id) then
6.          Fetch_Details_of_
            file_from_IPFS($F_{id}$)
7.      else
8.          Display("Invalid ID")
9.  else
10.     Display("Not an authorised user")
11. End
```

- Fetching Peer details: By giving this algorithm the unique identity value of the file, it will verify the document details and fetches the author of the file. This value can further be validated as to - if he is a valid permission to be an author, authenticated person from the team, intruder etc.

Algorithm 6: Algorithm for Retrieving Peer ID
Input: Unique Identity of the Evidence file, F_{id}
Output: Evidence Owner's ID

```
1.  Begin
2.  $F_{id}$ ← fetch_unique_Document_Id()
3.  userAddr ← fetch_user_addr()
4.  if($F_{id}$ .exists == true) then
5.      Fetch_owner($F_{id}$)
6.  else
7.      Display("Invalid Document Id")
8.  End
```

4.4. CERTIFICATE ASCENDANCY

This authority deals with the certificates – Fetching certificates from various Issuing authorities, storing them securely, assigning certificates to various Receiving authorities, Maintaining details of Issuers, Recipients and unbiased observer group.

- Certificate Registration: The responsibility of this function is to register a certificate for issuing it to various other recipients on request. This function receives secure hash value of the certificate which is generated in enrolment module as input.

Algorithm 7: Algorithm for Registering Certificate
Input: Secure hash value of the certificate, CH
Output: Unique Certificate id

```
1.  Begin
2.  userDetails ← fetch userID
3.  SenderDetails ← fetch SenderID
4.  DocumentHash ← fetch unique document
    Hash value
5.  if UserDetails.exists == true then
6.      if UserID.isIssuer == true then
7.  Convert the received
                document into required
                evidence format
8.              Update
                Issuer_of_Certificate
                [sku] to sender address
9.      else
10.             Display("Issuer not
                    registered to register
                    certificate")
11.     else
12.         Display "Register user for the
            requested hash code"
13. CertificateRegistered( SenderDetails, CH)
14. End
```

- Issuer Enrolment: The responsibility of this module is to register the issuers for issuing various certificates to the other participating recipients. By enrolling the issuer security will be provided to the system as no intruder can directly fetch or store information into it.

Algorithm 8: Algorithm for Enrolling an issuer
Input: Unique Identity of the Evidence file, F_{id}
Output: Enrolment status

```
1.  Begin
2.  User_Addr ← fetch User Address
3.  S_Addr ← fetch sender Address
4.  F_id ← fetch Document ID
5.  if (S_Addr.isIssuer==false) then
6.          Display "Issuer already registered"
7.  else
8.          Source_details= Fetch_evidence_
                details(F_id)
9.          IPFSHash ← fetch
                SecureHash(sku)
10.         Add IPFSHash to issuers list
11.         Change the status of S_Addr to
                issuer
12.             Issuer_Registered(S_Addr,
                IPFSHash)
13. End
```

- Recipient Enrolment: To interact with the system any participant has to register himself as to authenticate his candidature and to avoid any security breaches with the information stored in the system. Unique ID values will be provided for all the recipients who are enrolled with the system, through which they can interact with this system.

***Algorithm 9*:** Algorithm for Enrolling Recipient
Input: Secure Hash code of Peer, P_{id}
Output: Registration ID

```
1.  Begin
2.  User_Addr ← fetch User Address
3.  R_Addr ← fetch Recipient Address
4.  If (R_Addr.isRecipient==false) then
5.          Display("Recipient already
            registered")
6.  Else
7.      Add IPFSHash to Recipients list
8.      Change the status of R_Addr to
            Recipient
9.          RecipientRegistered(R_Addr,
            IPFSHash)
10. End
```

- Furnish Certificates: Furnishing of certificates is taken care of by the Certificate Ascendancy. Recipients place their request to the Recipient Authority which will be further processed to the Certificate Ascendancy. This Certificate Ascendancy communicates with the IPFS repository to find the requested certificates. The CA transfers the required documents if a match occurs else sends the same information for issuing authority about the required certificates.

Algorithm 10: Algorithm for Furnishing Certificate
Input: Recipient Address R_{addr}, Certificate Hash C_{hash}, Issuer Address I_{addr}
Output: Status of certificate as transferred from Issuer to Recipient

