APPROVAL SHEET

Title of Thesis: Automating GDPR Compliance using Policy Integrated Blockchain

Name of Candidate: Abhishek Chandrashekhar Mahindrakar
Master of Information Systems, 2020

Thesis and Abstract Approved: (*Signature of Supervising Professor)
(Dr. Karuna Joshi)
(Associate Professor)
(Information System)

Date Approved: July 16, 2020

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

Abstract

Title of Document: ***AUTOMATING GDPR COMPLIANCE USING POLICY INTEGRATED BLOCKCHAIN***

Abhishek Chandrashekhar Mahindrakar, Master of Information Systems, 2020

Directed By: Dr. Karuna Joshi, Assistant Professor, Dept. of Information Systems

Data Protection regulations, like GDPR, mandate security controls to secure Personal Identifiable Information (PII) of the users which they share with service providers. With the volume of shared data reaching exascale proportions, it is challenging to ensure GDPR compliance in real-time. We propose a novel approach that integrates GDPR Ontology with Blockchain to facilitate real-time automated data compliance. Our framework ensures data operation is allowed only when validated by data privacy policies in compliance with privacy rules in GDPR. When a valid transaction takes place the PII data is automatically stored off-chain in a database. Our system, built using Semantic Web and Ethereum Blockchain, includes an access control system that enforces data privacy policy when data is shared with third parties.

AUTOMATING GDPR COMPLIANCE USING POLICY INTEGRATED BLOCKCHAIN

By

Abhishek Chandrashekhar Mahindrakar

Thesis submitted to the Faculty of the Graduate School of
the University of Maryland, Baltimore County, in partial
fulfillment of the requirements for the degree of
M.S. Information Systems
2020

Dedication

This thesis is dedicated to the people who have supported me throughout my education. Special thanks to my family, my advisor, committee members and friends, they have helped me completely in this adventure.

Acknowledgements

I would like to appreciate my research advisor and mentor Dr. Karuna Joshi for guiding me throughout my research, and for always backing me to tackle challenging problems with an interesting and unique approach. She has been very supportive and motivating whenever things did not work out as per plan. I recognize her helping me out during the tough times in the research and encouraging me to most of the seminars. Attending seminars served me to promote my research projects and also received various insights from many people.

I am thankful to Dr. Zhiyuan Chen and Dr. Sisi Duan for their guidance and their consent to my thesis work. I am blessed to have amazing friends who helped me throughout my graduate studies and have created many memorable moments that I will cherish throughout my life. A special thanks to my best friend Chaitanya Kulkarni, without his emotional and moral support things would've been impossible. Also, gratitude to my colleagues Ankur, Ketki, Ronak, Divya, Adithya who shared the Ebiquity and Knacc lab with me and made the transition very stable. The achievement and existence of my work would not have been feasible without the support of my parents Veena and Chandrashekhar and my sister Priyanka.

Table of Contents

List of Figures

Chapter 1: Introduction

Service Providers are increasingly collecting personal details of their consumers to determine user-behavior patterns associated with market trends, fraud detection or forecasting customer loyalty. Most of these datasets include Personally Identifiable Information (PII) data [34] which are used for big data analytics. This large volume of PII data is often managed and shared using Cloud-based services. Secure usage of this data is critical to prevent the potential for inappropriate dissemination of an individual's private information. Many regulatory and standard bodies have released data protection regulations like the European Union's General Data Protection Regulation (GDPR), US Health Insurance Portability and Accountability Act (HIPAA), Payment Card Industry Data Security Standard (PCI-DSS), etc. to ensure correct private data usage[32][36][37]. It is the responsibility of the providers to adhere to these data protection regulations and ensure the security and privacy of the consumer data they collect.

Providers acquire the consent of their consumers to use and share their PII data via legal contracts like terms of services and privacy policy documents. Currently, these privacy policies are text-based documents containing legal jargon that requires significant manual effort to parse. The data protection laws are also text-based reports that often cross-reference various sections in the same document or other laws. Hence, it is challenging to determine in real-time if an action on the data will result in privacy violation. To automate the process of validating the privacy policies they should be made machine-readable.

We have developed a novel approach to capturing the knowledge embedded in the policy and regulatory documents in the form of semantically rich knowledge graphs that is machine-processable and can be reasoned over to automatically validate every data usage operation. This was accomplished using Semantic Web technologies, Natural Language Processing (NLP), and Text Mining. While we are generating knowledge representations of various policies and regulations, for this paper we have considered only the GDPR. We have integrated our GDPR knowledge graph with Blockchain technologies to have an audit log of every transaction and the corresponding GDPR policy that authorizes the operation. Every data transaction is traced by the smart contract and depending on the validation of the transaction the blocks will be formed to store transaction data. This system is private and permissioned as it includes Semantic Web, Ethereum, and encryption techniques. Anonymized datasets are distributed over the network, this dataset does not reveal the personal details of the consumer and hence helps in avoiding the breach of data.

Contribution : Extending our previous work [33], we parse knowledge graph for transaction authentication and create blocks over blockchain network. Before a transaction is invoked, a SPARQL query is executed which checks with the privacy ontology for permissions. The participants are initially registered on the network using ganache-cli. Ganache-cli is a private blockchain network for Ethereum development. Once the consumers and providers are registered on the network, transfer of data is permitted. For every successful transaction, a block of data is created with Transaction Hash, Block Number, Gas usage , Block Time and Result. As blockchain is an emerging technology and the process of storing and retrieval of data is much different than the mature enterprise

databases, complying with all regulations is difficult to achieve. Hence, we plan on extracting and storing block data in an external encrypted file.

For future references, if the user desires to know the details of the actions, the user needs to have the public key to decrypt the file including the Transaction Hash. Once the Transaction Hash is received the user can query the blockchain and retrieve results. In this paper, we outline our approach in detail and include the results of our initial implementation. Section II describes the background and related work. In Section III, we describe the methodology and architecture of the system. In Section IV we describe the experimental results. Section V has a conclusion and Section VI covers our ongoing work.

Chapter 2 : Related Work

In a service environment, consumers and providers need to be able to exchange information, queries and requests with some assurance that they share a common meaning. This is critical not only for the data but also for the data protection policies followed by service consumers or providers[33]. Nasr Al-Zaben et al.[11] proposed a data tracking system where the user should have the knowledge of the location of data consisting of PII which can be used to easily track the user information. They also defined Potential Personally Identifiable Information where the information may be used to identify an individual. The paper also proposed an architecture where the user shares hash values of PII, NPII (Non-Personally Identifiable Information) and PPII (Potential Personally Identifiable Information) to the controller and similarly the controller shares it to the processor. A block is created on successful completion of transaction [11]. A list of PII are mentioned below in Table 1 which were illustrated by Joshi et al. [15]. Zyskind et al. [16] described a decentralized personal data management system where users can track and control their data. Their system can be used to perform instructions like storing, querying and sharing of data. The type of transactions which are used by their system is Taccess for access management and Tdata for storage and retrieval of data. The data is encrypted and stored in the off-chain database and only the pointer is stored on the public ledger.

Personally Identifiable Information (PII)
Personal Details, Address, Phone, Personal Identification Number, Personal Characteristics, Property Details, Computer Assets, Geographical Indicators, Employment Information, Medical Information, Education Information, Financial Information

Figure 1: PII Data of a user

A. Ethereum Blockchain

Blockchain is a peer to peer distributed ledger technology which is being used in many scenarios which relates to trustable, confidential, secure and auditable platform. We have used Ethereum, which is an open-source software used to build our architecture. The Ethereum framework is used as a protocol for business to business or business to customer transactions. A useful feature of blockchain over other technologies for storing transactions and its data is non-repudiation. This feature can be used to verify the transactions using digital signatures and public-key infrastructures [14]. Ethereum is a decentralized blockchain platform that runs smart contracts. Decentralized applications are executed on this platform. The two types of accounts on Ethereum network are Externally Owned Accounts which are controlled by private keys and Contract Accounts controlled by the code in the contract. In the blockchain system, there is no central authority all the nodes are peer nodes and connected with each other. The decentralized consensus protocol which is Proof of Work is used to maintain the integrity between all the nodes. All the data which is stored in the blockchain needs to be in sync , it any of the data in any of the block is not in sync it is a bad block which is then removed from the chain. In the Proof of Work concept, all the miners solve a math's problem. This problem is created by the person who created the chain. The first one to solve the problem wins a reward and his block is added to the chain. The answer to the PoW problem is a hash. The problem shouldn't be

complicated to keep the work accurate and speed of the blockchain system. But if the problem is too easy then the chances of vulnerabilities, DoS attacks is more. Miners solve the puzzle, form the new block and confirm transactions. This is how the consensus between the nodes is maintained.

The Smart Contract refers to the code in the contract. The code in the contract is executed only when there is a transaction. The method of mining includes adding a block to the network. This happens after every successful transaction. Only members who are registered on the network are permitted to initiate mining. The block will be rejected when the authentication process fails. Ganache-CLI is an Ethereum client that connects with local decentralized applications. To write events over the blockchain network we have used Truffle framework. Truffle is the most popular Ethereum framework to create and compile solidity contracts. Truffle uses solidity compiler which can be used to compile and deploy smart contracts [40]. Everything we write to a blockchain network is immutable it can't be removed and hence we have used ganache-cli which is very useful for testing and development. Ganache-cli is an in-memory based virtual blockchain. Once the ganache is closed all the accounts will be destroyed. A ganache network creates 10 accounts by default and their private keys, it creates mnemonic for that particular network. The mnemonic can be used to reuse the same network in the future. The network can be initialized on any listening port. The default port is 8545 and ganache starts 10 accounts which can be used for recording the stakeholders which will be used for transactions in the system.

