Project ID - 2014CSEPID05

Project Report

on

Anonymization and Analysis of Healthcare Data

Submitted In Partial Fulfillment of the Requirement

For the Degree of

Bachelor of Technology

In

Computer Science and Engineering Department

By

ASHISH SHUKLA (1429010030)
MOHIT KUMAR SAHNI (1429010078)
SOURAV AGGARWAL (1429010137)

Under the Supervision of

Prof. Bipin Kumar Rai

ABES INSTITUTE OF TECHNOLOGY
GHAZIABAD

AFFILIATED TO
Dr A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY,
UTTAR PRADESH, LUCKNOW
(MAY-2018)

TABLE OF CONTENTS

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature:
Name: Ashish Shukla
Roll No: 1429010030
Date:

Signature:
Name: Mohit Kumar Sahni
Roll No: 1429010078
Date:

Signature:
Name: Sourav Aggarwal
Roll No: 1429010137
Date:

CERTIFICATE

This is to certify that Project Report entitled "Anonymization and Analysis of Healthcare Data" which is submitted by Ashish Shukla, Mohit Kumar Sahni & Sourav Aggarwal in partial fulfillment of the requirement for the award of degree B. Tech. in Department of Computer Science & Engineering of Dr. A.P.J. Abdul Kalam Technical University, is a record of the candidates own work carried out by him under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Prof. Bipin Kumar Rai
Supervisor

Prof. Dilkeshwar Pandey
Head of Department (CSE)

Date:

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to Professor Bipin Kumar Rai, Head of Department, Department of Information Technology, ABES Institute of Technology, Ghaziabad for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavors have seen light of the day.

We also take the opportunity to acknowledge the contribution of Professor Dilkeshwar Pandey, Head of Department, Department of Computer Science & Engineering, ABES Institute of Technology, Ghaziabad for his full support and assistance during the development of the project.

We also would not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our parents and friends for their constant support throughout the project.

Signature:
Name: Ashish Shukla
Roll No: 1429010030
Date:

Signature:
Name: Mohit Kumar Sahni
Roll No: 1429010078
Date:

Signature:
Name: Sourav Aggarwal
Roll No: 1429010137
Date:

Abstract

Information explosion is radically changing our perception of the surroundings and healthcare data is at the core of it. The nature of healthcare data being extremely sensitive poses a threat of invasion of privacy of individuals if stored or Exported without taking proper security measures. De-identification involves pseudonymization or anonymization of data which are methods to disassociate an individual's identity temporarily or permanently respectively. These methods can be used to provide secrecy to user's healthcare data. A commonly overlooked weakness of de-identification techniques is Inference attacks.

The report discusses an approach to de-identify Enterprise Healthcare Records (EHR) using chained hashing for generating short-lived pseudonyms to minimize the effect of inference attacks and also outlines a re-identification mechanism focusing on information self-determination.

It also explores the pros and cons of existing de-identification algorithms and mechanisms and tries to resolve them by using appropriate real-time de-identification algorithm titled as "Ephemeral Pseudonym Generation Algorithm (EPGA)" proposed by the authors of the report.

LIST OF FIGURES

LIST OF SYMBOLS

T	Relation containing all patients
D	Relation containing all de-identification information of all patients
P	Relation containing pseudonyms for all patients
$\mathbf{t_i}$	Dith patient belonging to relation T
$\mathbf{U_i}$	Basic identity information of t_i
$\mathbf{g_i}$	Group ID of t_i
$\mathbf{gu_i}$	Unique ID in group for t_i
$\mathbf{Qt_i}$	List of Quasi Specifiers for t_i
$\mathbf{gQt_i}$	Generalized or suppressed list of Quasi Specifiers for t_i
$\mathbf{Egu_i}$	Ephemeral Unique ID in group for t_i
$\mathbf{H_i}$	Globally Unique ID for t_i to map report IDs
$\mathbf{RH_i}$	Unique Global ID for ith report
$\mathbf{H_m}$	Highly collision resistant Hashing algorithm
||	Concatenation Symbol
$\mathbf{E_i}$	Ephemeral ID for ith report
HMAC	Hash based message authentication coding function
$\mathbf{Kg_i}$	Key for creating H_i through ith patient

CHAPTER - 1
INTRODUCTION

1.1. INTRODUCTION

The amount of data is growing at an exponential rate, it's nearly doubling every two years and changing how we live in the world. One of the most important and critical data being the one produced by healthcare departments. As the confidential data accumulates further and further the risk of it's exposure becomes greater. In-fact formal laws are being amended for protection of individual data. Notably GDPR , where General Data Protection Regulation (GDPR) (Regulation EU 2016/679) is a regulation by which the European Parliament, the Council of the European Union and the European Commission intends to strengthen and unify data protection for all individuals within the European Union (EU).

GDPR is set to be applied from 25 May 2018 in EU. So the question arises, How do we ensure confidentiality of data ? In specific healthcare data. Suddenly the conventional thoughts on Information Security evoke Encryption as the solution for every Information Security need. But to our surprise Encryption is not a solution or rather a Bad solution. The argument against Encryption is that it poses linear overhead of deciphering and enciphering information and being dependent on Keys the major drawback in this case is what if the keys for data are leaked? And does enciphering data provides true confidentiality of our personal information?

No.

As our identity is still attached to the data with the help of a key it can be directly seen to whom does the information belong. Thus although it satisfies basic criteria of GDPR but GDPR recommends a method called Pseudonymization.

Pseudonymization is a technique in which we replace quasi-specifiers with codes. Where quasi-specifiers are the fields in a dataset that identify the individual to whom the dataset belongs, it can be ZIP Code, DOB, Gender, Name, Aadhar Number etc. Partial Encryption maybe one way to visualize this technique. In a scenario where the datasets must be facilitated to various departments for further processing or analysis. If the data is not pseudonymized upon sharing the data it reveals identity of the individual. But pseudonymization hides the individual's identity and also removes the overhead of deciphering data for analysis or processing. But the process of retrieving identity of a patient from the pseudonymized dataset isn't really tough once we get the hold of the pseudonym table. Thus it is required to introduce concept of anonymization.

Anonymization is a type of information sanitization whose intent is privacy protection. It is the process of removing personally identifiable information from datasets completely so that re-identification is impossible. Now the second problem is Control over 'who should be able to identify data and who should not be.' The problem can be solved using Role Based Access Control (RBAC). The control of who can access the data of an individual should be with individual himself. In terms of Healthcare data a patient should be able to

control who can access his records, e.g. a doctor who is examining a patient should be able to access his previous records. A family member upon being authorized by patient should be able to share his records. The access can further be refined to which particular reports and fields an authorized person can access.

The usage of patient data for research poses risks concerning the patients' privacy and informational self-determination. Technologies like NGS(Next Generation Sequencing) and various other methods obtain data from biospecimen, both for translational research and personalized medicine. If these biospecimen are anonymized individual research can not be associated back to the individual. Thus we need some means to reassociate the data to individuals.

The reassociation of identity with the data is called re-identification. Anonymization completely removes the re-identification possibility whereas pseudonymization retains it. As the healthcare data is essential for research purposes it is evident that anonymization is a better option than pseudonymization for it because of no need of the identities of patients.

The consequences of Pseudonymization are that it does not provide complete assurance of confidentiality of identity. But it removes the threat of direct-identification. As the clause 125 in IHE (Integrating the Healthcare Enterprise) Infrastructure Handbook states-

"It is important to understand that you can only reduce the risks. The only way to absolutely assure a person can not be relinked to their data is to provide no data at all. De-identified data can still be full of identifying information. And may still need extensive privacy Protections."

So it is essential to optimize de-identification process to remove or in case of pseudonymization , pseudonymize identifying information completely or to an extent that it can not be relinked to the patient.

1.2. EXISTING DE-IDENTIFICATION METHODS

De-identification is the process of removing or transforming sufficient information from the source data. The goal is that the risk of re-identification is reduced to an acceptable level while also achieving the objectives of the intended use. Pseudonymization is a particular type of de-identification that both removes the association with data subjects and adds an association between a particular set of characteristics relating to the data subject and one or more pseudonyms. In pseudonymization we may not necessarily replace each data with a unique pseudonym but we can also replace a batch of values for one pseudonym. Data fields that are less identifying are usually not pseudonymized.

1.2.1. Types of Pseudonymization – Based on reversibility there are two types of Pseudonymization -

1.2.1.1. Irreversible Pseudonymization – The pseudonymized data do not contain information that allows the re-establishment of the link between the pseudonymized data

and the data subject. This overlaps with anonymization. But preserves continuity for the pseudonym throughout the resulting dataset.

1.2.2.2. Reversible Pseudonymization – The pseudonymized data can be linked with the data subject by applying procedures restricted to duly authorized users.
Our primary interest for re-identifiable data would be Reversible Pseudonymization. Protection of healthcare demographics can be achieved using perturbative methods such as noise addition and data swapping. However these methods fail to preserve data truthfulness e.g. they may change the age of a patient from 50 to 10, which can severely harm the usefulness of the published patient data. Non perturbative methods preserve data truthfulness. It is important to remember data that is appropriately de-identified for one purpose may not be correctly de-identified for a new use of the data.

1.2.2. Pseudonymization Techniques - Pseudonymization is a de-identification technique in which we introduce a pseudonym in place of the attributes that directly or indirectly identify an individual. IHE defines it as a technique that uses controlled replacements to allow longitudinal linking and authorized re-identification. Let there be a relation T(a1,a2 ……, ad) for which QT is the set of Quasi-identifiers for relation T. where for i = (1, ..., m) ai ∈ QT then pseudonymization is essentially replacing QT with PT where PT = (P1, P2 …, Pm) Keeping another relation PT → QT for re-identification.

1.2.2.1. Peterson's approach - Robert L Peterson suggested a key-based approach to provide access control and encryption of medical information. The patient holds a Personal Key (PEK). This approach also involves assigning a static pseudonym to the individuals. There exists a Global Key (GK) which uniquely identifies the patient in the pseudonymized records when used jointly with PEK. The records are secured by encryption on database using PEK thus the entire security of information is revolving around the encryption of information. If PEK is stolen then this approach is rendered ineffective against attackers.