```
1.  Begin
2.  R_addr ← fetch_Recipient_address()
3.  I_addr ← fetch_Issuer_address()
4.  C_hash ← fetch_certificate_hash()
5.  C_hashIssuer ← fetch Issuer_of_certificate()
```

```
6.  if($R_{addr}$.alreadyRegistered != true) then
7.          Display(“Recipient not registered
            to be issued a certificate”)
8.  else
9.          if($I_{addr}$.alreadyRegistered != true) then
10.                 Display(“Issuer not registered to
                    register for a certificate”)
11. Cert_Hash ← fetch_Certificates($I_{addr}$)
12. if ($I_{addr}$ == cert_Hash.$I_{addr}$) then
13.         Update (Issuer, Recipient,
            Certificate, count of Certificates)
14. else
15.         Display(“Issuer not registered to
            issue this certificate”)
16. Certificate_Issued ($C_{Hash}$, $I_{addr}$, $R_{addr}$)
17. End
```

• Fetching Furnisher of a certificate: This module fetches the details of the furnisher of the certificate. This module plays its role when the user has got his certificate but doesn’t know about who has furnished it. It takes in secure hash value as the input and returns all the details of the furnisher.

Algorithm 11: Algorithm for Retrieving Issuer of a Certificate
Input: Secure Identification hash of the file, H_x
Output: Unique ID of owner of Certificate

```
1. Begin
2. $H_x$ ← Retrieve Hash value of file
3. if ($H_x$.isValid and Exists) then
4.         Display(Issuer_ID)
5. else
6.         Display(“Not a valid hash value”)
7. End
```

• Fetching the recipient of certificate: This module helps the authority know about the recent recipient who has taken that particular certificate.

Algorithm 12: Algorithm for Retrieving the recipient of a Certificate
Input: Secure Identification hash of the file, H_x
Output: List of Recipients using it

```
1. Begin
2. $U_{addr}$ ← fetch_address_of_recipient
3. $C_{hash}$← fetch(Address_of_certificate)
4. if($U_{addr}$.alreadyregistered == true) then
5.         $R_{Details}$ ← fetch(Recipient)
6. else
7.         Display(“Invalid details”)
8. End
```

• Fetching all recipients of a certificate: This module helps the authority about the recipients who are currently using a particular certificate. This module is helpful whenever a certificate has to be revoked or changed the current users of the certificate will be informed of the same.

Algorithm 13: Algorithm for Retrieving all the recipients of a Certificate
Input: Secure Unique hash of the evidence in IPFS, F_{id}
Output: List of hashed address registered with that Certificate T[]

```
1. Begin
2. $F_{id}$ ← fetch_Evidence_id()
3. if ($F_{id}$.alreadyRegistered == true) then
4.         T[] ← fetch($H_x$)
5. else
6.         Display(“Invalid details”)
7. End
```

• Fetching all certificates of a recipient: This function fetches all the certificates which are being possessed by a recipient.

Algorithm 14: Algorithm for fetching all the Certificates fetched by a recipient
Input: Recipient Address, R_{id}
Output: certificates list, c_list[]

```
1. Begin
2. if ($R_{id}$ .isAlreadyRegistered == true) then
3.         C_list[] ← fetch_SecureHashValues
                   ($R_{id}$)
4. else
5.         Display(“Invalid Recipient”)
6. End
```

• Fetching Unique Certificate Identity: This module provides all the details of a particular certificate like the address of certificate, Issuer of certificate, Recipient of a certificate, Time of issue of the certificate.
Algorithm 15: Algorithm for fetching all the details of a Certificate
Input: Certificate Identifier, C_{id}
Output: Certificate details, c_details[]

```
1. Begin
2. if($C_{id}$. isRegistered == true) then
3.         C_details ← fetch_Certicate_Details
                   ($C_{id}$)
4. else
5.         Display(“Invalid Certificate id”)
6. End
```

• Fetching all certificates of a furnisher: This function fetches all the certificates which are being issued by a furnisher.

Algorithm 16: Algorithm for fetching all the Certificates being issued by a furnisher

Input: Furnisher Address, F_{id}

Output: certificates list, c_list[]

```
1.  Begin
2.  if(F_id . isRegistered == true) then
3.      C_list[] ← fetch_SecureHashValues(F_id)
4.  Else
5.      Display("Invalid Furnisher")
6.  End
```

- Count of certificates: This function lets the authorities know the total count of certificates stored in the system securely.

Algorithm 17: Algorithm for fetching total count of Certificates

Input: User Address, U_{id}

Output: certificates count, C_Count