When it comes to security it is composed of three components: Confidentiality, Integrity, and Availability. The blockchain serves all three components. 1. Confidentiality means the state of keeping an entity secret; all the blocks in the blockchain are encrypted

and stored in hash format; it satisfies confidentiality. 2. Integrity implies the accuracy, completeness, and consistency of data throughout its lifecycle; blockchain is immutable, once data is stored on the chain it cannot be changed and it stays throughout the chain's lifetime. 3. As blockchain is decentralized, the storing of data is not dependent on any single block or peer. This way blockchain is always available and the system does not need to rely on any third-party service[33]. Information security practice follow the CIA Triad i.e. Confidentiality, Integrity and Availability. Blockchain technology does not enforce confidentiality but implements integrity and availability. Blockchain being decentralized, the data elements are transparent to all the individuals who have registered on the network. Because of this the confidentiality element in this technology is not enforceable and with this any user on the network can read and write transactions to the blockchain. But once the data is written on the network it cannot be edited , so the integrity of elements is enforced. Also, blockchain and its data being on the network all the time it complies with the properties of availability.

B. Hyperledger Fabric

Fabric was one of the blockchain types we had used in our experimental research. The Fabric framework is used as a protocol for business to business or business to customer transactions. Chaincode and smart contract are the key features of Hyperledger Fabric. A useful feature of blockchain over other technologies for storing transactions and its data is non-repudiation. This feature can be used to verify the transactions using digital signatures and public key infrastructures. Hyperledger Fabric is a permissioned blockchain, nodes on the chain are identified based on their registration for the membership service. We will now

briefly overview the architecture of Hyperledger Fabric. It is composed of three components mentioned below [38]:

- **Membership Service**: As Fabric is permissioned all the participants on the network are managed using Membership Service Provider. All participants are identified by Certificate Authority using Public Key Infrastructure.
- **Ordering Service**: Orderer is one of the nodes on the network and broadcasts transactions.
- **Peers**: They are the basic elements of a blockchain network. Fabric works on endorsement process and all the transactions take place on the basis of the endorsement policy. A few peers work as endorsing peers and interact with the clients by sending and receiving transactions results.

Once a transaction takes place Fabric has three phases through which the transaction process is completed according to the order-execute model.

1. **Execution Phase**: The endorsing peer receives a signed transaction from one of the clients. A chaincode function is invoked and the endorsing peers verify the further points: 1. The proposal is correctly defined, 2. The transaction proposal has not been executed in the past to avoid redundancy, 3. The signature used for the proposal is valid, 4. The client who submitted the request is a valid client. Once all the four points are verified, the chaincode function is invoked and the arguments are submitted.
2. **Ordering Phase**: When enough number of endorsements are received by client, the client combines all the proposals and sends it to the ordering service. A block of transaction is created and delivered to all the peers on the chaincode.

3. **Validation Phase:** When the block of transaction is received by peers they are verified, and the endorsement policy is matched with the current state of ledger. Once successful, the peer appends the block of transaction to chain.

Hyperledger Fabric has a good performance, low latency and the delay between transaction proposal, validation and ledger update is similar to runtime of primary-backup replication of data entity in replicated databases with synchronization through middleware. Kazuhiro Yamashita et al. [38] covers the potential risks in the languages used by Smart Contract developers like Go, Node.js and Java.

C. Smart Contract

The smart contract is created to interact with the blockchain. The smart contract cannot be altered or manipulated once deployed. When the smart contract is deployed three interfaces are loaded - init, invoke, and query. Init is used to initialize the contract. A query is initiated when the end-users query the blockchain. Invoke is called when the user wants to insert data on the blockchain. When the smart contract is compiled two variables are created – ABI and Bytecode [24]. Contract Application Binary Interface is used to communicate with contracts. In a smart contract, a get represents a query to retrieve information about the current state of a business object, a put is used to create a new business object or to modify an existing object in the ledger world state and a delete represents the removal of a business object from the current state of the ledger, but not its history[30]. Critically, in all cases, whether transactions create, read, update, or delete business objects in the world state, the blockchain contains an immutable record of these changes.

Smart contracts are referred to as chain code, this chain code helps you to read and update data on the blockchain ledger. Smart contracts are invoked from peers on the ledger and then instantiated on channels. All the members of the blockchain who want to submit transactions or read data by using smart contracts need to install the contract on the peer. The smart contract is determined by name and version of the contract and hence it should be the same throughout the network for consistency. Only one network member needs to instantiate the smart contract. If a peer with a smart contract installed joins the channel where the same contract is already instantiated, the smart contract container starts automatically [29]. A smart contract outlines the rules between different organizations in executable code. That's because a smart contract can implement the governance rules for any type of business object so that they can be automatically enforced when the contract is executed. For example, a smart contract can ensure that a car is delivered in a specific time frame or a set of transactions is executed on prearranged terms and conditions. Most importantly, however, the execution of a smart contract is much more efficient than a manual human business process[30]. Emanuel et al. [31] proposed a smart contract language that is human readable and also binds all the legal conditions. They coined the domain-specific language as SmaCoNat with illustrated examples to show that language enhances readability and safety without violating any natural language specifics. Chun-Feng Liao et al. [29] presents a service platform that helps in the development, testing, and deployment of smart contracts.

Decentralized app: These are the applications which use smart contracts. The goal of Dapp is to have a User interface to your smart contract. These apps can be run on central servers or can also be executed on top a peer node. Hyperledger Fabric follows endorsement policy

and hence enables confidentiality through its channel architecture. On a Fabric network, a channel is created similar to a network overlay. Only those nodes that participate in a channel have access to the smart contract (chaincode) and the data transacted, preserving the privacy and confidentiality of both[27].

Reasons to choose Ethereum over other Blockchains:

Ethereum is a public blockchain network and focuses on running programming code on the decentralized application and smart contracts through its virtual machine (EVM).By querying to Ethereum, we can retrieve results of a block using the block hash or transaction hash. The communication between consumers and providers is efficient and it takes seconds to create a block.

D. General Data Protection Regulation (GDPR)

The GDPR is lately proposed by the European Union as a privacy regulation. Whenever any personal data is distributed between two parties the transaction should comply with the rules and regulations of GDPR. The law determines that personal data processing is prohibited unless it is allowed by law or has the consent of the owner [10]. The data owner also has the right to invoke the consent any time after the data is shared. The withdrawal of permission of personal data must be easy and the data should be erased from all the locations. According to GDPR Encryption, all the data which is processed in our system will be encrypted and stored on the database. This minimizes the risk of cyber-attacks and data leakage. Personal data can be identified as the data which directly identifies a person or an entity. The private data can be either subjective like name, address, contact number, or objective like opinions, suggestions, and expressions using which a person can be correlated. The processing of personal data should also be taken place in

compliance with the rules and regulations of the contract. All the details related to the processor, controller, and categories should be defined. The right to be forgotten regulates the erasure obligation. This defines that the personal data needs to be erased whenever it is no longer required. The controller and the processor are obliged to delete the personal data when processing is complete, and when the data is no longer required. We have planned on storing the PII data on an external storage , whenever a request for erasure of personal data is invoked, we delete the data which is stored externally and all the keys related to it. Once the transaction hash and Block hash is erased, we cannot access the block data. Lavanya et al. developed a data compliance ontology which integrates with the knowledge representation of GDPR[35]. Semantic Web Ontology Language is utilized by the author to construct a knowledge graph that will be consumed by our proposed system for implementation. The GDPR rolled out regulations related to sharing and storing of personal data with respect to the consumers and providers. The GDPR does not care about where the data is stored but the data providers and consumers must follow regulations.

The GDPR rules which are considered for consumer and provider for our system [35]:

Processing of personal data: The personal data collected should be stored and processed transparently. The collected data should not be used for purposes not defined in the policy document. Under the Rights of the EU Data Subjects, Know Your Customer(KYC) is defined wherein personal data is consumed for processing purposes. All the data which is consumed by the system which is PII needs to store with the consent of the user. The data should be stored complying with all the policy rules defined in the policy document. The stored PII should always be accessible by the owner of the data.

Responsibility of Consumer: The consumer should be capable enough to implement methods that are in accordance with the regulation. The consumer needs to give PII data to use all the services provided by the provider. Hence all the requested data needs to be submitted. Before submitting the PII data the consumer should be aware of all the policy documents and related rules defined in the document with respect to the processing, sharing and storing of PII data.

Data Protection and by Default: The organizations should collect and store data only for purposes mentioned in the policy document and all the transactions should be recorded. The data used in the application should be pseudonymized, anonymized or encrypted. Pseudonymized data is the data which processes the PII data and removes data which directly recognizes the user. This pseudonymized data cannot be reidentified to track the user, this complies with GPPR rules.

Responsibility of Provider: The provider should hold onto confidentiality obligations and also follow the rules while appointing sub-providers. The lawful consent of the provider and sub-provider should be followed to track and manage data between data subjects, processors and controllers. The provider when consumes PII data it needs to make sure that it is compliant with all the rules defined in the policy document. Pseudonymization is defined within the GDPR as "the processing of personal data in such a way that the data can no longer be attributed to a specific data subject without the use of additional information, as long as such additional information is kept separately and subject to technical and organizational measures to ensure non-attribution to an identified or identifiable individual" (Article 4(3b)). Anonymization is the process of removing personal identifiers, both direct and indirect, that may lead to an individual being identified.

Processing under consumer authority: The processor should follow instructions presented by the consumer while processing data and should also be conscious of the data provenance. Every organization assigns a Data Protection Officer (DPO) which handles all the policy documents and makes sure that the transactions are complaint with the policy. In our system, we have designed a trusted controller which works like a DPO. This trusted controller manages all the transactions happening through the system, manages all the keys assigned to the registered consumer and provider. It also preserves the smart contracts which are used for all the transactions in the system. Whenever a transaction needs to be executed , our trusted controller makes sure it is compliant with the policy which are defined in the knowledge graph. The knowledge graph which is related to the consumer and provider is compared real time and then the result is provided to the transaction. Based on the result , the transaction is either approved to proceed or denied.