1.2.2.2. Slamanig and Stingl's Approach - This approach suggests storage of User Information and Medical Data on different databases. These two are mapped with the help of some central components. Same as Peterson's approach, Slamanig's approach also suggests storing data in encrypted form and giving the encryption key to the patient. It focuses on access control as well but doesn't ensure the security of data if the data is to be shared with a 3rd party (e.g. for research purpose).

Similar approaches were suggested by Pommerening and Thielscher as well. All of the approaches seem to be greatly affected by the problem of inference attacks as the used pseudonyms are persistent and eventually start to work as a unique identifier as the patient's information grows larger. Thus a need for variable or ephemeral pseudonyms arises to weaken the inference attacks.

1.2.3. Anonymization - Anonymization is a de-identification technique that dis-associates all identifiers from the data. For example, creating a teaching file for radiological images illustrating a specific condition requires anonymization of the data. Here the important point is that there is no requirement to be able to identify the patient later so all traces of

the patient should be removed and the data is made fully anonymous by manually reviewing the files and their fields to determine which fields are required for instructional purposes and which required fields can be used for re-identification of patient. In practice, such fields are rewritten to retain useful meaning while not disclosing any private information.
Anonymization has following three principles-
Let there be a relation T(a1,a2 ……, ad) for which QT is the set of Quasi-identifiers for relation T. where for i = (1, ..., m) ai ∈ QT .Then,

1.2.3.1. k-anonymity - Qti for ti∈ T should be indistinguishable from at least k-1, tj∈ T where j∈(1,...,d) and j != i. The process of enforcing k-anonymity is called k-anonymization in which T is partitioned into groups gj such that j ∈ (1...h) and | gj | < k, here |x| means the size of x. tuples in gj are made identical to the QT in process of k-anonymization.

1.2.3.2. l-diversity - Only providing k-anonymity may cause inference of an individual's values in the sensitive values (SA), this is called value disclosure. To prevent value disclosure each anonymized group must contain at least l well-represented values. Here well-represented value means distinct and leads to the principle called distinct l-diversity. which requires each anonymized group to contain at least l distinct SA values.

1.2.3.3. Recursive (c, l) diversity - Given parameters c,l, which are specified by data publishers, a group gj is (c,l)-diverse when r1 < c × (rl + rl+1 + ...+ rn), where ri,i ∈ {1,...,n} is the number of times the i-th frequent SA value appears in gj, and n is the domain size of gj. T is (c,l)-diverse when every gj, j = 1,...,h is (c,l)-diverse.

1.3. RISK FACTOR & INFERENCE ATTACKS

1.3.1. RISK FACTOR - Risk Management for HealthCare Data Anonymization involves considering the following factors while sharing anonymized data-
1. The likelihood of re-identification for each individual i, li due to previously released versions.
2. The severity of potentially released sensitive information is if reidentification actually happens.
3. The total expected utility (u) of sharing anonymized data.
4. The part of the population poi that could be affected by re-identification.
Using above risk factors we can easily define the potential risk in sharing anonymized data as follows -

$$\text{Risk} = \bigcup_{o}^{|s|} (s_i, l_i, po_i)$$

∪ here is a monotonically increasing function that combines the inputs and transforms them into a standardized range.

1.3.2. INFERENCE ATTACKS - Pseudonymized data is prone to inference attacks. The biggest loophole being persistent pseudonym usage. Inference attacks relate to data mining

techniques. If an adversary can infer the identity associated with some pseudonymized data with high confidence then the data is said to be leaked. As pseudonymization is not a technique of encryption and rather relies on hiding the identity of individuals, it is highly liable to this attack. Statistical frequency analysis attacks are a very basic example of inference attacks. Dataset aggregation techniques are also used heavily by attackers in order to derive an inference from existing datasets.

If there is a relation T(a1,a2 ……, ad). for which Q_T is the set of Quasi-identifiers for relation T. and there exists another relation D(d1,d2 ……, dd) which contains identification information about the individuals belonging to relation T.
if for i = (1, ..., m) ai $\in$ Q_t and ai $\in$ D then we can associate an identity based on the other attributes in the same tuple belonging to D.

One such example for EHR is evident with D_T as Voter List. If the pseudonymization was done on basis of YOB, ZIP, and Sex then for a particular state the total number of possible pseudonyms can be in the range of 10,000s.

Which is significantly low and the actual identity can be derived using further inferences. This particular inference attack was exploited heavily and caused the creation of HIPAA (Health Insurance Portability and Accountability Act of 1996). Nevertheless, inference attacks are still prevalent as although the process of formation of pseudonyms has significantly changed but the underlying loopholes remain the same and the persistence in pseudonyms poses a wide threat to user's privacy.
Based on these facts it's obvious that intuitive pseudonymization methods are almost certain to fail in order to provide privacy. Successful pseudonymization requires a deep knowledge of the data. It is necessary to design models keeping in mind that other datasets may be used in association with the existing records to derive identities.

CHAPTER - 2
METHODOLOGY

2.1. PROPOSED SOLUTION

The solution assumes that there exists an authorized body that regulates the identification information and provides a unique identifier for each resident. Let the identifier be represented by Ui , The patient is represented by ti ∈ T where T is set of all patient's identification records.

The system consists of 3 Nodes namely Accession Node, Key Node and Data Node. Accession Node enrolls the user in Healthcare system only once. It extracts Qti = (q1i , q2i …. , qni) (Quasi Specifiers for ti) from ti and transmits it to Key Node. Key Node applies 'Ephemeral Pseudonym Generation algorithm - Initialize' (EPGA-Init) on Qti which produces gi (ith group) and gui (unique ID in gi) for ti and initializes a report_schema for insertion of records in form of record IDs in Data Node corresponding to a Hi which is a hash of gui and gi.

Another relation is maintained for retrieval of gui through mapping of biometrics of patients at Key Node. Whole communication on the network is protected using ECDH (Elliptic Curve Diffie Hellman). There should exist a mapping of Ei (Ephemeral IDs) corresponding to each Hi, Ei will be used by healthcare services to insert and retrieve data for a patient. Ei will be generated for de-identification purposes in EPGA-D.

Each Ei is only one time usable thus it gives a strong protection against caching of pseudonyms and makes it hard to infer the identity of an individual from records. To reassociate the identity of individuals with Ui the user must provide his consent by providing the gui.

2.2. Ephemeral Pseudonym Generation Algorithm

EPGA is divided into three parts i.e. Initialize (EPGA-Init), De-identification (EPGA-D) and Re-identification (EPGA-R).

2.2.1. EPGA-Init - EPGA-Init Algorithm generates a global pseudonym Hi against which we will store the report_schema which will contain the Record IDs of the reports and other de-identified documents. In EPGA-Init generalize_or_suppress function returns the generalized form of an identifier else a null string if identifier should be suppressed. Hm is a highly collision resistant Hashing algorithm (e.g. SHA256). Kgi stands for ith group's key. The getLast function takes the argument as group id and returns the de-serialized object associated with that gid else returns Null if group id doesn't exist in Key Node's Database.

- **EPGA-Init(Qti)**:

1. gQti ⟵ generalize_or_suppress(qi: qi∈ Qti)

2. Kgi ← '\0'
3. Kgi || qi : qi ∈ gQti
4. gi ← Hm(Kgi)
5. count ← getLast(gi)
6. gui ← randomize_count(count)
7. Hi ← HMAC(gi || gui , key = Kgi)
8. return Hi , gui

To define getLast function we assume that there must exist a cQueue associated with each group id in Key Node's database which stores the counts of revoked gui corresponding to gi to avoid overflow in group unique ID's counts. randomize_count takes count as seed and maps the count to another number within a defined prime number's range. It only introduces randomness in generated group unique IDs.

- **getLast(gi)**:

1. retrieve gi row from database.
2. if gi doesn't exist in database:
   2.a. gi.count = 0
   2.b. return 0
3. cQueue ← deserialize(gi.cQueue)
4. if cQueue is null:
   4.a. return gi.count+1
5. else:
   5.a. count ← dequeue(cQueue)
   5.b. serialize(cQueue)
   5.c. update gi.cQueue in database
   5.d. return count

2.2.2. EPGA-D - EPGA-D Algorithm de-identifies the streaming data and fulfills the purpose of real-time de-identification of streaming data. If the data is being produced by a producer on a stream processing platform e.g. Kafka in a predefined format e.g. FHIR (Fast Healthcare Interoperability Resources) then we can apply EPGA-D on producer-end if the producer is reliable else on consumer-end on Data Node to de-identify data in real-time. The de-identification of a patient report is partially influenced by safe harbor method[5] which suggests suppression of 18 identifiers like Names, Locations, Dates directly relating to an individual, Telephone numbers, Fax numbers etc. The key difference being that EPGA-D assigns a short-lived pseudonym as the report's ID called Ephemeral ID (Ei) along with suppression of identifiers suggested in safe harbor method. The Ephemeral ID is generated by user's consent on report producer's end after providing gui. Upon receiving the pseudonymized data with Ei on Data Node, the Data Node generates a random identifier RHi and replaces Ei with RHi. RHi is updated in the report_schema corresponding to the patient's Hi who generated the Ei.
In order to generate Ei patient can send the request for the generation of Ei to Key Node through an authenticated medium by providing his gui and Ui.

- **createEi(gui , Ui):**

1. retrieve Qti from identification body through Ui.
2. gQti ← generalize_or_suppress(qi: qi∈ Qti)

3. gi ⟵ Hm(concat(qi) : qi ∈ gQti)
4. creates a random identifier Ei and associate it with the Hi.
5. return Ei

We further subdivide the EPGA-D algorithm into two parts i.e. @Producer and @Consumer where Producer is the segment that should be used on the stream's end which produces the de-identified report and Consumer is the stream's end which receives the de-identified report i.e. Data Node.

@Producer

- **EPGA-D(Report):**

1. Request patient to generate Ei .
2. Ei ⟵ createEi(gui , Ui)
3. gReport ⟵ generalize_or_suppress(Report)
4. gReport.id ⟵ Ei
5. Stream gReport on data pipeline.