```
1. Begin
2. if(U_id.isRegistered == true) then
3.     C_Count ← fetch(count(generated_Certificates))
4. Else
5.     Display("Invalid user")
6. End
```

Our proposed system consists of five special functions which directly interact with the blockchain. These functions send the required parameters that are to be stored in the blockchain's transactions. These special functions are called events. Events are the logs of all the transactions done in the blockchain. So, for every function of the smart contract, we log its details through events. Our system keeps a track of the following events:

- Addition of Source Files: This module keeps a log of all the source files added to the repository. The details include Document type, Document Hash and Author Id.
- Register Issuer: This module keeps a log of details of every registered issuer. IssuerRegistered(issuerAddress, IPFS_hash) function stores the values of all Issuers to the blockchain.
- Register Recipient: This function keeps a log of the details of every registered recipient. RecipientRegistered(recipientAddress, IPFS_hash) function stores the values of all Recipients to blockchain.
- Register Certificate: This function keeps a log of the details of every registered certificate. CertificateRegistered(issuerAddress, IPFS_hash) function stores the certificate details into the blockchain.
- Issue Certificate: This function keeps a log of the details of of every issued certificate. CertificateIssued(certificateID, issuerAddress, recipientAddress) function stores the details of which certificate has been issued by which issuer to recipient as transactions in blockchain.

V. EXPERIMENTAL EVALUATION

5.1. CASE STUDY

This application is designed for gathering, validating and storing the most important files-audio, video which can include CCTV footage, documents, also images, which are very crucial and much prone to tampering. To restrict tampering or morphing of data we propose this new framework where CCTV footage is linked to blockchain technology which is tamperproof and authentic. In this system the restriction is imposed on people who are going to perform the operations directly on the system.

Users of this system will be provided with their own personal credentials (hash keys), upon validating their details with the system it grants them permission to use, store or retrieve the information. These files are stored in IPFS which is a decentralised file storage platform capable of handling larger sized files with great efficiency and less latency by performing multiple hash operations on the blocks of specified size. The files are identified by the Content Identifiers which are result of one-way hash functions. All the operations that are performed on the data are validated by multiple smart contracts. The current version of blockchain as in use is