Summary of GDPR obligations over Consumer and Provider [32]:

Consumer is obligated to the below regulations:

1. Notify about Personal Data Breach to Supervisory Authority.
2. Communicating about personal breach to data subject.
3. Carry out Data Protection Impact assessment (DPIA).
4. Consulting Supervisory Authority before processing if DPIA shows high risk.
5. Appointing Data Protection Officer

Provider is obligated to the below regulations:

1. Support consumer during data breaches
2. Processing data as per consumer instructions
3. Maintain records of all processing activities

4. Provide sufficient data security to consumer
5. Implement Privacy by Design / Default
6. Removing all the personal data after end of provision
7. Notify data consumer for subcontracting

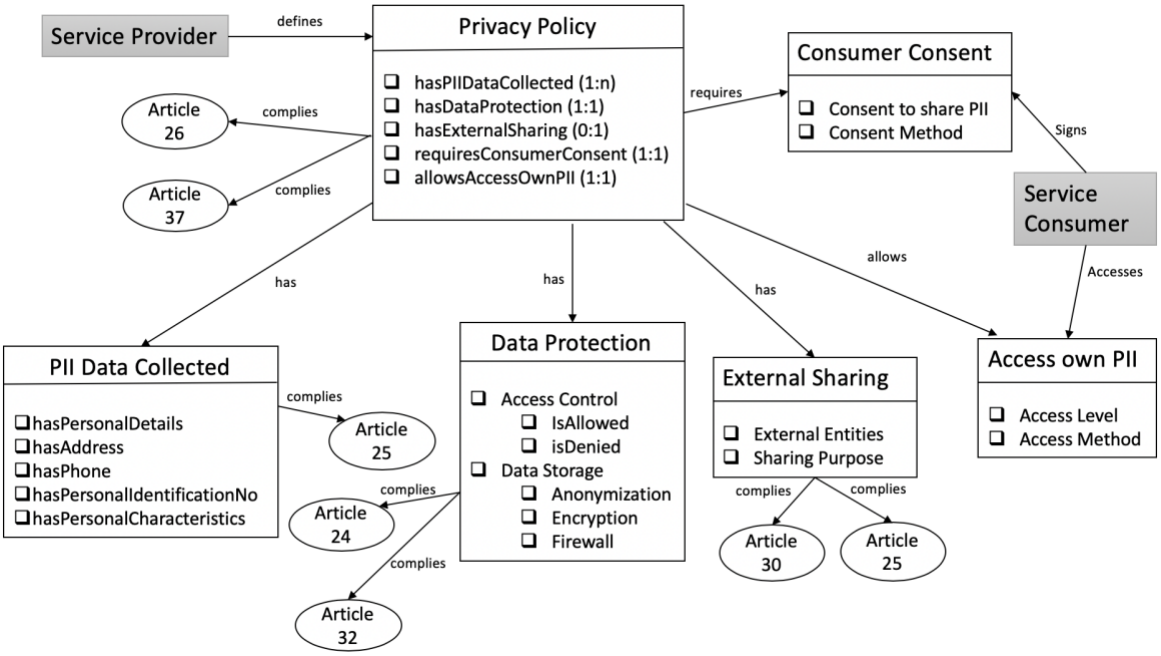

Figure 2: Snapshot of the Privacy Policy Knowledge Graph for System

Chapter 3: Methodology

Privacy Policy Ontology

Joshi et al. [15] proposed classes of an ontology that can be used as components in our system. The author recognized that there should be at least one instance of consumer and provider consent to access the PII data.

The classes are described below:

Collection Purpose: The class defines the purpose and scope of the data which is a collection in the form of PII. The class should also contain the information which is being used to transform data like combining it with other datasets and creating an aggregated list. The document which is referred to as policy document must also have mentioned the duration of the storage of PII data.

Data Protection: The class defines the access control and data storage policies. The rules defined in the access control class are isAllowed or isDenied; this class is based on the policy definition of the privacy policy and sets the attribute. Date Storage class defines the method in which the data is stored which can be either anonymized data, encrypted data. The Firewall in the data protection class can also define which all IP's are allowed to access the data.

External Sharing: This class defines the policies defined for sharing the data externally and the purpose related to it.

PII Data Collected: The attributes which cover the personal data include name, address, telephone number, and various identity characteristics. Many other details can also be used as PII data as it might reveal the user information like the insurance details, financial and education details. All privacy policies will have an instance of PII data connected to it and

might have multiple instances as well, as many times a customer might change their telephone number and the provider might want to store both the instances and hence multiple versions of details in small amounts can be stored.

Data Storage Controls: This class mainly takes care of access control and storage controls. The ontology addresses role-based access control and attribute-based access control.

Consumer Consent: The privacy should explicitly specify the storing and maintaining of the PII data collected from the consumers. The permission to share this data should also be approved by the consumer before the provider shares it.

Access own PII: The consumers should be able to access their own PII data. The method used to access the PII should also be mentioned in the document hence clearly satisfying the terms and conditions of the privacy policy.

Joshi et al. [15] also covered various standards and guidelines for data privacy policy by multiple organizations like the National Institute of Standards and Technology(NIST), European Union Data Protection Standards, etc. The key points identified by the author were 1. Explanation of how businesses collected and used personal information, 2.Age restriction of collecting data, 3. Make new customers aware of the privacy policy and data control, 4. Purpose of the data collection, 5. How is the data which is collected secure, 6. How users can access their collected data.

In this paper, we have used Semantic Web, Natural Language Processing techniques, Ethereum, and AES for encryption to automate the process of data transfer between either consumer to provider or provider to provider in compliance with GDPR. Our application uses privacy policies ontologies and GDPR knowledge graph which are

stored on the cloud. As we know that basic blockchain technology is not compliant with GDPR, we are using blockchain and smart contracts to store, track, and authorize transactions. GDPR Article 25 - Data protection by design and default, one of the items in this article includes encryption which our system follows. Article 24 - Responsibility of the Controller, it is to process data in accordance with the regulation, our system makes sure all the transactions are in compliant with the policy ontology. Article 30 –Records of Processing Activities, as blockchain is immutable all the transactions (valid, invalid) are stored on the network using Transaction Hash can be retrieved. Article 37 – Designation of Data Protection Officer, the trusted controller in our system works like an automated Data Protection Officer where it verifies each transaction with policy ontology. We have identified a few steps in our process and detailed them below.

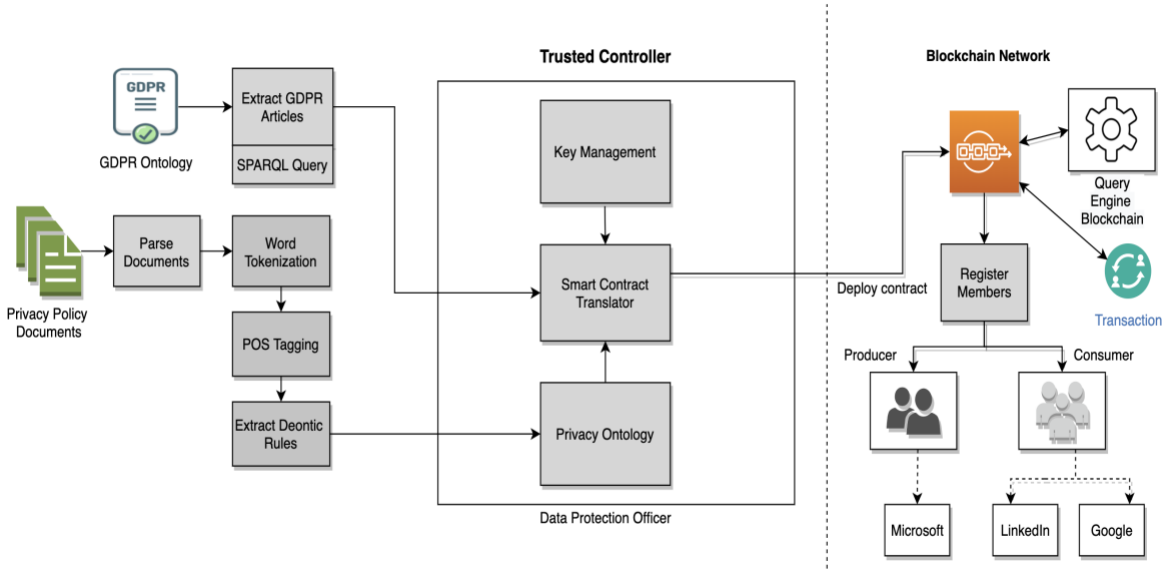

Figure 3: Architecture of the System

Key Modules of the system:

Parse Privacy Policy : The privacy policies are consumed from either online or on the local repository. These privacy policies are in text format and they are converted into an owl for classification.

Extract Key Terms and Rules: The ontology is loaded to perform classification and also to create new instances for further implementation. The classes in ontology can be added or removed. A relation consists of tuples which are subject, predicate, and object. The documents are tokenized, then they are tagged based on part of speech like noun, pronoun, adjective, verb, adverb, etc. The deontic words are extracted from every sentence to create a deontic rule which will help the system in defining if the sentence claims to be a permission, obligation, dispensation, or prohibition with respect to the subject. Obligation can be defined based on the words – should, shall, must. Permission and Dispensation based on the words – can, may, could. Prohibition can be based on the words – can't, must not, may not. The rules are extraction for Permission and Prohibition.

The modal verbs present in all the documents help in identifying the text into Permission, Obligation, Prohibition or Dispensation.