@Consumer

- **EPGA-D(Report):**

1. create random identifier RHi.
2. Ei ⟵ Report.id
3. Request Hi corresponding to Ei from Key Node.
4. On receiving Hi request Key Node deletes Ei from the map and returns corresponding Hi.
5. Report.id ⟵ RHi
6. Save RHi in report_schema corresponding to Hi

2.2.3. EPGA-R - EPGA-R Algorithm re-associates the identity of an individual with a Report with the explicit consent of the patient. The patient generates a short-lived one-time usable Ephemeral Group Unique ID (Egui) by providing his gui, Ui and Lifetime of Egui. In case the patient does not provide the lifetime of Egui a default timeout must be set up to prevent misuse of Egui through malevolent attempts.
In order to generate Egui patient can send the request for the generation of Ei to Key Node through an authenticated medium by providing his gui , Ui and optionally the time to live (ttl) for Egui.

- **createEgui(gui , Ui , ttl = default_time)**:

1. retrieve Qti from identification body through Ui.
2. gQti ⟵ generalize_or_suppress(qi: qi∈ Qti)
3. gi ⟵ Hm(concat(qi) : qi ∈ gQti)
4. creates a random identifier Egui and associate it with the Hi.
5. Set ttl of Egui.
6. return Egui

Let us assume there exists a 'Service' which wants to re-identify the patient.

- **EPGA-R(gui , Ui)**:

1. Service requests patient to generate Ei.

2. Egui ← createEgui(gui , Ui, optional_ttl)
3. Service requests patient to provide Ui.
4. Service sends Ui and Egui to DataNode.
5. Data Node requests Key Node to return Hi corresponding to Egui and Ui .
6. KeyNode returns the Hi to Data Node and deletes Egui.
7. DataNode returns requested data associated with Hi to the Service.

2.3. IMPLEMENTATION OF EPGA

2.3.1. VRAHAD - VRAHAD stands for Vast Re-identifiable Aadhaar-based Healthcare Associated Data-records. It is a proof of concept for Ephemeral Pseudonym Generation Algorithm. It provides a GUI for registering users into the system and also pseudonymizes a healthcare record based on the mock-data. The system involves three major modules. The major modules along with their functionality are as follows-

2.3.1.1. Device Registration - The device from which the employee of healthcare center sends requests to register a resident must be verified. Thus it is first registered by an SPOC who is already authorized manually by a central authority for a certain Healthcare center. Here healthcare centers are the centers where residents can go and sign themselves up for accessing the VRAHAD's services. The workflow follows following steps for device registration -

1. Send request to KeyServer to verify SPOC SPOCID and password. We will use ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) for securing the communication.
2. KeyServer generates a ECC key pair and save the PrivateKey in temporary cache SPOCID and returns the PublicKey corresponding to it.
3. SYSID module creates DeviceID and returns the same with the Token to the KeyServer.
4. If the Decrypted Text is the SPOCID corresponding to the Token then save the PrivateKey in KeyDB corresponding to the SPOCID+"CURR_DEVICE" 's value with SPOCID as ForeignKey.
5. Device is now registered and has a PublicKey Associated with it.

Every Request Here on Will Have X-Device Header associated with it. Which will have Device ID in it.

2.3.1.2. Verify Employee - Second loophole in the system is the employee itself who will be registering the residents from Healthcare Center thus he must also be verified. His verification workflow follows following steps-

1. SignUP Module sends EmployeeID with Password to KeyServer with X-Device Header.
2. Compute HMAC of the data dump and associate it in POST as "HMAC" key = pubkey Compute Signature of UnEncrypted JSON Data and associate it in POST as "SIGN" Encrypt the Data with Received Public Key as EDat.

   EDat = {"U":E_i, "P":Pwd}

Objective here is to verify that the employee indeed is trying to login to system and System is Fingerprinted. This objective can only be fulfilled by verifying employee with both biometrics and OTP.

3. Get UID of Employee from DB and send OTPInit on PhoneNumber on Success Response of OTPInit, create a Token and save it corresponding to EmloyeeID.
4. SingUP@VerifyEmployee forward to OTP page (To Secure URL on redirect send NewToken with EmployeeID to RedisREST@HCenter and render the page with hidden input as the received NewToken)
5. Forward OTP received on Mobile and Fingerprint Hash with new token as value corresponding to EDat.
6. Invoke AuthInit with them on KeyServer if response is OK then change the existing Token with AuthToken with 3600 seconds timeout and return the encrypted token.
7. Redirect to EnrollResident choice page.

2.3.1.3. EnrollResident - After verifying himself the employee may login to the dashboard using following steps to enroll a resident -

1. With this EmployeeID, AuthToken on load of EnrollResident we create a Socket Connection with the SessionServer.
2. The Socket serves as a proof that the Resident Enroll requests are recent.
3. If the Socket connection drops, It's associated EmployeeID's AuthToken will be deleted i.e. Session ends.
4. The employee enters his credentials in order to login to the dashboard and an OTP is sent to his mobile number that is associated with his Unique Identification Number which we already have in our Niraadhaar Database.
5. The employee scans his biometrics and enters the OTP and submits them to the AUA server which in turn verifies him from Niraadhaar servers.
6. Upon successful authentication the employee logs in and then either scans the resident's aadhaar card else enters his credentials directly which are then forwarded after entering OTP and scanning thumb of the resident.
7. After successfully authenticating the resident a GUID is generated as per EPGA which is securely sent as a text message and a mail to the resident's registered mail and mobile.

2.3.2. Technologies Used - The project involves usage of various technologies from diversified technical domains which are as follows-

2.3.2.1. Python - Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990 and first appeared on 20th Feb 1991. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python design offers some support for functional programming in the Lisp tradition as well. The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a

higher plethora over other programming languages used in the industry. Some of its advantages are-

1. any features like string manipulations, internet, web service tools, operating system interfaces and protocols. Most of the well and widely used programming tasks are already scripted into it.
2. **Integration Feature** - Python have powerful control capabilities as it calls directly through C**Extensive Support of Libraries** - Python have large standard libraries that include m, C++, and JAVA.
3. **Improved Programmer's Productivity** - Python has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the language like Java, VB, Perl, C, C++ and C#.

Some of Python's notable features are :

1. Uses an elegant syntax,making the programs you write easier to read.
2. Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.
3. Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.
4. Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
5. Is easily extended by adding new modules implemented in a compiled language such as C or C++.
6. Can also be embedded into an application to provide a programmable interface.
7. Runs anywhere, including Mac OS X, Windows, Linux, Unix.
8. Is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed, because while the language is copyrighted it's available under an open source license.

2.3.2.2. Django - Django is a free and open-source web framework, written in Python, which follows the Model-View-Template (MVT) architectural pattern. It is maintained by the Django Software Foundation(DSF), and independent organization established as non-profit. Django is a high-level Python Web Framework that encourages rapid development and clean pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasize reusability and pluggability of components,

less code, low coupling, rapid development, and the principle of ‘don’t repeat yourself’.

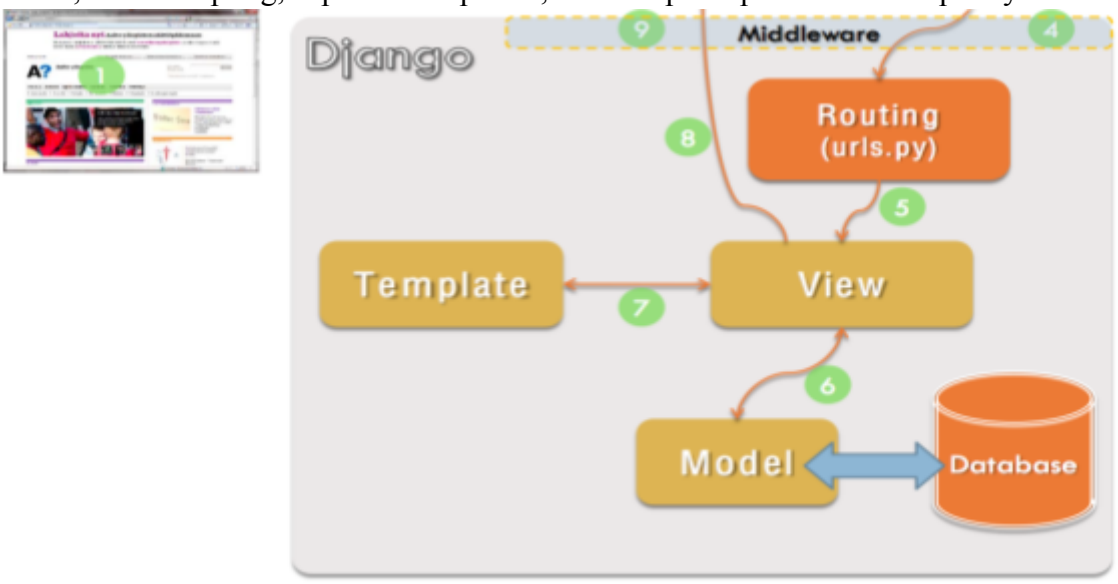


Fig - 1 Django MVC

Some of the advantages of Django are :

1. **Ridiculously Fast** - Django was developed to help developers take application from concept to completion as quickly as possible.
2. **Fully loaded** - Django includes dozens of extras you can use to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks — right out of the box.
3. **Reassuringly secure** - Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.
4. **Exceedingly scalable** - Some of the busiest sites on the planet use Django’s ability to quickly and flexibly scale to meet the heaviest traffic demands.
5. **Incredibly versatile** - Companies, organizations and governments have used Django to build all sorts of things — from content management systems to social networks to scientific computing platforms.

Some of the notable features of Django are -

1. Django is based on Python, which is a popular language that's easy to learn and powerful. Python works on any platform and is also open source.
2. Django's official documentation is comprehensive and includes easy to follow tutorials.
3. Django has an active community that contribute via open packages, tutorials, books and code snippets.
4. Because Django takes the "batteries included" approach, downloading it and getting started is quick and easy.
5. With Django's built-in admin interface, it's easy to organize model fields or process data.
6. Django adapts the well-known MVC architecture and therefore code can be written with clean abstractions.
7. Django community makes regular stable releases, some of which are Long Term Support (LTS) releases that get three years of support.