{"info":{"version":120100,"protocolversion":70012,"blocks":592066,"timeoffset":0,"connections":29,"proxy":"","difficulty":10183488432890,"testnet":false,"relayfee":0.0001,"errors":"Warning: unknown new rules activated (versionbit 1)","network":"livenet"}}

- PERFORMANCE TEST:

We implemented a prototype of the proposed system to study the performance and also to check the attainment of security levels. The prototype is implemented using- node of version 12.7.0, npm of version 6.11.2, go-ethereum as the platform for blockchain and IPFS version v0.4.13. The Solidity Framework and IDE for deploying smart contracts – Truffle is of version v5.0.30, ganache-cli v6.7.0-beta.0 and the scripting language used to write the contracts – solidity is of version v12.7.0. The handling of requests between Ethereum nodes is maintained by web3js is of version v^1.2.0.

We deployed the blockchain node is four machines, and each possess a 2.30GHz core Intel processor with 8 GB primary memory. Some nodes were deployed in various environments like- ubuntu 16 OS, Windows 10. All the machines deployed the storage system for handling all kinds of files possible in the project. The complexity of this experiment is calculated in terms of gas as deploying or execution of a smart contract is done with it. This can also be called as space complexity of the algorithm. Table II shows the amount of cost incurred for each operation that is being performed in the experiment. The metrics used is Gwei – one of the values in cryptocurrencies where 1 ether = 10^{18} wei and 1 ether = 10^{9} Gwei. When this experiment was tested the gas price was set to 5gwei which is equivalent to 5000000000 wei and 0.000000005 ether. **Total cost = number of units of gas consumed * Cost of each unit of gas.**

TABLE II: Cost Incurred For The Operations In The Framework

S. No.	Operation being performed	Number of units of gas consumed		Cost Incurred		Total Cost (in GWei)	Total Cost (in USD)
		Transaction (in gas)	Execution(in gas)	Transaction (in GWei)	Execution(in GWei)		
1.	Source file details Contract deployment	852719	607219	4263595	3036095	7299690	1.298
2.	Adding Source file	194209	164937	971045	824685	1795730	0.319
3.	Fetching Source file Details	22537	1137	112685	5685	118370	0.021