- Permission: Permissions are expressions or rules that describe the rights or authorizations for an entity
- Obligation: Obligations expressions are the compulsory actions that an entity must accomplish
- Dispensation: Dispensations that describe optional expressions and describe non-mandatory conditions

- Prohibition: Prohibitions are the expressions that specify the actions which are prohibited

The following grammar rules are used for classification [41]:
Permission:
< Noun/Pronoun > < deontic > < verb >
Prohibition:
< Noun/Pronoun > < deontic > < negation > < verb >

Extract GDPR Articles: A GDPR ontology knowledge graph is created and only articles related to consumer and provider are extracted using a SPARQL query. SPARQL is an RDF query language, it is a semantic query language used to extract details from an ontology. The query extracts the articles which are further used for the reasoning of the smart contract translator.

Trusted Controller

Key Management: Public key infrastructure is used in blockchain technology which helps in authenticating the network and ensures integrity. We use Ganache, previously known as Testrpc, this sets up 10 default Ethereum accounts with their private keys and preloads them with 100 simulated Ether each.

Privacy Ontology: Once the extraction of key terms and rules is completed a privacy ontology is created which is referred by all the future transactions. Every transaction needs to be compliant with the privacy policy regulations and hence when a transaction is executed the initial step is to check if it has the permission. Once the permissions are verified, the transaction is successfully executed.

Smart Contract Translator: A smart contract is a computer protocol that has methods defined based on the access policies. The contract has methods that are used to execute an

operation on the blockchain. If the operation is not defined in the contract, the transaction is failed. Every transaction on the system calls a method on the smart contract which verifies if the operation is permitted.

Register consumer/provider on blockchain: The stakeholders are registered on the blockchain network and only registered stakeholders can initiate a transaction. The registered members are assigned an account number and a private key. This account number is unique to the member and is used for all the transactions. There can be multiple users on the network as long as the unique account numbers and private keys are valid.

Write on Blockchain: Once a transaction is successful the result of the transaction is used as an input to create a block over the chain. The block consists of transaction details and a hash. All the transactions which are executed on the chain are immutable. Hence nothing can be deleted or modified once it is written. The blocks are protected, the transaction receipt can only be revealed when the user is aware of the transaction hash or the block hash.

Extract Data off-chain: The PII data on the blockchain network is immutable. It cannot be updated or deleted. Hence, we extract the transaction data after every transaction and store it in an encrypted file. The file is further stored on a database. We propose storing data off-chain to accomplish the GDPR rule. The rule 'Right to be Forgotten' can be partially achieved if we delete the encrypted copy of the data or deny sharing of a public key with sub-providers, the provider will not have the transaction details or the data of the consumer.

Encrypt Data and Store data: The text data which is extracted from the transaction is stored in an encrypted format. The encryption of the file is performed with AES256 and

stored on the database. The encrypted data can only be accessed by Transaction Hash and Public Key. Whenever a user requests PII data, the database is queried to retrieve results.

Query Blockchain: The blockchain can be queried to view the results of transactions. The specific transactions can be retrieved using the transaction hash or block number. The query uses the transaction id or transaction hash to retrieve the transaction receipt.

The queries use web3 libraries to retrieve details on blockchain. The examples of the query statements:

web3.eth.getBlock(blocknumber,console.log);

web3.eth.getTransactionFromBlock(Transaction Hash);

Chapter 4: Experiments and Results

Experiments were performed on a single Linux server – RedHat Version - CentOS Linux release 7.6.1810 (Core), Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz. Components used are – Ganache CLI v6.9.0, Truffle v5.1.12, Ethereum web3, Solidity v0.5.16, Node v11.13.0, Web3.js v1.2.1. The technologies used are – python version 3.6.7, Nodejs version 6.16, Go version 1.12.1, Docker version 18.09, NPM version 6.13.7, OpenSSL 1.0.2k-fips.

Privacy policies are consumed in text format. The word tokenization, part of speech tagging, and deontic rules extraction are performed using the NLTK python library. A privacy policy ontology is created which is used for verification of transactions on the blockchain network. Relations are added or removed depending on the requirements of the use case. The transactions executed on the network should be policy compliant. The PII data is encrypted using a private key, AES is used for encryption of data and the file is stored on the database. The Ethereum blockchain network is then queried to retrieve results as per the user requirements. If the transaction fails to transfer data, one of the possibilities for failure can be the user is not registered on the network which implies the user is restricted for any transactions on the network. In our system, all the consumers and the providers have to be registered on the blockchain network for initiating transactions. As in blockchain, the data is always updated as per the consensus algorithm and the data values are matched according to the ledger value, keeping the data off-chain benefits in tracking the previous values.

All the transactions are executed over the blockchain network, the data will always be tracked and the end-users will be aware of where the data is being shared and to how many

providers it is being transferred. With this setup of a decentralized platform, transferring assets, registering users, controlling data related transactions in compliance with regulations is much simpler and easier.

Before coming down to a conclusion of using Ethereum Blockchain we did some experimentation with Hyperledger Fabric, I have added in line the experiment screenshots and some description related to it. We had taken an example where we had an rdf file which was used to extract details related to a user. The rdf file parsed and the results are extracted and stored in a text file. The transactions included transfer of points and we wanted to make sure that all the transactions are compliant to the rules and stored in the process. Hence , hyperledger fabric evm was used to track and store the transactions. With the help of decentralized application, the users can interact with the smart contract by submitting a query to the contract and retrieving results. The smart contracts are written in Solidity language and interaction is accomplished using web3 library. Steps involved in the initialization and transaction on hyperledger fabric are mentioned below:

1. Deploy the Hyperledger Fabric EVM on the local network
2. Deploy smart contract
3. Start the DApp instance to interact with Fabric
4. Interact with contract using Web3 library
5. Retrieve friends list
6. Register Tim as a provider on the chaincode
7. Add a couple of Tim's friends as the consumers(Register Tim's friends on the chaincode)
8. Do the transfer of points from Tim to his friends

9. If the person exists on the list the transaction is successful, otherwise transaction fails

Components used in the experiment:

1. Hyperledger Fabric 1.4, Hyperledger Fabric EVM Chaincode
2. Ethereum web3.js
3. Solidity
4. Foaf.rdf

Screenshots related to Hyperledger Fabric experiment:

Docker containers running for the Hyperledger Fabric Chaincode:

```
[amahind1@knacc2 ~] docker ps
CONTAINER ID        IMAGE                                                                                                                   COMMAND
        PORTS                     NAMES
787eadaf64b5        dev-peer0.org1.example.com-evmcc-0-aaf37d90d8d396f179cf05229debb5f1c9d24e5737db406a2af8112b9f7b7c75    "chaincode -peer.add…"
                                  dev-peer0.org1.example.com-evmcc-0
d8a222ae274b        dev-peer1.org2.example.com-mycc-1.0-26c2ef32838554aac4f7ad6f100aca865e87959c9a126e86d764c8d01f8346ab   "chaincode -peer.add…"
                                  dev-peer1.org2.example.com-mycc-1.0
7113317b86f6        dev-peer0.org1.example.com-mycc-1.0-384f11f484b9302df90b453200cfb25174305fce8f53f4e94d45ee3b6cab0ce9   "chaincode -peer.add…"
                                  dev-peer0.org1.example.com-mycc-1.0
903e0edaaf73        dev-peer0.org2.example.com-mycc-1.0-15b571b3ce849066b7ec74497da3b27e54e0df1345daff3951b94245ce09c42b   "chaincode -peer.add…"
                                  dev-peer0.org2.example.com-mycc-1.0
073940383ab2        hyperledger/fabric-tools:latest                                                                         "/bin/bash"
                                  cli
3fd50f720378        hyperledger/fabric-peer:latest                                                                          "peer node start"
        0.0.0.0:10051->10051/tcp  peer1.org2.example.com
6ac3d1988b47        hyperledger/fabric-peer:latest                                                                          "peer node start"
        0.0.0.0:7051->7051/tcp    peer0.org1.example.com
040ee76019b0        hyperledger/fabric-peer:latest                                                                          "peer node start"
        0.0.0.0:8051->8051/tcp    peer1.org1.example.com
ff13a1ac3961        hyperledger/fabric-peer:latest                                                                          "peer node start"
        0.0.0.0:9051->9051/tcp    peer0.org2.example.com
1ae98e1fa24d        hyperledger/fabric-orderer:latest                                                                       "orderer"
        0.0.0.0:7050->7050/tcp    orderer.example.com
```

Figure 4: Peer node Docker Images

When the network is up, the smart contract is deployed which is shown in Figure 6. The advantage of using a smart contract is, once the contract is deployed by a single instance on the network it is deployed on the ledger and updates all the Dapps on the network, hence all peer nodes will have an updated version of the smart contract. The below screenshot will

show the details of the deployed smart contract. The important parameters to be noted are the transaction hash, blockhash and contractAddress.