2.3.2.3. Django REST Framework - With the rise of Single-Page-Applications and the trend of separating monoliths into services with distinct front and backends, knowing how to make your own RESTful API for your backend is more important than ever. Officially, a RESTful API is a Representational State Transfer-ful Application Programming Interface. In other words, it means to putting data onto your web server in a way that's accessible to other servers and clients, and it works through HTTP requests and responses and carefully structured URL routes to represent specific resource(s). Django Rest Framework (or simply DRF) is a powerful module for building web APIs. It's very easy to build model-backed APIs that have authentication policies and are browsable. With its free and open source, it makes the web development process very easy and the developer can fully focus on the designing process and boost performance. Thus, Django becomes an ideal tool for startups, where web design is the need, to bring out the real concept and prospects of the company. With its free and open source, it makes the web development process very easy and the developer can fully focus on the designing process and boost performance. Thus, Django becomes an ideal tool for startups, where web design is the need, to bring out the real concept and prospects of the company.
Some of the notable advantages of DRF are -

1. **Fast** - This has been designed in a way to help the developers make an application as fast as possible. From idea, production to release, Django helps in making it both cost effective and efficient. Thus it becomes an ideal solution for developers having a primary focus on deadlines.
2. **Fully Loaded** - It works in a way that includes dozens of extras to help with user authentication, site-maps, content administration, RSS feeds and much more such things. These aspects help in carrying out the web development process completely.
3. **Secure** - When you are doing it in Django, it is ensured that developers don't commit any mistakes related to security. Some of the common mistakes include SQL injection, cross-site request forgery, clickjacking and cross-site scripting. To manage effectively usernames and passwords, the user authentication system is the key.
4. **Scalable** - To meet the heaviest traffic demand, the benefits of Django framework can be seen. Therefore, the busiest sites use this medium to quickly meet the traffic demands.
5. **Versatile** - Content management, scientific computing platforms, and even big organizations, all these aspects are very efficiently managed by the use of Django.

Other features that DRF are :

1. **Authentication** - From basic and session-based authentication to token-based and Oauth2 capabilities, DRF is king.
2. **Serialization** - It supports both ORM and Non-ORM data sources and seamlessly integrates with your database.
3. **Great Documentation** - If you get stuck somewhere, you can refer to it's vast online documentation and great community support.

2.3.2.4. Apache Kafka - Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Its storage layer is essentially a "massively scalable pub/sub message

queue architected as a distributed transaction log,"] making it highly valuable for enterprise infrastructures to process streaming data. Additionally, Kafka connects to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. Apache Kafka is a distributed streaming platform.

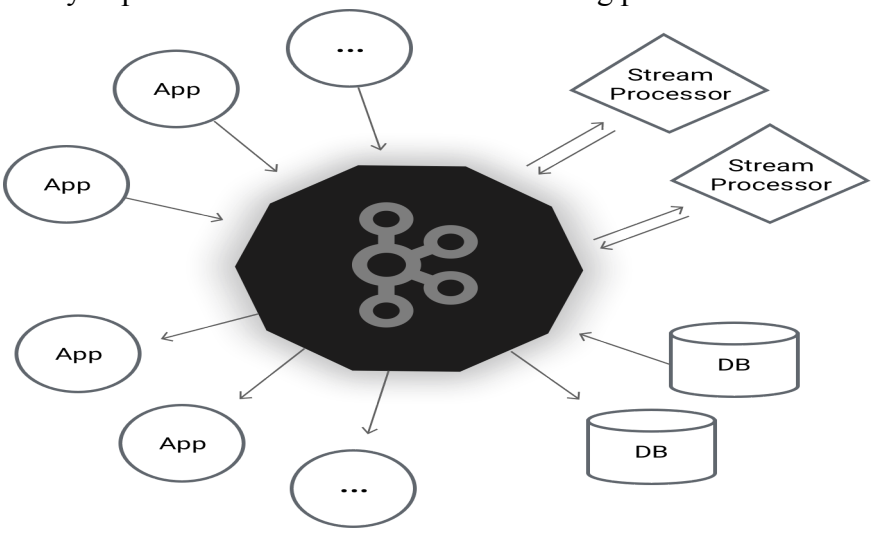


Fig 2- Kafka Architecture

A streaming platform has three key capabilities:

1. Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
2. Store streams of records in a fault-tolerant durable way.
3. Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

1. Building real-time streaming data pipelines that reliably get data between systems or applications.
2. Building real-time streaming applications that transform or react to the streams of data.

Kafka provides following requirements -

1. High throughput, supporting hundreds of thousands of messages per second, even with modest hardware.
2. Persistent messaging with O(1) disk structures that provide constant time performance, even with terabytes of stored messages.
3. Explicit support for partitioning messages over Kafka servers and distributing consumption over a cluster of consumer machines while maintaining per-partition ordering semantics.

Also Kafka have several advantages over other streaming platforms -

1. Kafka is highly scalable.
2. Kafka is highly durable.
3. Kafka is reliable.
4. Kafka offers high performance.
5. Kafka has high throughput.
6. Kafka provides low latency.

2.3.2.5. BigChainDB - BigchainDB allows developers and enterprise to deploy blockchain proof-of-concepts, platforms and applications with a scalable blockchain database.

BigchainDB supports a wide range of industries and use cases from identity and intellectual property to supply chains, energy, IoT and financial ecosystems without sacrificing scale, security or performance.. With high throughput, sub-second latency and powerful crypto-condition escrow functionality to automate release-of-assets, BigchainDB looks, acts and feels like a database but has the core blockchain characteristics that enterprises want.

BigchainDB is a scalable blockchain database. It's designed to merge the best of two worlds: the "traditional" distributed database world and the "traditional" blockchain world.

BigchainDB starts with a traditional distributed database (initially RethinkDB), which has characteristics of:

1. Scale.
2. Queryability.

From that, it posses blockchain characteristics:

1. decentralized (no single entity owns or controls it)
2. immutable (tamper-resistance)
3. assets (you own the asset if you own the private key, aka blockchain-style permissioning)

Rather than trying to enhance blockchain technology, BigchainDB starts with a big data distributed database and then adds blockchain characteristics - decentralized control, immutability and the transfer of digital assets.

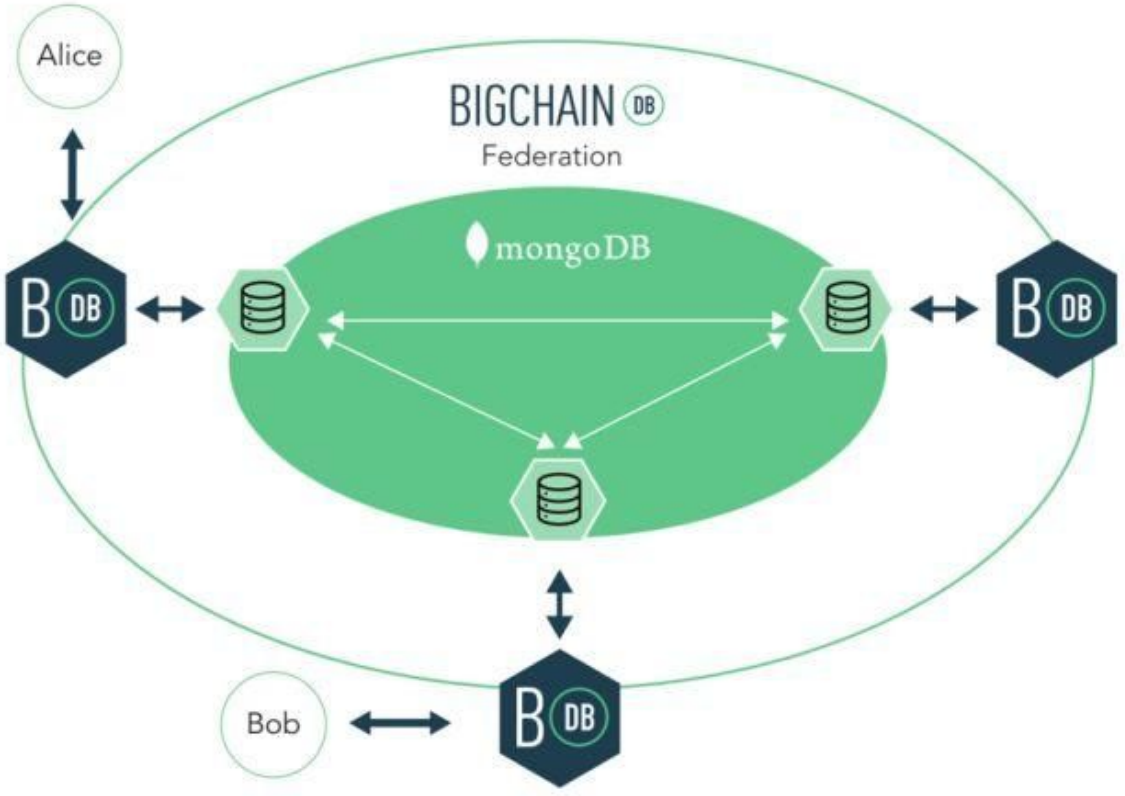


Fig 3 - BigChainDB Architecture

2.3.2.6. PostgreSQL - PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards

compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users; on macOS Server, PostgreSQL is the default database; and it is also available for Microsoft Windows and Linux (supplied in most distributions).
PostgreSQL is ACID-compliant and transactional. PostgreSQL has updatable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other expandability. PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

Advantages of PostgreSQL are :

1. Open Source DBMS - Only PostgreSQL provides enterprise-class performance and functions among current Open Source DBMS with no end of development possibilities. Also, PostgreSQL users can directly participate in the community and post and share inconveniences and bugs.
2. Diverse Community - One of the characteristics of PostgreSQL is that there are a wide variety of communities. Regarding PostgreSQL as Open Source DBMS, users themselves can develop modules and propose the module to the community. The development possibility is superiorly high with collecting opinions from its own global community organized with all different kinds of people. Collective Intelligence, as some might call it, facilitates transmission of indigenous knowledge greatly within the communities.
3. Function - SQL functions called ‘Store Procedure’ can be used for server environment. Also, we support languages similar to PL/SQL in Oralcle such as PL/pgSQL, PL/Python, PL/Perl, C/C++, and PL/R.
4. ACID and Transaction - PostgreSQL support ACID(Atomicity, Consistency, Isolation, Durability).
5. Diverse indexing techniques - PostgreSQL not only provides B+ tree index techniques, but various kinds of techniques such as GIN(Generalized Inverted Index), and GiST(Generalized Search Tree), etc as well.
6. Flexible Full-text search - Full-text search is available when searching for strings with execution of vector operation and string search.