4.	Retrieving Document Hash	23444	2044	117220	10220	127440	0.023
5.	Retrieving Peer ID	23422	2022	117110	10110	127220	0.023
6.	Deploying Certificate Authority	2840335	2110995	14201675	10554975	24756650	4.403
7.	Registering Issuer	707802	686402	3539010	3432010	6971020	1.24
8.	Registering certificate	48888	24032	244440	120160	364600	0.065
9.	Registering recipient	109613	84757	548065	423785	971850	0.173
10.	Issuing Certificate	293937	267673	1469685	1338365	2808050	0.499
11.	Certificate Count	21754	482	108770	2410	111180	0.02
12.	Retrieving all recipients of a Certificate	27002	2146	135010	10730	145740	0.026
13.	Retrieving Certificate Id	24647	3183	123235	15915	139150	0.025
14.	Get Count of all the Certificates of Issuer	24182	1502	120910	7510	128420	0.023
15.	Retrieving ID of Certificate Issuer	25961	1105	129805	5525	135330	0.024

16.	Retrieving the recipient of a Certificate	24728	2048	123640	10240	133880	0.024
17.	Get Count of Certificates Recipient possess	24248	1568	121240	7840	129080	0.023
18.	Retrieve owner details	21868	596	109340	2980	112320	0.02

For the operations performed in rows 3,4,5,11,12,13,14,15,16,17,18 the cost applies only when it is called by a contract. Hence the total cost for executing this framework is 7.997 USD.

- **SECURITY AND PRIVACY ANALYSIS**

By combing BlockChain technology with IPFS and validating them with smart contracts the following advantages will be obtained over the traditional centralized surveillance and storage system. Here we discuss the other advantages of the proposed system.

- **CONTROL OF DATA**

In the initial phase of this system the recipients share their public key to the issuer, for encrypting and sending these files for further validation and storage. The registration authority after validating sends it to IPFS. Upon receiving a file IPFS will save the file as shown in Fig 3.

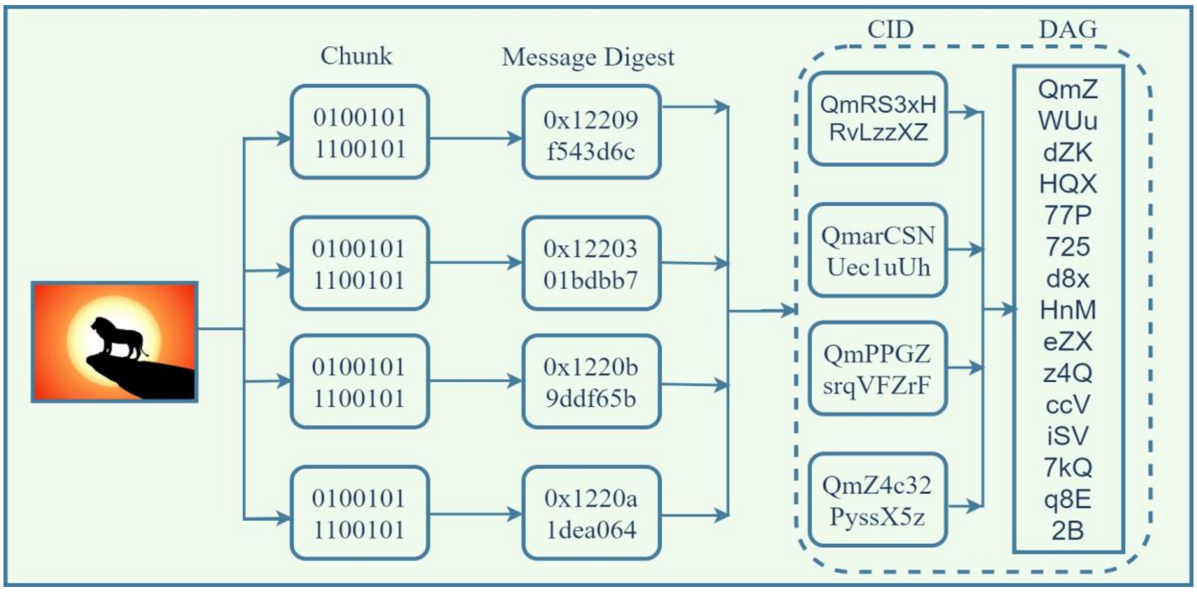

FIGURE 3: CID Representation of Input File

The merkle tree can be represented as

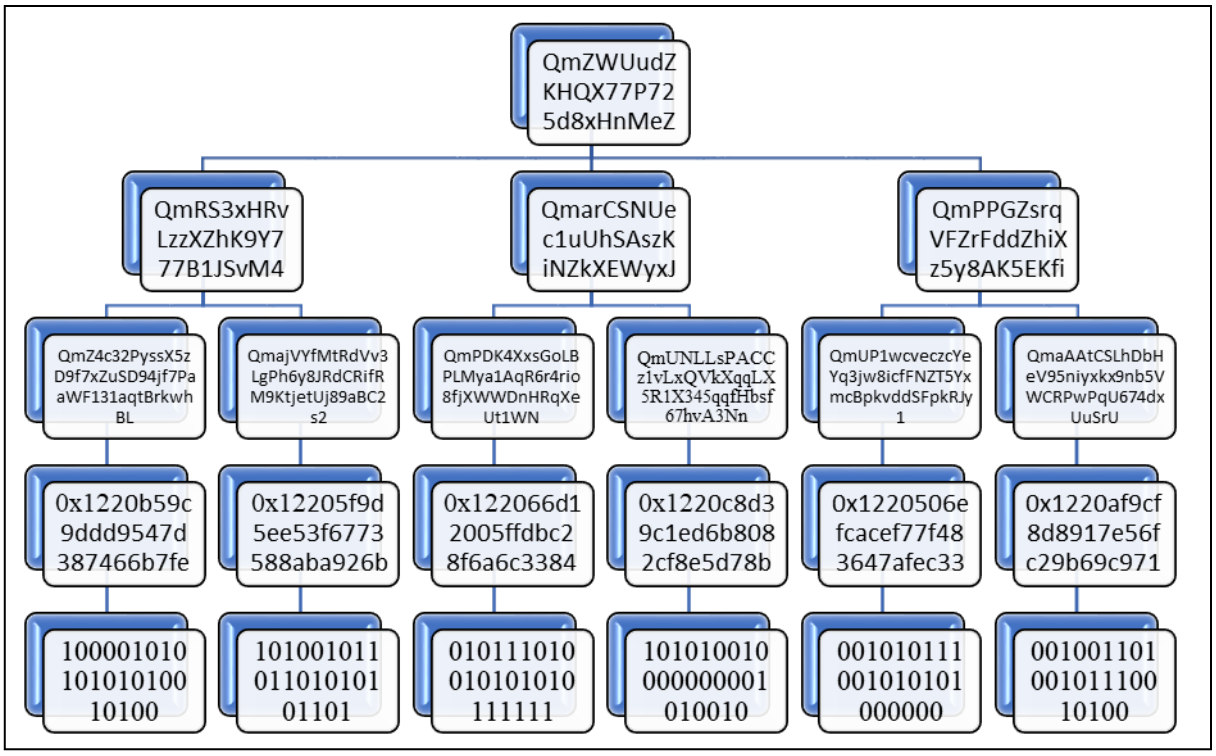
QmZWUudZ KHQX77P72 5d8xHnMeZ
QmRS3xHRv LzzXZhK9Y7 77B1JSvM4
QmarCSNUe c1uUhSAszK iNZkXEWyxJ
QmPPGZsrq VFZrFddZhiX z5y8AK5EKfi
QmZ4c32PyssX5z D9f7xZuSD94jf7Pa aWF131aqtBrkwh BL
QmajVYfMtRdVv3 LgPh6y8JRdCRifR M9KtjetUj89aBC2 s2
QmPDK4XxsGoLB PLMya1AqR6r4rio 8fjXWWDnHRqXe Ut1WN
QmUNLLsPACC z1vLxQVkXqqLX 5R1X345qqfHbsf 67hvA3Nn
QmUP1wcveczcYe Yq3jw8icfFNZT5Yx mcBpkvddSFpkRJy 1
QmaAAtCSLhDbH eV95niyxkx9nb5V WCRPwPqU674dx UuSrU
0x1220b59c 9ddd9547d 387466b7fe
0x12205f9d 5ee53f6773 588aba926b
0x122066d1 2005ffdbc2 8f6a6c3384