DApps are running on the servers now on the port 5000 and 5001:

```
bash-4.2$ ps -ef | grep fab3
amahind1  85577  85155  0 06:43 pts/0    00:00:00 grep fab3
amahind1 141162      1  0 Jul13 ?        00:14:23 ./fab3
amahind1 142851      1  0 Jul13 ?        00:15:40 ./fab3
```

```
>
> web3.eth.accounts
[ '0xf1d788308b743c253e7ec4d948348251329e5c1b' ]
>
```

Figure 5: Account number of the provider

Once the contract is deployed, we register the provider on the network. Figure 4 shows the registration command used from a peer node. Figure 7 also covers the registration and encryption flow.

```
> web3.eth.getTransactionReceipt(deployedContract.transactionHash)
{ transactionHash: '0xb9eb28ba85badbe6e838fe03314421565eba22904fc410efa5b89f80e1f5e088',
  transactionIndex: 0,
  blockHash: '0xda3a14fae05f10c0f53ea6d383787581098c8934cdd38ffb5fd7f71a5b9502bc',
  blockNumber: 14,
  contractAddress: '2fe9b85e3a8819eff3faa66a6fd3de036bb48006',
  gasUsed: 0,
  cumulativeGasUsed: 0,
  to: '',
  logs: null,
  status: '0x1' }
```

Figure 6: Contract Information

The consumer is now registered on the network in the similar pattern but with personal details like first name, last name and email id. Figure 7 shows the registration of consumer on a peer node.

```
> myContract.registerPartner("Tim", { from: web3.eth.accounts[0] })
'a47462ea4750b21ae1f87e609e7164f83ef12bd807b78ee04b2e9983bd73c561'
> myContract.partners(web3.eth.accounts[0])
[ '0xf1d788308b743c253e7ec4d948348251329e5c1b', 'Tim', true ]
> myContract.partnersInfo(0)
[ '0xf1d788308b743c253e7ec4d948348251329e5c1b', 'Tim', true ]
> myContract.partnersInfoLength()
{ [String: '1'] s: 1, e: 0, c: [ 1 ] }
```

Figure 7: Registration information

```
> web3.eth.accounts
[ '0x5e240b8da64137a5beae32bcc0d2a1ad8b669567' ]
>
```

Figure 8: Account 2 on different peer node and his account number

```
> myContract.registerMember("Joe", "Lambda", "joe@lambda.com", { from: web3.eth.accounts[0] })
'9e72de0472e2c13b88add73e1b030947936194c72a4b10dfe01851aa3cb74ac5'
```

Figure 9: Consumer 2 is registered on the chaincode

```
> myContract.members(web3.eth.accounts[0])
[ '0x5e240b8da64137a5beae32bcc0d2a1ad8b669567',
  'Joe',
  'Lambda',
  'joe@lambda.com',
  { [String: '0'] s: 1, e: 0, c: [ 0 ] },
  true ]
```

Figure 10: Transaction Information

```
> myContract.earnPoints(200, '0xf1d788308b743c253e7ec4d948348251329e5c1b', { from: web3.eth.accounts[0] })
'c30026cb467866e55ffc9181236693c584daf4b0d7929e6ce71ecbf243301c4e'
> myContract.members(web3.eth.accounts[0])
[ '0x5e240b8da64137a5beae32bcc0d2a1ad8b669567',
  'Joe',
  'Lambda',
  'joe@lambda.com',
  { [String: '200'] s: 1, e: 2, c: [ 200 ] },
  true ]
```

Figure 11: Transferred points from Consumer 1 to Consumer 2

```
> myContract.transactionsInfo(0)
[ { [String: '200'] s: 1, e: 2, c: [ 200 ] },
  { [String: '0'] s: 1, e: 0, c: [ 0 ] },
  '0x5e240b8da64137a5beae32bcc0d2a1ad8b669567',
  '0xf1d788308b743c253e7ec4d948348251329e5c1b' ]
```

Figure 12: Transaction Information

```
> myContract.members(web3.eth.accounts[0])
[ '0x5e240b8da64137a5beae32bcc0d2a1ad8b669567',
  'Joe',
  'Lambda',
  'joe@lambda.com',
  { [String: '150'] s: 1, e: 2, c: [ 150 ] },
  true ]
```

Figure 13: Consumer 2 transferred 50 tokens to Consumer 1

The details of the consumer are extracted in a json format and then encrypted using AES encryption.

Actual Data in JSON File of PII :
0x5e240b8da64137a5beae32bcc0d2a1ad8b669567, Joe, Lambda, joe@lambda.com, 200,true
Encrypted Data After Encryption with AES:
AES^B^@^@^[CREATED_BY^@pyAesCrypt0.4.3^@<80>^@^@^@^@^ @^ @^ @^ @^ @^ ?õ^Y52'%AéJNÁi]Éél:K<93>x¹©Ó&5:^K}¢Mø<86><8a>è·<86>B½ár?j<9c >¶,ÏîZ<9d><99>E·<8a><9c>{ñ<84>EÖ^Hk3ÄìR<8d>ÿ^SISûct µ!kzÊðÅÓ<8 b>¹³æM^?Z<8f>-<8d>ú<80>Û^F^N»5]í`x^U<ö<86>Ù-Ç^R©Öf<94>^H<9 5>ÎV¯út§^F"D\Ë^Vºv±°^QÎÁz¼\<81>-wã|]¿à<81>À^RNÌ^\ëc´<94><9b>^ Q¯.ß§óRõ<8c>ü}<97><98>§x´Z^Mv,#Ö?¶+CÈ^FO:^WYíJÊ/<84>ÍQ^Eª<8f ><8a>Â<82><8f>i¶<92>W	

Figure 14: Encryption of PII Data in Json Format

To set up the Ethereum environment we created a blockchain network using ganache-cli. Ganache-cli creates 10 accounts which were then assigned to our dedicated consumers and providers. Figure 15 shows the accounts created by ganache-cli on the Ethereum network. All the members on the network are required to be registered on the network using the account id and private key. When the network is up, the smart contract is deployed which is shown in Figure 16.

The advantage of using a smart contract is, once the contract is deployed by a single instance on the network it is deployed on the ledger and can be accessed by all the members on the network, hence all members will have an updated version of the smart contract. The smart contract is immutable and hence it confirms security over the network. The important parameters to be noted are the transaction hash, block number, and contract address. Once the contract is deployed, we can test the contract methods using the test feature of truffle. Truffle is the Ethereum framework to create, compile, and deploy solidity contracts. Truffle automation tests use the Mocha framework to test solidity contracts. Mocha is a feature-rich JavaScript test framework. Figure 4 shows the execution of the DataComplianceContract which is executed on our network once the solidity contract is deployed. The test execution shows us the time(in milliseconds) required to execute each method in the contract.

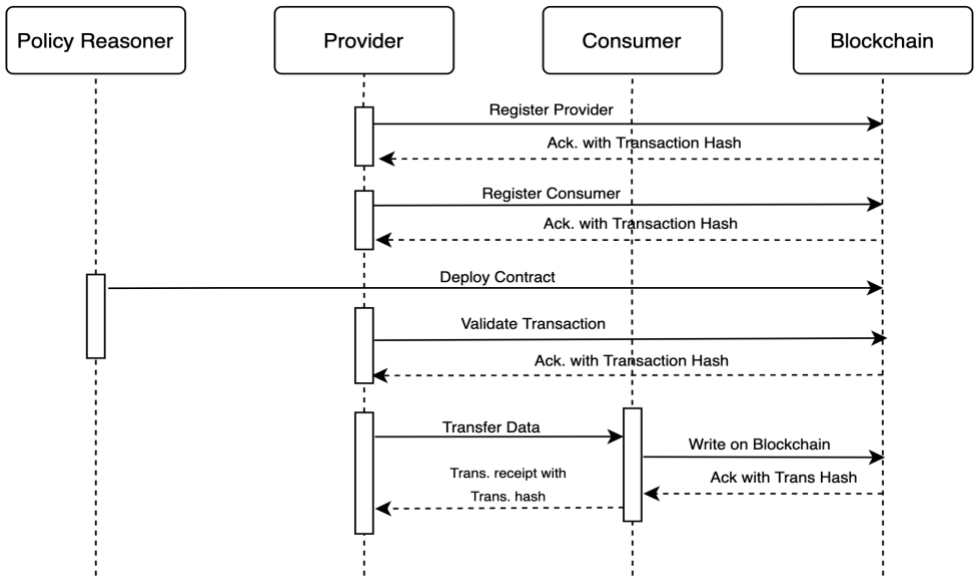

Figure 15: Workflow of Registration

```
bash-3.2$ ganache-cli --networkId 1 --deterministic "pyramid creek cost left seminar west spread solid milk invest
 suffer rough" --accounts 5
Ganache CLI v6.9.0 (ganache-core: 2.10.1)

Available Accounts
==================
(0) 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (100 ETH)
(1) 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0 (100 ETH)
(2) 0x22d491Bde2303f2f43325b2108D26f1eAbA1e32b (100 ETH)
(3) 0xE11BA2b4D45Eaed5996Cd0823791E0C93114882d (100 ETH)
(4) 0xd03ea8624C8C5987235048901fB614fDcA89b117 (100 ETH)

Private Keys
==================
(0) 0x4f3edf983ac636a65a842ce7c78d9aa706d3b113bce9c46f30d7d21715b23b1d
(1) 0x6cbed15c793ce57650b9877cf6fa156fbef513c4e6134f022a85b1ffdd59b2a1
(2) 0x6370fd033278c143179d81c5526140625662b8daa446c22ee2d73db3707e620c
(3) 0x646f1ce2fdad0e6deeeb5c7e8e5543bdde65e86029e2fd9fc169899c440a7913
(4) 0xadd53f9a7e588d003326d1cbf9e4a43c061aadd9bc938c843a79e7b4fd2ad743

HD Wallet
==================
Mnemonic:      myth like bonus scare over problem client lizard pioneer submit female collect
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975
```

Figure 16: Accounts created by Ganache-cli

```
eth_sendTransaction

  Transaction: 0x5f97778b2892a57d2386ee80a9d2778e9317770fd3c60ff2705e40bfaf7d6724
  Contract created: 0xcfeb869f69431e42cdb54a4f4f105c19c080a601
  Gas usage: 2778544
  Block Number: 3
  Block Time: Wed Mar 11 2020 18:36:57 GMT-0400 (Eastern Daylight Time)
```

Figure 17: Results of the Smart Contract Deployment

```
(env) bash-3.2$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: DataComplianceContract
    ✓ Register consumer (69ms)
    ✓ Register provider (49ms)
    ✓ Transfer Data From Provider A to Provider B (52ms)
    ✓ Transfer Data From Consumer to Provider A (50ms)


  4 passing (319ms)
```