2.3.2.7. Redis - Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.
In order to achieve its outstanding performance, Redis works with an in-memory dataset. Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. Persistence can be optionally disabled, if you just need a feature-rich, networked, in-memory cache.

Redis also supports trivial-to-setup master-slave asynchronous replication, with very fast non-blocking first synchronization, auto-reconnection with partial resynchronization on net split.

1. Other features include
2. Transactions
3. Publisher/Subscriber
4. Lua Scripting
5. Keys with a limited time-to-live
6. LRU eviction of keys
7. Automatic failover

Following are certain advantages of Redis:

1. Exceptionally fast − Redis is very fast and can perform about 110000 SETs per second, about 81000 GETs per second.
2. Supports rich data types − Redis natively supports most of the datatypes that developers already know such as list, set, sorted set, and hashes. This makes it easy to solve a variety of problems as we know which problem can be handled better by which data type.
3. Operations are atomic − All Redis operations are atomic, which ensures that if two clients concurrently access, Redis server will receive the updated value.
4. Multi-utility tool − Redis is a multi-utility tool and can be used in a number of use cases such as caching, messaging-queues (Redis natively supports Publish/Subscribe), any short-lived data in your application, such as web application sessions, web page hit counts, etc.

2.3.2.8. React Native - React Native is the next generation of React - a JavaScript code library developed by Facebook and Instagram, which was released on Github in 2013. React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. It's based on React, Facebook's JavaScript library for building user interfaces, but instead of targeting the browser, it targets mobile platforms. In other words: web developers can now write mobile applications that look and feel truly "native," all from the comfort of a JavaScript library that we already know and love. Plus, because most of the code you write can be shared between platforms, React Native makes it easy to simultaneously develop for both Android and iOS.

Similar to React for the Web, React Native applications are written using a mixture of JavaScript and XML-esque markup, known as JSX. Then, under the hood, the React Native "bridge" invokes the native rendering APIs in Objective-C (for iOS) or Java (for Android). Thus, your application will render using real mobile UI components, notwebviews, and will look and feel like any other mobile application. React Native also exposes JavaScript interfaces for platform APIs, so your React Native apps can access platform features like the phone camera, or the user's location.

Notable advantages of React Native are -

1. Covered Android and IOS : Initially, Facebook only developed React Native to support iOS. However with its recent support of the Android operating system, the library can now render mobile UIs for both platforms. Facebook used React Native to build its own Ads Manager app, creating both an iOS and an Android version. Both versions were built by the same team of developers.
2. Reusable components allow hybrid apps to render natively.

3. React Native UI components to an existing app's code - without any rewriting at all.
4. It's one of the top mobile JS frameworks among developers and growing.
5. It offers third party plugin compatibility less memory usage and a smoother experience.

2.3.3. Project Architecture - The project has following objectives-

A. Authentication of Device that will be used to enroll the end-user in the main system.
B. Securing the transmission channel.
C. Authenticating the employee on the enrollment center.
D. And De-Identification of the end-user's identification data.

The Project Architecture Diagram consist of following sub modules -

2.3.3.1. CLI Clients - CLI Client is the module for the Authentication of the devices which are installed on the end-user enrollment centers. This module further contains two modules as

System Verification - System verification module is the core module that provides a way to verify a system uniquely with the help of the properties of system e.g. dmidecode information. This information is then used to create a unique hash which is sent to the main key server.

authSPOC - authSPOC is the key person of the enrollment-center and is assumed to be already verified who is then provided with the System Verification Script. This authSPOC person is also provided his/her username and password which will be asked at the time of System Verification Script execution. authSPOC is also responsible for the enrollment of the employees of the enrollment center. Every time when an employee is to be enrolled the personal details and a valid mobile number of the employee is to be registered on the main system.

2.3.3.2. SignUp - SignUp Module is responsible for the proper sign-up of the end-user. There are two sub modules in this module as -

Verify Employee - Verify Employee module is responsible for the employee verification on the enrollment center at the time of end-user enrollment. It is important to note that the end-user is supposed to be enrolled through the verified device and by the verified employee of the enrollment center. For this we have already seen that the employee is registered on the main system. Now when an end-user come to enroll him/her-self, then end-user provides the necessary details. However when the employee submit all the details of the end-user, the employee is supposed to enter the valid credentials and an OTP that is sent on the mobile number of the employee. Employee is also supposed to provide the biometric information. In this way the employee is also verified.

EnrollResident - Upon submitting valid and necessary details the system gets the response from the main system that consists of the information that is supposed to be related to the end-user. This information is also sent to the E-Mail and Mobile Number as well and enrollment center also provides a hard copy of this information.

2.3.3.3. DB and REST - All the servers and databases in the main system are integrated using REST APIs. DB and REST is the module that is the REST API and Database used as the main system. It consists of the following sub module -

Session Server - Session server is used to maintain the sessions at many places. A Redis REST API linked with Redis Database is used for the purpose of maintaining the sessions as Redis provides the functionality such that the session key will automatically be deleted from the database after a particular amount of time.

Key Server - Key server consists of following two -
PostgresREST - postgresREST is responsible to and linked to PostgreSQL which is used to store all the Keys in KeyDB. Also AadharDB is maintained in this API as well.
Vrahad Init- CA : It is a certification authority that is pre certified by OpenCA module. It is responsible for providing the Digital certificate to all the AUAs.

2.3.3.4. Database and Schemas -Starting from the Aadhar Database that is maintained in the PostgreSQL have the three main tables as following -
Resident - Resident is a person of the country. And the details of this person is stored in this table. The attributes of the table is shown in the image.

AUA - Main system that is VRAHAD is supposed to be enrolled in the Aadhar DB. This table contains the information of the AUAs that are present in the system.

ASA - ASAs are also maintained in the system. And HelthCenters are registered in the ASA table.

Now we have one more system as KeyDB that maintain the data of following :
Health Center - All the Health Centers present across the nation are supposed to be registered in this table.

HCenterSPOC - Health Center ID in the HealthCenter table is a foreign key to this table. This table maintain the data of SPOC of the corresponding Health Center.

Fingerprint(Systems) - All the devices present on the Health center that are registered by the HCenterSPOC are maintained in this table. hCenterSPOC is a foreign key to this table.

HCenterEmployee - This table is used to maintain the data of employees registered by the hCenterSPOC. Also hCenterSPOC is a foreign key to this table.