0x1220c8d3 9c1ed6b808 2cf8e5d78b
0x1220506e fcacef77f48 3647afec33
0x1220af9cf 8d8917e56f c29b69c971
100001010 101010100 10100
101001011 011010101 01101
010111010 010101010 111111
101010010 000000001 010010
001010111 001010101 000000
001001101 001011100 10100

FIGURE 4: Merkle Tree Representation of Input File

Further upon receiving requests from issuer, receiver or third party the certificate authority is responsible for managing the flow of data between them.

○ **SINGLE POINT OF FAILURE**

By converting the existing centralised system to a decentralized system and combing it with blockchain and IPFS technologies, there no more exists single point of failure in the entire system as both the technologies are designed in a decentralized tamper-proof way. In the blockchain data is stored in a redundant way available with every node which is part of that network. Any modification done to any block can be made out as the remaining nodes contains the original data and by changing information in a single block makes the user to generate the entire merkle tree again. The IPFS works in a peer-to-peer manner with the help of distributed hash table routing technology to maintain a high throughput.

○ **SECURITY AND PRIVACY**

Storing information in blockchain and IPFS gives the user's data utmost privacy as the chunks of the files are hashed and then stored in a decentralized way. The node that stores the files is also unaware as to where the file has been stored and to search for the information that he has stored. Security is provided to the information with the help of unique address that are generated to every user. The unique peerid in IPFS restricts all the users to access every file. Intended users of the file has to register with the system well before to utilize. The oneway hash function is used to encrypt the data that has to be stored in the blockchain. Though a non-intended user gets to know of the hashed information he cannot decode it as it requires private key which is unknown to him and reverse hashing is not possible. All these restrictions and permissions are all govered by the smart contracts Registration Authority and Certificate Authority which work as follows.

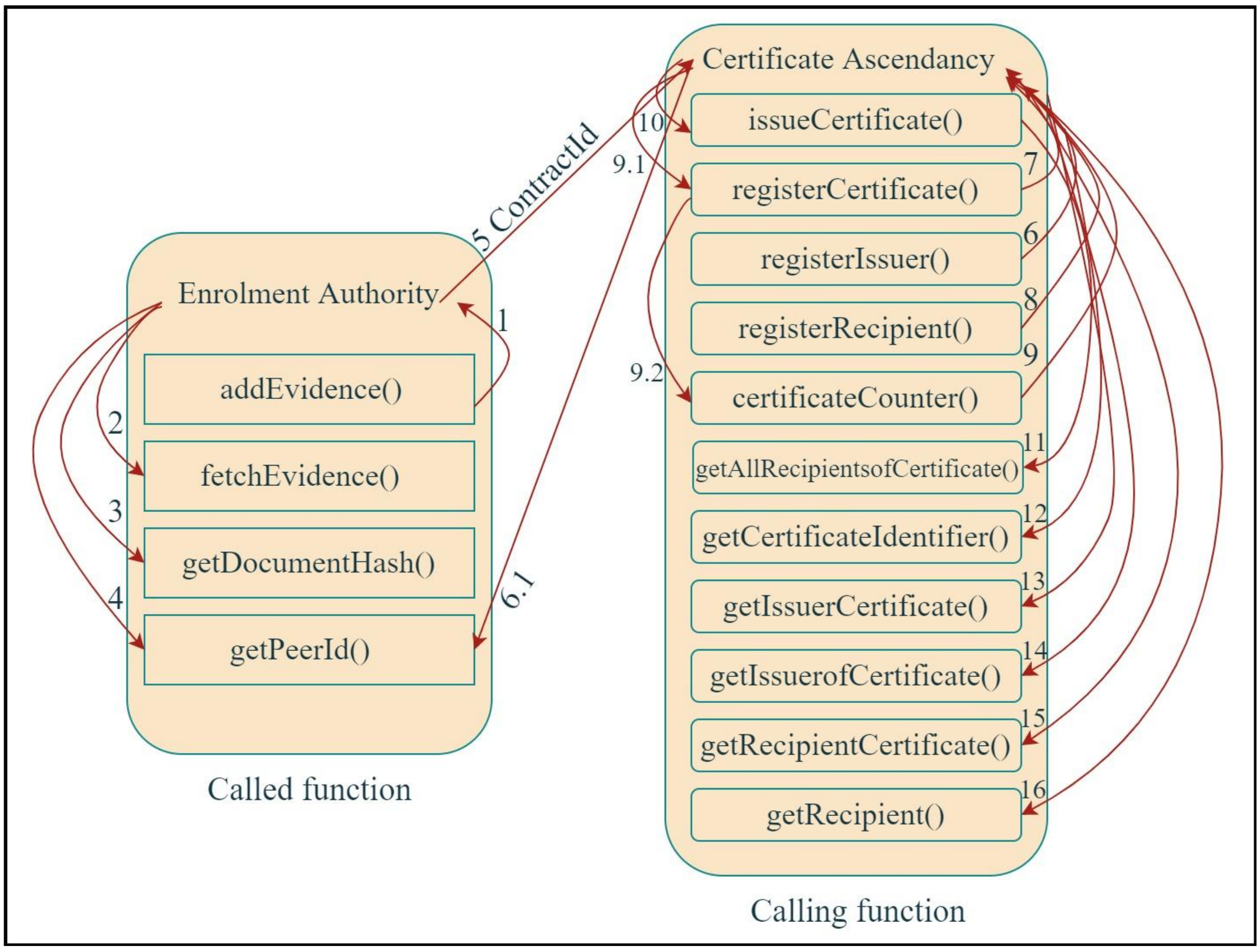

FIGURE 5: Secure Surveillance Storage Model Interaction Diagram

VI. Conclusion