Figure 18: Deployed Contract Testing and Network

The DataComplianceContract is the smart contract deployed on the network and automatically validates a condition and determines if data is allowed to be transferred from Consumer to Provider. Figure 18 shows the Transaction Receipt of the contract deployed on the network.

```
(env) bash-3.2$ python register_users_truffle.py
Press Enter to Register Users on Network
Using network 'development'.

Getting deployed version of DataComplianceContract...
Registering Consumer on the network
Transaction: 0x16ce4eae0a13c6059f051172c889ea3c4e2f63cdf528c0710e2150d696a11d50
Finished!
Using network 'development'.

Getting deployed version of DataComplianceContract...
Registering Provider on the network
Transaction: 0xc5babf99dde004c2c463e904443223d86d2695037b2a779bc2419b7f4d67a8dd
Finished!
```

Figure 19: Registering Members on the Network

```
Tranfer Data from Microsoft to LinkedIn
Using network 'development'.

Getting deployed version of DataComplianceContract...
Transfering user data from Microsoft to LinkedIn
Transaction: 0xbf57ad652b39bcb4706a7221b5fc67531e387c81c6f78572aa311d0d3cb06ddb
Transaction Successful!
```

Figure 20: Transfering Data from Microsoft to Linkedin

```
Transaction: 0xbf57ad652b39bcb4706a7221b5fc67531e387c81c6f78572aa311d0d3cb06ddb
Gas usage: 102037
Block Number: 9
Block Time: Wed Mar 11 2020 18:51:01 GMT-0400 (Eastern Daylight Time)
```

Figure 21: Transaction Receipt for Successful Data Transfer

For every transaction on the network, a Sparql query is executed on the privacy ontology to check for permission. Based on the checks defined in the smart contract, the transaction is executed. In the case of Figure 19 and Figure 20, Microsoft is transferring PII data to LinkedIn. As the regulatory compliance allows Microsoft to transfer data to LinkedIn in the awareness of the consumer, the transaction is successful ensuring Article 24 i.e. Processing is performed in accordance with regulation.

```
Using network 'development'.

Getting deployed version of DataComplianceContract...
Transfering user data from Microsoft to Google
Transaction Failed
Provider is not allowed to transfer Data
```

Figure 22: Transfering Data from Microsoft to Google

```
Transaction: 0xdfb311196d00613c5181d8f22b4cdda472a079f807c806fc00e4e09883d32f91
Gas usage: 25404
Block Number: 11
Block Time: Wed Mar 11 2020 18:53:12 GMT-0400 (Eastern Daylight Time)
Runtime Error: revert
Revert reason: Partner is not allowed to transfer assets
```

Figure 23: Transaction Receipt for Failed Data Transfer

When it comes to Microsoft transferring PII data to Google, the transaction is denied in Fig 21 and 22. The regulatory compliance of Microsoft does not allow the transfer of data to any other company other than its own companies. When the transaction is initiated, the Sparql query checks with the policy ontology for permission, ensuring privacy based on the GDPR article 24 which says it is the responsibility of the controller to process data in accordance with the regulation. All the transactions are recorded on the blockchain network which ensures article 30. Gas utilization for every transaction function is in shown in figure 23.

Less execution time

Transaction	Gas used
Contract 1 creation - Datacompliancecontract	2794411
Contract 2 creation - Migrations	172059
Register consumer 1 on the network	134090
Register provider 1 on the network	175161
Register consumer 2 on the network	160173
Register provider 2 on the network	134138
Transfer Data from Microsoft to LinkedIn	132037
Transfer Data from Microsoft to Google	25404
Unlock Provider	29134
Unlock Consumer	24604

More computation

Figure 24: Gas utilized per transaction

Chapter 5: Conclusion

Many governing bodies are releasing new data protection and security laws every year to ensure data privacy and security. These regulations are in text format and it needs a high amount of human time and effort to get an assurance of compliance over the network. We believe that the semantically rich, machine-readable knowledge graph or ontology manages to capture most of the regulations which assist in automating an organization's data compliance process. The private data of a consumer is private and should not be shared without the consent of the consumer. The data should be available by the end-user until the time the provider maintains the data. Our architecture uses the blockchain platform to support maintain this scenario where the consumer will always have access to their data and track the lifecycle of his data. As the blockchain platform is decentralized, delivering regulatory decisions about collecting, storing, and sharing of personal data is much easier.

Our proposed system addresses the predicaments of data tracking or PII tracking plus the data stored in an encrypted format ensures security. This system was only implemented using Ethereum as blockchain technology, but we plan on exploring other blockchain frameworks like Hyperledger Iroha, Hyperledger Indy, and tools like Hyperledger Composer to create a more extensive network and better processing capabilities. We also plan on extending this project which can help automate the whole process without any human intervention. As blockchain is an emerging technology and the process of storing and retrieval of data is much different than the experienced enterprise databases, complying with all regulations is challenging. Huge development is in progress in the blockchain industry to integrate the GDPR, similarly, we plan to cover a few GDPR

issues like ‘Privacy by Design’, ‘Processing’, ‘Right to Access’, ‘Third Countries’[29] in our future work.

Appendices

Smart Contract – Data Compliance Contract File:

```
pragma solidity 0.4.24;

contract DataComplianceContract {


// model a member

struct Member {

address memberAddress;

string firstName;

string lastName;

string email;

uint points;

bool islock;

bool isRegistered;

}


// model a partner

struct Partner {

address partnerAddress;

string name;

bool islock;

bool isRegistered;

}
```

```
// model points transaction
enum TransactionType { Earned, Redeemed }
struct PointsTransaction {
uint points;
TransactionType transactionType;
address memberAddress;
address partnerAddress;
}
address public owner;
constructor() public {
owner = msg.sender;
}

//members and partners on the network mapped with their address
mapping(address => Member) public members;
mapping(address => Partner) public partners;
//public transactions and partners information
Partner[] public partnersInfo;
PointsTransaction[] public transactionsInfo;


function renderHelloWorld () public pure returns (string) {
```

```
return 'helloWorld';
}

//register sender as member
function registerConsumer (string _firstName, string _lastName, string _email, bool
_islock, address memberAddress) public {
//check msg.sender in existing members
require(!members[memberAddress].isRegistered, "Account already registered as
Consumer");

//check msg.sender in existing partners
require(!partners[memberAddress].isRegistered, "Account already registered as
Provider");

//add member account
members[memberAddress] = Member(memberAddress, _firstName, _lastName, _email,
0, _islock, true);

}
//register sender as partner
function registerProvider (string _name, bool _islock, address partnerAddress) public {
//check msg.sender in existing members
```

```
require(!members[partnerAddress].isRegistered, "Account already registered as
Consumer");

//check msg.sender in existing partners
require(!partners[partnerAddress].isRegistered, "Account already registered as
Provider");

//add partner account
partners[partnerAddress] = Partner(partnerAddress, _name, _islock, true);

//add partners info to be shared with members
partnersInfo.push(Partner(partnerAddress, _name, _islock, true));

}

//update member with points earned
function transferAssetstoConsumer (uint _points, address _partnerAddress, address
_memberAddress ) public {
// only member can call
require(members[_memberAddress].isRegistered, "Sender not registered as Consumer");

// verify partner address
require(partners[_partnerAddress].isRegistered, "Provider address not found");
```

```
//islock Status added by Abhishek
require(!members[_memberAddress].islock, "Member is not allowed to transfer data");

// update member account
members[_memberAddress].points = members[_memberAddress].points + _points;

// add transction
transactionsInfo.push(PointsTransaction({
points: _points,
transactionType: TransactionType.Earned,
memberAddress: members[_memberAddress].memberAddress,
partnerAddress: _partnerAddress
}));

}

//update member with points used
function transferAssetstoProvider (uint _points, address _partnerAddress, address
_memberAddress) public {
// only member can call
require(members[_memberAddress].isRegistered, "Sender not registered as Consumer");
```

```
// verify partner address
require(partners[_partnerAddress].isRegistered, "Provider address not found");

//require(_partnerAddress != address(accounts[6]), "Member not allowed to transfer
Assets");
//require(_partnerAddress !=
address(0xeBdA127CffeA9C1461e3EF3ab24156903C807191), "Member not allowed to
transfer Assets");
require(!partners[_partnerAddress].islock, "Provider is not allowed to transfer data");

// verify enough points for member
require(members[_memberAddress].points >= _points, "Insufficient points");

// update member account
members[_memberAddress].points = members[_memberAddress].points - _points;

// add transction
transactionsInfo.push(PointsTransaction({
points: _points,
transactionType: TransactionType.Redeemed,
memberAddress: members[_memberAddress].memberAddress,
partnerAddress: _partnerAddress
}));
```

```
}

//get length of transactionsInfo array
function transactionsInfoLength() public view returns(uint256) {
return transactionsInfo.length;
}

//get length of partnersInfo array
function partnersInfoLength() public view returns(uint256) {
return partnersInfo.length;
}

//Give access to user to transact Assets
function unlockProvider (bool _islock, address _partnerAddress) public {
//check msg.sender in existing members
require(partners[_partnerAddress].isRegistered, "Provider is not registered");


//update partner account
//partners[msg.sender] = Partner(msg.sender, _name, _islock, true);
partners[_partnerAddress].islock = _islock;

}
```

```
//Give access to user to transact Assets
function unlockConsumer (bool _islock, address _memberAddress) public {

//check msg.sender is registered
require(members[_memberAddress].isRegistered, "Consumer not registered");

//update members account
members[_memberAddress].islock = _islock;

}

function showConsumerDetails(address _memberAddress) public view returns(address,
string, string, string, uint256) {
require(members[_memberAddress].isRegistered, "Consumer not registered");
return
(members[_memberAddress].memberAddress,members[_memberAddress].firstName,
members[_memberAddress].lastName, members[_memberAddress].email,
members[_memberAddress].points);
//return (members[_memberAddress].memberAddress);
}
```

```
function showProviderDetails(address _partnerAddress) public view returns(address,
string) {
require(partners[_partnerAddress].isRegistered, "Partner is not registered");
return (partners[_partnerAddress].partnerAddress,partners[_partnerAddress].name);



}

function deleteProviderDetails(address _partnerAddress) public {
require(partners[_partnerAddress].isRegistered, "Provider is not registered");
partners[_partnerAddress].name = '';



}

function deleteConsumerDetails(address _memberAddress) public {
require(members[_memberAddress].isRegistered, "Consumer is not registered");
members[_memberAddress].firstName = '';
members[_memberAddress].lastName = '';
members[_memberAddress].email = '';

}
}
```