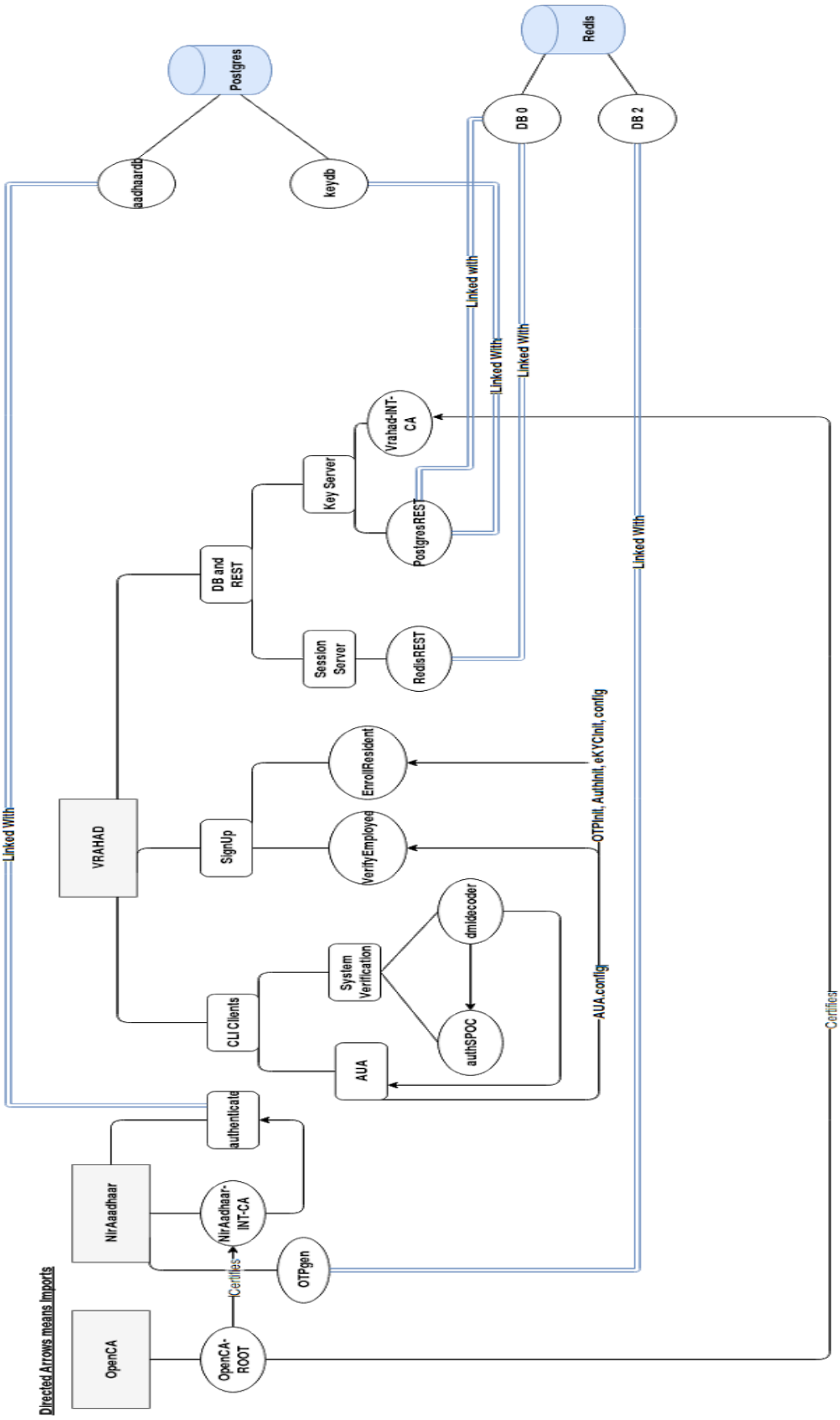


Fig 4 - Project Module Distribution

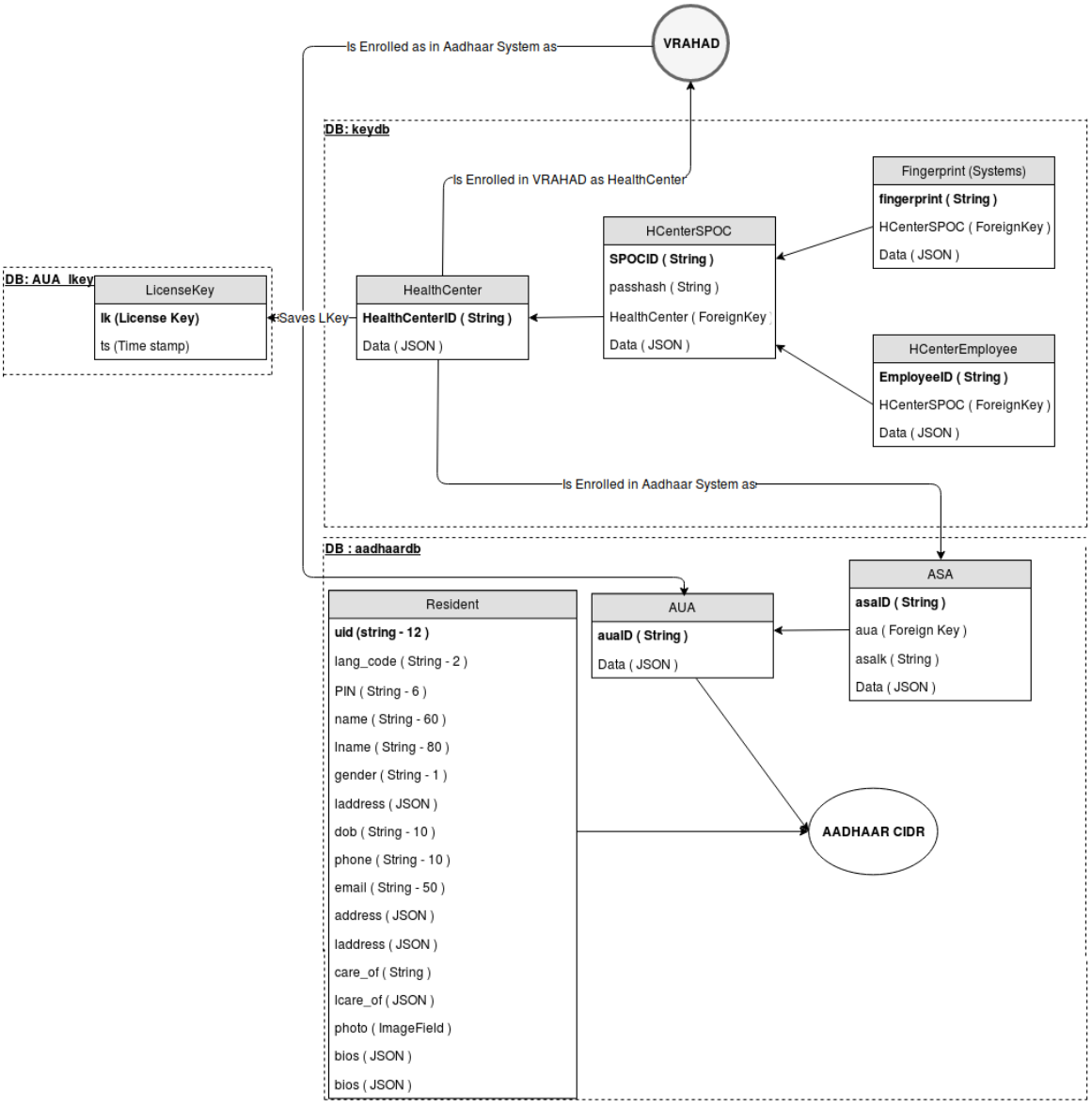


Fig 5 - Project DB Schema

2.3.4. Project Modules - Many of the dependencies in order to build the system were not available as open-source or even available at all thus few sub modules needed to be built as a part of fulfilment of dependencies. Those modules are as follows -

2.3.4.1. NirAadhaar - It's an implementation aimed at mimicking the working of UIDAI's Aadhaar's Authentication, eKYC and OTP APIs to use on local environment. It is built using Django framework and involves a SOAP based interaction. It fully replicates the procedure involving the implementation of ASA's and AUA's as suggested by the UIDAI's information manual. Primarily it has 3 modules as OTPGen, authenticate and eKYC and uses PostgreSQL as backend although its independent of a seperate DB dependency as it internally can work using solely SQLite3. Instead of using the Public Test Aadhaar API this module was fundamental in realizing the VRAHAD system.

2.3.4.2. OpenCA - The architecture involves certificates based on ECC whereas usual CA's work based on RSA. Thus a library was required that can handle the complications of CA's and certification process. OpenCA is a tool based on pyOpenSSL to easily create and manage Certification Authorities. Following is how we can use this API-

```
from OpenCA import createCA, signReqCA, createCSR
createCA('root','ROOT','root-pass',{'CN':'FQDN_ROOT'})
createCA('int','INTERMEDIATE','inter-pass',{'CN':'FQDN_INETRM
EDIATE'})

signReqCA('ROOT','INTERMEDIATE','root-pass','ca')

createCSR('USER','user-pass',{'CN':'FQDN_USER'})
createCSR('SERVER','server-pass',{'CN':'FQDN_SERVER'})

signReqCA('INTERMEDIATE','USER.csr.pem','inter-pass','usr')
signReqCA('INTERMEDIATE','SERVER.csr.pem','inter-pass','svr')

from OpenCA import Utils
Utils.verify_chain('ROOT/certs/ROOT.cert.pem',open('INTERMEDI
ATE/certs/INTERMEDIATE.cert.pem','rb').read()) # True

Utils.verify_chain('ROOT/certs/ROOT.cert.pem',open('USER.cert
.pem','rb').read()) # False
Utils.verify_chain('ROOT/certs/ROOT.cert.pem',open('SERVER.ce
rt.pem','rb').read()) # False
Utils.verify_chain('INTERMEDIATE/certs/INTERMEDIATE.cert.pem'
,open('USER.cert.pem','rb').read()) # False
Utils.verify_chain('INTERMEDIATE/certs/INTERMEDIATE.cert.pem'
,open('SERVER.cert.pem','rb').read()) # False

# End Certificates can only be verified using the chain of
trust

Utils.verify_chain('INTERMEDIATE/certs/ROOT.INTERMEDIATE.chai
n.pem',open('USER.cert.pem','rb').read()) # True
Utils.verify_chain('INTERMEDIATE/certs/ROOT.INTERMEDIATE.chai
n.pem',open('SERVER.cert.pem','rb').read()) # True
```

create ROOT CA -

```
from OpenCA import createCA
createCA('root','ROOT_NAME','ROOT_PASS',
{'CN':'FQDN.Goes.Here'})
```

create Intermediate CA -

```
from OpenCA import createCA, signReqCA

createCA('int', 'INTERMEDIATE_NAME', 'INT_PASS',
{'CN':'FQDN.Should.Not.Be.Same.As.Of.Root.CA'})
signReqCA('PATH_TO_ROOT_CA_FOLDER','PATH_TO_INTERMEDIATE_CA_F
OLDER','ROOT_PASS', csr_type = 'ca' )
```

signReqCA saves the certificate of Intermediate CA in ROOT CA's *newcerts* directory and enrolls it in index.db. return value of signReqCA is the certificate bytes of Intermediate CA's generated certificate.
For user or servers -
Users/server generates a PKey and CSR and hands it over to Intermediate CA.

```
from OpenCA import createCSR
createCSR('User','User_password',{'CN':'USER_FQDN'})
```

It will create two files in the current directory -
1.User.private.pem
2.User.csr.pem

create End user certificate on Intermediate CA-

```
from OpenCA import signReqCA

signReqCA('PATH_TO_INTERMEDIATE_CA_FOLDER','PATH_TO_CSR_OF_US
ER_OR_SERVER','INT_PASS', csr_type = <'usr' or 'svr')
```

2.3.4.3. PyZIPIN - In order to extract information from ZIP codes a module was needed, this requirement was fulfiled using PyZIPIN. Functionality of the same is as follows -

decode: Convert ZIP to Information(District, State etc.).
encode: Convert District to ZIP.
return types are in JSON strings. use after loading with json module's loads

```
import PyZIPIN as PZ
from json import loads

PZ.decode("209801")
```

```
# '{"officename": "Itauli B.O", "pincode": "209801",
"officetype": "B.O", "Deliverystatus": "Delivery",
"divisionname": "Kanpur
# Moffusil", "regionname": "Kanpur", "circlename": "Uttar
Pradesh", "taluk": "Unnao", "districtname": "Unnao",
"statename":
# "UTTAR PRADESH"}'

PZ.encode("Unnao")

# '{"pincode": "209831", "officename": "Atesuwa B.O"}'

print(loads(PZ.encode('Unnao'))['pincode'])

# 209831
```

One ZIP may belong to more than one offices so to list all the results matching pass additional argument all_results = True to both of the functions.

```
PZ.encode("Unnao",all_results=True)

# '[{"pincode": "209831", "officename": "Atesuwa B.O"},
{"pincode": "209863", "officename": "Dodiya Khera B.O"},
# .... {"pincode": "209801", "officename": "Unnao H.O"}]'
```

2.3.4.4. BioSampler - BioSampler is a utility application built using React Native framework that works on iOS and Android which scans the user's fingerprints using the sensors available on the device, matches them to the existing fingerprints on the device and returns a specified acknowledgement to the URI that it takes as input on the start of application boot.