Our Secure Surveillance Storage model proposes a way to provide security for the surveillance data with the blockchain technology. Also, with the implementation of various protocols high level security is provided to the information stored in it. All the operations and permission management are automated by smart contracts. By combining our work with various machine learning algorithms and neural networks, for performing operations like event tracking, object detection, Segmentations etc.

REFERENCES

[1]. Benet,J.: IPFS-Content Addressed, Versioned, P2P File System Retrieved from https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[2]. G. Wood: Ethereum: A secure decentralised generalised transaction ledger. In: Ethereum Project Yellow Paper, 2014, vol. 151

[3]. M. N. Asghar and N. Kanwal and B. Lee and M. Fleury and M. Herbst and Y. Qiao: Visual Surveillance Within the EU General Data Protection Regulation: A Technology Perspective. In: IEEE Access, 2019, vol. 7, pp. 111709-111726. IEEE, https://doi.org/0.1109/ACCESS.2019.2934226

[4]. S. Y. Nikouei and R. Xu and D. Nagothu and Y. Chen and A. Aved and E. Blasch: Real-Time Index Authentication for Event-Oriented Surveillance Video Query using Blockchain. In: IEEE International Smart Cities Conference (ISC2) 2018, IEEE, pp. 1–8. https://doi.org/10.1109/ISC2.2018.8656668

[5]. R. Wang and W. Tsai and J. He and C. Liu and Q. Li and E. Deng: A Video Surveillance System Based on Permissioned Blockchains and Edge Computing. In: IEEE International Conference on Big Data and Smart Computing (BigComp) 2019, IEEE, pp. 1–6. https://doi.org/10.1109/BIGCOMP.2019.8679354

[6]. D. Nagothu and R. Xu and S. Y. Nikouei and Y. Chen: A Microserviceenabled Architecture for Smart Surveillance using Blockchain Technology. In: IEEE International Smart Cities Conference (ISC2) 2018, IEEE, pp. 1–4. https://doi.org/10.1109/ISC2.2018.8656968

[7]. Michael Kerr, Fengling Han, Ron van Schyndel : A Blockchain Implementation for the Cataloguing of CCTV Video Evidence. In: 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) 2018, IEEE, pp. 1–6. https://doi.org/ 10.1109/AVSS.2018.8639440

[8]. Y. Jeong and D. Hwang and K. Kim: Blockchain-Based Management of Video Surveillance Systems. In: International Conference on Information Networking (ICOIN) 2019, IEEE, pp. 465–468. https://doi.org/10.1109/ICOIN.2019.8718126

[9]. S. B. H. Youssef, S. Rekhis, N. Boudriga. : A Blockchain based Secure IoT Solution for the Dam Surveillance. In: IEEE Wireless Communications and Networking Conference (WCNC) 2019, IEEE, pp. 1–6. https://doi.org/10.1109/WCNC.2019.8885479

[10]. Iansiti, Marco; Lakhani, Karim R.: The Truth about Blockchain. Harvard Business Review. Harvard University, 1–11 (2017)