Download privacy policy files from the web:

```
#!/usr/bin/python3


import wget

import time


if __name__ == '__main__':

url =

'https://raw.githubusercontent.com/abhimaaahi/GDPR_owlfile/master/GDPR_PCIDSS.o

wl?token=AHYIFVSUIOOSVPPK26XHJFC6JVMGA'

print("Downloading the GDPR OWL file..")

time.sleep(2)

wget.download(url,'/Users/abhimaahi/thesis_code/Policy_Repository')

print('\n')

url1 =

'https://raw.githubusercontent.com/abhimaaahi/GDPR_owlfile/master/Microsoft.txt?toke

n=AHYIFVU653G2YHT5MCINWYK6JVMWM'

print('Downloading the privacy policy of Microsoft Company A')

time.sleep(2)

wget.download(url1,'/Users/abhimaahi/thesis_code/Policy_Repository')

print('\n')
```

```
url2 =
'https://raw.githubusercontent.com/abhimaaahi/GDPR_owlfile/master/Microsoft1.txt?tok
en=AHYIFVRUUPIV5O5DE3YEHR26JVM2M'
print('Downloading the privacy policy of Microsoft Company B')
time.sleep(2)
wget.download(url2,'/Users/abhimaahi/thesis_code/Policy_Repository')
url3 =
'https://raw.githubusercontent.com/abhimaaahi/GDPR_owlfile/master/aws_gdpr.txt?toke
n=AHYIFVQJRTMB3DIAYVFRBNC6K43TC'
wget.download(url3,'/Users/abhimaahi/thesis_code/Policy_Repository')
print('\n')
```

Parse the GDPR ontology policy document:

```
#!/usr/bin/python3
import rdflib
import re
import os
import time

def processGDPRGraph():
g = rdflib.Graph()
```

```
# Download the Policy file and read it on the local repository
g.load("Policy_Repository/GDPR_PCIDSS.owl")

# Read the owl file directly from the Git repository
#
g.load("https://raw.githubusercontent.com/abhimaaahi/GDPR_owlfile/master/GDPR_PCI
DSS.owl?token=AHYIFVXPH54AEBKMTEWLPCK6K2TIG")
# Link to gdpr repository from cloud
print("")
gdpr_link = 'https://www.csee.umbc.edu/~kjoshi1/regulations/GDPR_PCIDSS.owl'
print("Loading GDPR Ontology from Link -", gdpr_link)
print("")
#g.load("https://www.csee.umbc.edu/~kjoshi1/regulations/GDPR_PCIDSS.owl")
#g.load(gdpr_link)
#return gdpr_link
# the QueryProcessor
print("Extracting Articles only related to Consumer and Provider")
print("==========================================")
time.sleep(2)

for row in g.query(
#'select ?country ?artist ?email where { ?x cd:country ?country .  ?x cd:artist ?artist . ?x
cd:email ?email}'):
```

```
'SELECT ?subject ?object WHERE { ?subject rdfs:subClassOf ?object }'):
results = row.subject
results1 = row.object
#if (results ==
'http://www.semanticweb.org/lavi/ontologies/2018/2/GDPR_PCIDSS#GDPR_Articles'):
results = results.split('#')
results1 = results1.split('#')
articles = results[1]
#print(articles)
#print(results[1], " " ,results1[1])

# Print Consumer Obligations
if (results1[1] == 'Consumer_Obligations'):
print(results[1], " ", results1[1])
# Print Provider Obligations
if (results1[1] == 'Provider_Obligations'):
print(results[1], " " ,results1[1])

# Print GDPR Articles
if (results1[1] == 'GDPR_Articles'):
#if (results[1] == 'Art.24' or results[1] == 'Art.25' or results[1] == 'Art.26' or results[1] ==
'Art.27' or results[1] == 'Art.28' or results[1] == 'Art.29' or results[1] == 'Art.30'):
#print(results[1], " ", results1[1])
```

```
#print(results)

#articles = results[1]

#print(articles)

pattern = re.compile(r'Art.2\d')

pattern1 = re.compile(r'Art.3\d')

if (pattern.search(articles)):

print(results[1], " ", results1[1])

if (pattern1.search(articles)):

print(results[1], " ", results1[1])

if (results1[1] == 'Art.24' or results1[1] == 'Art.25' or results1[1] == 'Art.30' or

results1[1] == 'Art.26' or results1[1] == 'Art.37' or results1[1] == 'Art.32'):

print(results[1], " ", results1[1])


# Remove a to print articles on console

processGDPRGraph()

#a = processGDPRGraph()

#print(a)
```

Python file to check policy documents for modal verbs and create smart contract:

```
import codecs

import re

import string

import math

import os

import sys

import time

import subprocess


#Validation for User inputs

if len(sys.argv) < 3:

print('Warning : Please enter two documents for policy check')

sys.exit(1)

# if len(sys.argv[2]) < 2:

#  print('Please enter document 2 name')


#Enter filename on the cmdline after the python file

filename = sys.argv[1]

filename1 = sys.argv[2]


def term_frequency(term, tokenized_document):
```

```
return tokenized_document.count(term)


#os.system('python gdpr_Articles.py')
input("Press enter to extract modal verbs ..")


# Execute POS_TAG file to detect Modal Verbs in the Policy Documents
#command = "python pos_tag.py Policy_Repository/Microsoft.txt"
#We need the below statements
#command = "python pos_tag.py '%s'" %(filename)
#os.system(command)


#subprocess.run(python pos_tag.py "filename")
#print(command)
from pos_tag import isallowed, isdenied


print("==========================")
input("Press enter to print the articles related to consumer and producer")
os.system('python gdpr_Articles.py')


print('------------------------------');
print('The contracts mentioned below will be deployed on the blockchain network');
print('------------------------------');
```

```
tokens = []

tokens1 = []

count = 0

printflag = 0

printflag1 = 0

printflag2 = 0

printflag3 = 0



with open(filename, 'rb') as f:

lines = [l.decode('utf8', 'ignore') for l in f.readlines()]


with open(filename1, 'rb') as f1:

lines1 = [l.decode('utf8', 'ignore') for l in f1.readlines()]


subs = 'GDPR'


# Deontic Logics

subs1 = 'Permission'

#subs2 = 'Obligation'

#subs3 = 'Dispensation'

subs4 = 'Prohibition'
```

```
for line in lines:

tokens = line.split('.')

#print(tokens)

#pattern = re.compile(r'GDPR')

res = list(filter(lambda x: subs in x, tokens))

res1 = list(filter(lambda y: subs1 in y, tokens))

#res2 = list(filter(lambda z: subs2 in z, tokens))

#res3 = list(filter(lambda a: subs3 in a, tokens))

res4 = list(filter(lambda b: subs4 in b, tokens))

#res1 = str(res)

count = len(res)

#print(count)


if (count == 1 and printflag==0):

print("============== Checking Policy Document 1 ==========")

time.sleep(2)

print ("GDPR is mentioned in the policy1, please deploy SmartContract ")

printflag = 1


if (isallowed == 1 and printflag1 == 0):

print ("Permission Present, please deploy SmartContract and Consumer A Allowed to

transfer Data ")

printflag1 = 1
```

```
if (isdenied == 1 and printflag2 == 0):
print("Prohibition Present, please deploy SmartContract 3 \n")
printflag2 = 1
count = 0
printflag = 0
printflag1 = 0
printflag2 = 0
printflag3 = 0

for line in lines1:
tokens1 = line.split('.')

res = list(filter(lambda x: subs in x, tokens1))
res1 = list(filter(lambda y: subs1 in y, tokens1))
#res2 = list(filter(lambda z: subs2 in z, tokens1))
#res3 = list(filter(lambda a: subs3 in a, tokens1))
res4 = list(filter(lambda b: subs4 in b, tokens1))
#res1 = str(res)
count = len(res)
#print(count)

if (count == 1 and printflag==0):
print("=============== Checking Policy Document 2 ============")
```

```
time.sleep(2)
print ("GDPR is mentioned in the policy2, please deploy SmartContract ")
printflag = 1

if (isallowed == 1 and printflag1 == 0):
print ("Permission Present, please deploy SmartContract and Consumer A Allowed to
transfer Data ")
printflag1 = 1

# if (isdenied == 1 and printflag2 == 0):
#  print("Prohibition Present, please deploy SmartContract 3 \n")
# printflag2 = 1

# Deploying smartcontracts on all the networks where partners are present
input("Press Enter to Deploy SmartContract on the Blokchain Network ")
os.chdir('/Users/abhimaahi/loyalty-points-evm-fabric/truffle-env')
#os.system('truffle compile')
os.system('truffle migrate --network development')
time.sleep(2)
os.system('truffle migrate --network development_live')

# Execute python scripts to register users on network
os.system('python register_users_truffle.py')
```