2.3.4.5. SYSID - In order to uniquely identify each system a utility is required. SYSID is a utility that works on all UNIX based systems and generates a unique identification number for all the devices. It is especially useful to have a unique identification for each device when we want to verify that the response is coming from the registered device or not. It creates the system ID using dmidecode functionality of UNIX systems and utilizes HMAC based on sha256 with Processor ID, BaseBoard Serial & System UUID.

CHAPTER - 3
RESULTS

3.1. EXPORT OF DE-IDENTIFIED DATA

The root purpose of de-identifying data is so that we can export it later for research purposes. In order to extract information from the existing database based on the input given by the research group we can use following procedure, Let us consider that the requested information was a set I = $\{i_0 , i_1 , ….., i_n\}$. We induce the information set I in the pipeline to extract data and generalize I to get G, We create a unique ID using the members of G and extract corresponding RH_{id}.

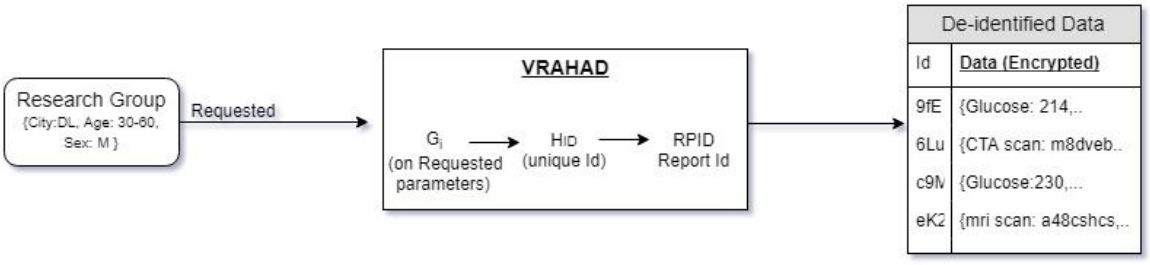


Fig 6 - Data Export Pipeline

3.2. SCALABILITY AND DE-IDENTIFICATION RATE IN EPGA

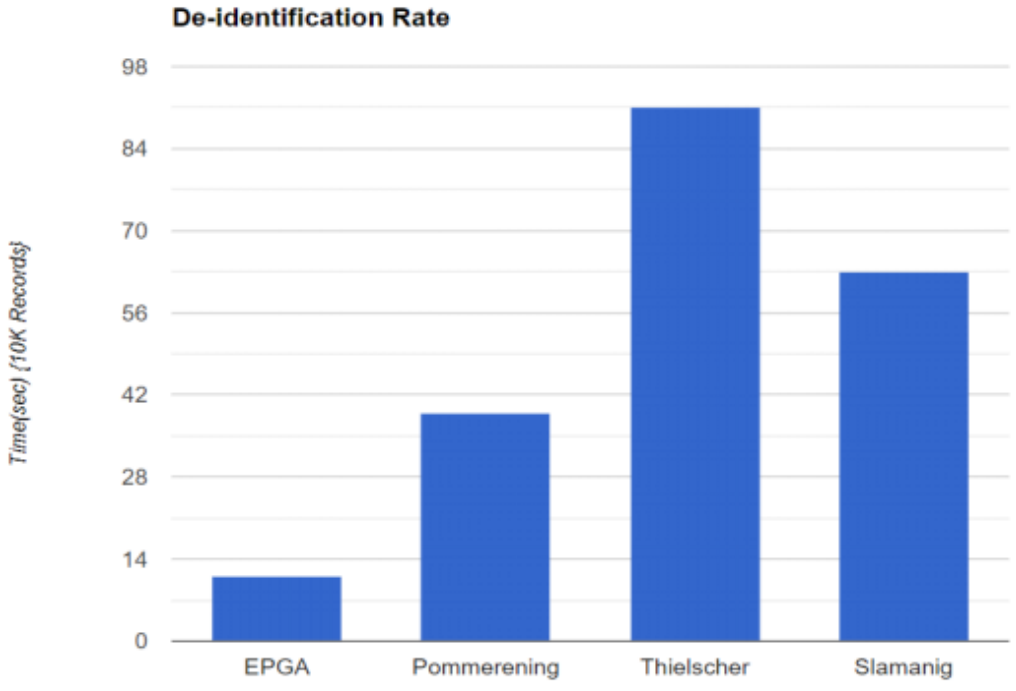


Fig 7 - De-Identification Rate of Algorithms

Scalability and Speed is always been an issue in real world implementation of any problem. Our suggested approach i.g. EPGA algorithm is better when it comes to scalability and rate of de-identification. Other approaches e.g. Pommerening, Thiescher, and Slamanig approaches are slow when applied to real-time streaming as well on a large batch data.
Since we are using PostgreSQL database which have a feature of ease of scalability. The PostgreSQL always run on a cluster mod in which we have an ease of adding or removing a node. Hence it's easy to scale the system.

Speed however differ when we take the real-time streaming data and batch time processing. In real-time streaming data, it depends on the streaming platform i.g. Kafka, since Kafka have a high throughput and low latency rate it's always been ease to use. Now when it's about batch processing the EPGA is better in comparison of the standard algorithms.

3.3. PROTECTION AGAINST INFERENCE ATTACKS

As EPGA-D Algorithm de-identifies the streaming data and fulfills the purpose of real-time de-identification of streaming data. If the data is being produced by a producer on a stream processing platform e.g. Kafka in a predefined format e.g. FHIR (Fast Healthcare Interoperability Resources) then we can apply EPGA-D on producer-end if the producer is reliable else on consumer-end on Data Node to de-identify data in real-time. The de-identification of a patient report is partially influenced by safe harbor method[5] which suggests suppression of 18 identifiers like Names, Locations, Dates directly relating to an individual, Telephone numbers, Fax numbers etc. The key difference being that EPGA-D assigns a short-lived pseudonym as the report's ID called Ephemeral ID (Ei) along with suppression of identifiers suggested in safe harbor method. The Ephemeral ID is generated by user's consent on report producer's end after providing gui. Upon receiving the pseudonymized data with Ei on Data Node, the Data Node generates a random identifier RHi and replaces Ei with RHi. RHi is updated in the report_schema corresponding to the patient's Hi who generated the Ei.
In order to generate Ei patient can send the request for the generation of Ei to Key Node through an authenticated medium by providing his gui and Ui.
EPGA-D provides a fairly complex relation between report ID and Hi which makes it hard to find a straight relation between reports and patient pseudonyms making inference attacks less effective. To reduce the effect of inference attacks even more we can split report_schemas on distributed resources which will require inference from multiple sources making it even harder to identify patient through inference attacks.
The stored pseudonyms are never shared with any of the third-party services in the whole mechanism instead a short-lived pseudonym is shared which makes caching of pseudonym corresponding to Ui ineffective.

CHAPTER - 4
DISCUSSION

4.1. RBAC & INFORMATION SELF-DETERMINATION

Role-based-access-control (RBAC) is a policy neutral access control mechanism defined around roles and privileges. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication. Its popularity is evident from the fact that many products and businesses are using it directly or indirectly.

Access control is a means by which the ability is explicitly enabled or restricted in some way.
With Role-Based access control access decisions are based on the roles that individual users have as a part of a system. It introduces us to the concept of role and permission. Permissions vary as per the roles. Permissions are assigned to roles. In terms of healthcare database it is essential to maintain RBAC for patients as it's of utmost concern who access which information of whom.

RBAC supports 3 security principles -
1. Least Privilege
2. Separation of duties
3. Data Abstraction

The degree to which data abstraction is supported should be determined by the implementation details of healthcare data.
We could implement RBAC using Oauth, XACML or SAML but Oauth stands steady against the modern technology stack unlike XACML or SAML. The Oauth 2.0 protocol is inherent to the procedure of generation of tokens and usage of GUIDs in EPGA.
While defining the RBAC model, the following conventions have been kept in mind -
If we consider the following assumptions -

1. S = Subject = A person or automated agent
2. R = Role = Job function or title which defines an authority level
3. P = Permissions = An approval of a mode of access to a resource
4. SE = Session = A mapping involving S, R and/or P
5. SA = Subject Assignment
6. PA = Permission Assignment
7. RH = Partially ordered Role Hierarchy. RH can also be written: ≥ (The notation: x ≥ y means that x inherits the permissions of y.).

Then following constraints are held true in EPGA.

1. $P\ A \subseteq P \times R$
2. $S\ A \subseteq S \times R$
3. $R\ H \subseteq R \times R$

4.2. PROBLEMS WITH BIGCHAINDB AND TENDERMINT

The system uses BigChainDB under the hood as a blockchain based storage to store data. But BigchainDB itself faces a big problem of Byzantine Fault Tolerance. In order to improve the performance of database BigChainDB is intentionally not kept Byzantine Fault Tolerant like many of the distributed systems which makes our system a bit insecure in case of an extremely well organized attack on database or in case of leak of private keys of a full node owner the database can be largely at risk.
Although recent revisions in the source have shown Tendermint being used to make it BFT.