Python file to extract modal verbs from policy documents:

```
import sys

import re

import nltk

import os

import json

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

import time


#Validation for User inputs

if len(sys.argv) < 2:

print('Warning : Please enter document for pos_tagging and modal verbs finding')

sys.exit(1)

# if len(sys.argv[2]) < 2:

#  print('Please enter document 2 name')


#Enter filename on the cmdline after the python file

filename = sys.argv[1]

#filename1 = sys.argv[2]
```

```
with open(filename, 'rb') as f:

lines = [l.decode('utf8', 'ignore') for l in f.readlines()]


def processContent():

try:

modal_verbs = []

k = 0

isallowed = 0

isdenied = 0

for item in lines:

tokenized = nltk.word_tokenize(item)

tokenized1 = [w for w in tokenized if not w in stop_words]

#print(tokenized1)

tagged = nltk.pos_tag(tokenized1)

#print(tagged)


#chunkGram = r"""Chunk: {<RB>?<VB>*<NN>*<MD>}"""

#chunkParser = nltk.RegexpParser(chunkGram)


#chunked = chunkParser.parse(tagged)


for i in tagged:

if i[1].startswith('MD'):
```

```
#print(i)
with open('jsons.txt','a') as json_file:
json.dump(i,json_file)
k = k + 1
j = tagged.index(i)
# Permission
if (i[0] == 'may' or i[0] == 'can' or i[0] == 'could'):
isallowed = 1
# Prohibition
if (i[0] == 'would' or i[0] == 'WILL' or i[0] == 'will' or i[0] == 'shall' or i[0] == 'should' or
i[0] == 'must'):
isdenied = 1
print(tagged[j-1],i,tagged[j+1])
#print(tagged[i])
#for i in tagged:
# if i[0] == 'may':
#  break
#print('Permission is given')
print("Total number of modal verbs present in the document", k)
#print(isallowed)
#print(isdenied)
return isallowed, isdenied
except Exception as e:
```

```
print(str(e))


print("==============================================")
print("Modal Verbs Present in the Policy Document which will be used to detect
Permission or Obligation")
print("==============================================")
time.sleep(2)
isallowed, isdenied = processContent()
print(isallowed)
print(isdenied)
# Copy policy based on the result
if (isdenied == 1):
os.system('cp Policy_Repository/privacy_policy_isdenied.owl privacy_policy.owl')
elif (isallowed == 1):
os.system('cp Policy_Repository/privacy_policy_isallowed.owl privacy_policy.owl')
#os.system('python gdpr_Articles.py')
```

Python file to register all the stakeholders on Truffle:

```
import os


# Deploying smartcontracts on all the networks where partners are present
input("Press Enter to Register Users on Network ")
```

```
os.chdir('/Users/abhimaahi/loyalty-points-evm-fabric/truffle-env')
#os.system('truffle compile')

# Register Consumer1 on the network
os.system('truffle exec RegisterConsumer.js  --network development')
# Register Provider1 on the network)
os.system('truffle exec RegisterProvider.js  --network development')
#Register Provider2 on the network
os.system('truffle exec RegisterProvider_1.js  --network development')
#Register Consumer2 on the network
os.system('truffle exec RegisterConsumer_1.js  --network development')
```

Python file to transfer data from one company to another using Truffle:

```
#from pos_tag import isallowed, isdenied
import os
import time
import sys
import rdflib
import re

def check_policy_for_permission():
```

```
list1 = []
isallowed = 0
isdenied = 0
g = rdflib.Graph()
g.load("privacy_policy.owl")
for o in g.objects():
list1.append(o)

subs = 'isallowed'
res = list(filter(lambda x: subs in x, list1))
if len(res) > 0:
print("Provider is allowed to transfer Data")
isallowed = 1
subs1 = 'isdenied'
res1 = list(filter(lambda x: subs1 in x, list1))
if len(res1) > 0:
print("Provider is not allowed to transfer Data")
isdenied = 1
#use a sparql query here to check if provider allowed to transfer data or no
if (isallowed == 1):
print("Tranfer Data from Microsoft to LinkedIn")
time.sleep(2)
os.chdir('/Users/abhimaahi/loyalty-points-evm-fabric/truffle-env')
```

```
os.system('truffle exec transferAssets.js  --network development')
elif (isdenied == 1):
print("Tranfer Data from Microsoft to Google")
time.sleep(2)
os.chdir('/Users/abhimaahi/loyalty-points-evm-fabric/truffle-env')
os.system('truffle exec lockprovider.js  --network development')
os.system('truffle exec transferAssets_failed.js --network development')
def clear_userDetails():
print ("Erasing User Details")
time.sleep(2)
os.chdir('/Users/abhimaahi/loyalty-points-evm-fabric/truffle-env')
os.system('truffle exec eraseUserDetails.js  --network development')
check_policy_for_permission()
```

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Consulted, 1(2012):28, 2008.

[2] EUROPEAN COMMISSION. Commission proposes a comprehensive reform of data protection rules to increase users' control of their data and to cut costs for businesses. 2012.

[3] Karuna Pande Joshi et al., "Automating Cloud Services Lifecycle through Semantic technologies", Article, IEEE Transactions on Service Computing, January 2014,K. Elissa, "Title of paper if known," unpublished.

[4] IT Pro Portal. (2018). PCI and GDPR: How to be cross-compliant. [online] Available at: https://www.itproportal.com/features/pci-and-gdpr-how-to-be-cross-compliant/ [Accessed 17 Aug. 2018].

[5] OWL: Web Ontology Language, Sean Bechhofer, https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_1073

[6] S. Soderland, Learning to extract text-based information from the world wide web, in KDD, vol. 97, 1997, pp. 251254

[7] Understanding the 12 Requirements of PCI DSS -Practical steps to achieve and maintain compliance, OpinionPiece[Online].www.dimensiondata.com/globalpresence. [Accessed: 21- Feb- 2016]

[8] K. P. Joshi and C. Pearce, "Automating Cloud Service Level Agreements Using Semantic Technologies," 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ,2015, pp. 416-421, doi: 10.1109/IC2E.2015.63

[9] EU GDPR Portal. (2018). GDPR Glossary of Terms. [online] Available at: https://www.eugdpr.org/glossary-of-terms.html [Accessed 17 Aug. 2018].

[10] "General Data Protection Regulation (GDPR) – Final text neatly arranged." General Data Protection Regulation (GDPR), gdpr-info.eu/.

[11] Nasr Al-Zaben , Md Mehedi Hassan Onik , Jinhong Yang , Nam-Yong Lee , Chul-Soo Kim.General Data Protection Regulation Complied Blockchain Architecture for Personally Identifiable Information Management

[12] OAuth Protocol, https://tools.ietf.org/html/rfc6749

[13] The truth about Blockchain, Harvard Business Reviews, https://hbr.org/2017/01/the-truth-about-blockchain.

[14] http://bitfury.com/content/5-white-papers-research/bitfury_white_paper_on_blockchain_auditability.pdf

[15] Karuna P. Joshi et al., Semantic Approach to Automating Management of Big Data Privacy Policies, in proceedings of IEEE Big Data, 2016.

[16] Guy Zyskind, Oz Nathan, Alex 'Sandy' Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data.2015 IEEE CS Security and Privacy Workshops.

[17] Wang,Huaiqing,Chen,Kun,Xu, Dongming.A maturity model for blockchain adoption, Financial Innovation, Dec 2016.

[18] D. McGuinness, F. Van Harmelen, et al., OWL web ontology language overview, W3C recommendation, World Wide Web Consortium, 2004.

[19] Chinmay Saraf, Siddharth Sabadra. Blockchain Compendium. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8376323

[20] Bin Yu1 et. Al . Platform-independent Secure Blockchain-Based Voting System. https://eprint.iacr.org/2018/657.pdf

[21] JIALU HAO1 et al . Efficient Attribute-based Access Control with Authorized Search in Cloud Storage. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8672564

[22] Stephen Kirkman. A Data Movement Policy Framework for Improving Trust in the Cloud Using Smart Contracts and Blockchains. https://ieeexplore.ieee.org/document/8360339

[23] Shinsaku Kiyomoto, Mohammad Shahriar Rahman, Anirban Basu. On Blockchain-Based Anonymized Dataset Distribution Platform. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7965711

[24] Shangping Wang, Yinglong Zhang , Yaling Zhang. A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. https://ieeexplore.ieee.org/document/8400511

[25] Kristin B. Cornelius, Department of Information Studies, University of California, Los Angeles, United States of America. Standard form and contracts and a smart contract future. https://policyreview.info/articles/analysis/standard-form-contracts-and-smart-contract-future

[26] Chun-Feng Liao. Toward A Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD styles.https://ieeexplore.ieee.org/document/8241535

[27] https://hyperledger-fabric.readthedocs.io/en/release-1.4/smartcontract/smartcontract.html

[28] Emanuel Regnath, Sebastian Steinhorst. SmaCoNat: Smart Contracts in Natural Language.https://s-steinhorst.github.io/PDF/2018-FDL-SmaCoNat%20-%20Smart%20Contracts%20in%20Natural%20Language.pdf

[29] https://gdpr-info.eu/

[30] https://bitsonblocks.net/2016/02/29/a-gentle-introduction-to-immutability-of-blockchains/

[31] Lavanya Elluri, Ankur Nagar, Karuna Pande Joshi. An Integrated Knowledge Graph to Automate GDPR and PCI DSS Compliance.

[32] Lavanya Elluri and Karuna Pande Joshi. A Knowledge Representation of Cloud Data controls for EU GDPR Compliance.

[33] Agniva Banerjee and Dr. Karuna Pande Joshi. Link Before You Share: Managing Privacy Policies through Blockchain.

[34] https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-javascript

[35] Srishty Saha, Karuna P. Joshi, Renee Frank, Michael Aebig, Jiayong Lin. Automated Knowledge Extraction from the Federal Acquisition Regulations System https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8258353

[36] The HIPAA PrivacyRulehttps://www.hhs.gov/hipaa/for professionals/privacy/index.html

[37] Payment Card Industry (PCI) Data Security Standard Version 3.2, https://www.pcisecuritystandards.org/document_library, August 2019

[38] Kazuhiro Yamashita et al. Potential Risks of Hyperledger Fabric Smart Contracts. https://ieeexplore.ieee.org/document/8666486