Tendermint - Tendermint is software for securely and consistently replicating an application on many machines. By securely, we mean that Tendermint works even if up to 1/3 of machines fail in arbitrary ways. By consistently, we mean that every non-faulty machine sees the same transaction log and computes the same state. Secure and consistent replication is a fundamental problem in distributed systems; it plays a critical role in the fault tolerance of a broad range of applications, from currencies, to elections, to infrastructure orchestration, and beyond.
The ability to tolerate machines failing in arbitrary ways, including becoming malicious, is known as Byzantine fault tolerance (BFT). The theory of BFT is decades old, but software implementations have only became popular recently, due largely to the success of "blockchain technology" like Bitcoin and Ethereum. Blockchain technology is just a reformalization of BFT in a more modern setting, with emphasis on peer-to-peer networking and cryptographic authentication. The name derives from the way transactions are batched in blocks, where each block contains a cryptographic hash of the previous one, forming a chain. In practice, the blockchain data structure actually optimizes BFT design.
Tendermint consists of two chief technical components: a blockchain consensus engine and a generic application interface. The consensus engine, called Tendermint Core, ensures that the same transactions are recorded on every machine in the same order. The application interface, called the Application BlockChain Interface (ABCI), enables the transactions to be processed in any programming language. Unlike other blockchain and consensus solutions, which come pre-packaged with built in state machines (like a fancy key-value store, or a quirky scripting language), developers can use Tendermint for BFT state machine replication of applications written in whatever programming language and development environment is right for them.
Tendermint is designed to be easy-to-use, simple-to-understand, highly performant, and useful for a wide variety of distributed applications.

CHAPTER - 5
CONCLUSION

5.1. CONCLUSION

Clive Humby in 2006 said, “Data is the new oil”.

The statement might be vague in terms of practicality when considered that oil is limited and naturally occuring resource while data is artificial and is ever increasing in nature. But the common ground is held by both in terms that they both are extremely sensitive and precious. But it puts us to shame when we think about how one is kept under all supervision while other is lying naked over the networks.

Being avid open source consumers the project involved usage of many open source technologies most notably Python and React Native. In return to this the project has breeded two libraries indexed in Python Package Index (PyPI) titled **PyZIPIN** and **OpenCA**.
Not only that but the whole code has been made publicly accessible for future use for any further reference at **https://github.com/ash2shukla/VRAHAD**.

The overall emphasis on the privacy of data may seem like a secondary case for a country which is still in development phase like India but recent events have shown inclination of the world’s largest democracy towards securing the Right-To-Privacy for it’s citizens. As the paper Data Protection Framework in India dated at 2018-Nov-30th exclaimed that the heat on this topic is not settling down in upcoming future. Based on these recent events we can surely say that in upcoming time our work might help in securing the new electricity for our citizens that is Data.

CHAPTER - 6

(APPENDIX)
(RESEARCH PAPER - REAL-TIME DE-IDENTIFICATION OF HEALTHCARE DATA USING EPHEMERAL PSEUDONYMS)

International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 7, Issue 2, March-April 2018 ISSN 2278-6856

Real-time De-identification of Healthcare Data Using Ephemeral Pseudonyms

[1]Ashish Shukla, [2]Mohit Kumar Sahni, [3]Sourav Aggarwal, [4]Bipin Kumar Rai

[1] [2] [3]Student, Computer Science & Engineering Department, ABES IT, Ghaziabad
[4]Research Scholar, Banasthali University & Associate Professor, Information Technology Department, ABES IT, Ghaziabad

Abstract: ***Information explosion is radically changing our perception of the surroundings and healthcare data is at the core of it. The nature of healthcare data being extremely sensitive poses a threat of invasion of privacy of individuals if stored or exported without taking proper security measures. De-identification involves pseudonymization or anonymization of data which are methods to disassociate an individual's identity temporarily or permanently respectively. These methods can be used to provide secrecy to user's healthcare data. A commonly overlooked weakness of Pseudonymization technique is Inference attacks. This paper discusses an approach to de-identify Enterprise Healthcare Records (EHR) using chained hashing for generating short-lived pseudonyms to minimize the effect of inference attacks and also outlines a re-identification mechanism focusing on information self-determination.***

Keywords: De-identification, Electronic Healthcare Records, Pseudonymization, Inference Attack.

1. Introduction

Electronic Health Records (EHRs) provides us many advantages such as better communication between healthcare services and patients, no-need of carrying previous reports, reduced costs of treatment and also serves as a repository to retrieve data for research purpose. Healthcare data is inherently extremely sensitive by its nature. The leakage of the same can result in social as well as economic losses to the individual. Thus securing EHR is extremely important. Securing data follows two approaches namely Encryption and de-identification. Although Encryption is the conventional and most reliable way of assuring the data security it has significant drawbacks like the overhead of decrypting data for any analysis or real-life usage. An alternative approach is de-identification of data which is essentially disassociation of personal identifiers from data. It should be noted that de-identification is not a technique of securing data itself, instead, it is a technique of protecting an individual's privacy. De-identification follows two approaches Anonymization and Pseudonymization.

1.1. Anonymization - Anonymization is a de-identification technique that dis-associates all identifiers from the data. For example, creating a teaching file for radiological images illustrating a specific condition requires anonymization of the data.[1] Here the important point is that there is no requirement to be able to identify the patient later so all traces of the patient should be removed and the data is made fully anonymous by manually reviewing the files and their fields to determine which fields are required for instructional purposes and which required fields can be used for re-identification of patient. In practice, such fields are rewritten to retain useful meaning while not disclosing any private information.[1]

Anonymization has following three principles-

Let there be a relation $T(a_1,a_2 \ldots\ldots, a_d)$ for which Q_T is the set of Quasi-identifiers for relation T. where for $i = (1, \ldots, m)$ $a_i \in Q_T$ .Then,

1.1.1. k-anonymity[2] - Qt_i for $t_i \in T$ should be indistinguishable from at least k-1, $t_j \in T$ *where* $j \in (1,\ldots,d)$ *and* $j \mathrel{!}= i$. The process of enforcing k-anonymity is called k-anonymization in which T is partitioned into groups g_j *such that* $j \in (1\ldots h)$ *and* $|g_j| < k$, *here* $|x|$ *means the size of* x. tuples in g_j are made identical to the Q_T in process of k-anonymization.

1.1.2. l-diversity[2] - Only providing k-anonymity may cause inference of an individual's values in the sensitive values (SA), this is called value disclosure. To prevent value disclosure each anonymized group must contain at least *l well-represented* values. Here well-represented value means *distinct* and leads to the principle called *distinct l-diversity.* which requires each anonymized group to contain at least l distinct SA values.

1.1.3. Recursive (c, l) diversity[2] - Given parameters c,l, which are specified by data publishers, a group g_j is (c,l)-diverse when $r_1 < c \times (r_l + r_{l+1} + \ldots + r_n)$, where r_i,i $\in$ {1,...,n} is the number of times the i-th frequent SA value appears in g_j, and n is the domain size of g_j. T is (c,l)-diverse when every g_j, j = 1,...,h is (c,l)-diverse.

1.2. Pseudonymization - Pseudonymization is a de-identification technique in which we introduce a pseudonym in place of the attributes that directly or indirectly identify an individual. IHE defines it as a technique that uses *controlled replacements to allow longitudinal linking and authorized re-identification.*[1] Let there be a relation $T(a_1,a_2 \ldots\ldots, a_d)$ for which Q_T is the set of Quasi-identifiers for relation T. where for $i = (1, \ldots, m)$ $a_i \in Q_T$ then pseudonymization is essentially replacing Q_T *with* P_T *where* $P_T = (P_1, P_2 \ldots, P_m)$ Keeping another relation $P_T \rightarrow Q_T$ for re-identification.

International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 7, Issue 2, March-April 2018 ISSN 2278-6856

The definitions of de-identification techniques itself clarify that being unable to reassociate data with any individual Anonymization is not suitable for all the purposes in EHR. It is the reason why Pseudonymization is often the recommended process for providing privacy to users. Pseudonymization is also advised to be used by EU General Data Protection Regulation (GDPR) which will be enforced on May 25, 2018.
Few significant pseudonymization approaches are following -

1.2.1. Peterson's approach[6] - Robert L Peterson suggested a key-based approach to provide access control and encryption of medical information. The patient holds a Personal Key (PEK). This approach also involves assigning a static pseudonym to the individuals. There exists a Global Key (GK) which uniquely identifies the patient in the pseudonymized records when used jointly with PEK. The records are secured by encryption on database using PEK thus the entire security of information is revolving around the encryption of information. If PEK is stolen then this approach is rendered ineffective against attackers.

1.2.2. Slamanig and Stingl's Approach[7] - This approach suggests storage of User Information and Medical Data on different databases. These two are mapped with the help of some central components. Same as Peterson's approach, Slamanig's approach also suggests storing data in encrypted form and giving the encryption key to the patient. It focuses on access control as well but doesn't ensure the security of data if the data is to be shared with a 3rd party (e.g. for research purpose).
Similar approaches were suggested by Pommerening and Thielscher as well.[8] All of the approaches seem to be greatly affected by the problem of inference attacks as the used pseudonyms are persistent and eventually start to work as a unique identifier as the patient's information grows larger. Thus a need for *variable* or *ephemeral* pseudonyms arises to weaken the inference attacks.

1.3. Pseudonym Generation Techniques - Primarily we use two pseudonym generation techniques namely *Hashing* and *Tokenization*, Hashing is computationally more expensive and leaves no traceback of the information it has been generated from whereas tokenization is a method that creates a pseudonym that retains the data it originated from and requires much less computation. Although tokenization and hashing both have their respective use cases but generally tokenization increases the possibilities of inference attacks.

1.4. Real-time de-identification - Real-time de-identification refers to de-identification of data as it streams. This is a basic requirement if we are dealing with data that needs to be de-identified as it's generated and EHR falls under such category. It's hard to create a secure mechanism for such cases as it involves dealing with relations having varying attributes. To resolve this there must exist a standard API or protocol that has values in a predefined format.

1.5. Inference Attacks and Pseudonymization - Pseudonymized data is prone to inference attacks. The biggest loophole being *persistent pseudonym* usage. Inference attacks relate to data mining techniques. If an adversary can *infer* the identity associated with some pseudonymized data with high confidence then the data is said to be leaked. As pseudonymization is not a technique of encryption and rather relies on hiding the identity of individuals, it is highly liable to this attack. Statistical frequency analysis attacks are a very basic example of inference attacks. Dataset aggregation techniques are also used heavily by attackers in order to derive an inference from existing datasets.
If there is a relation $T(a_1,a_2 \ldots\ldots, a_d)$. for which Q_T is the set of Quasi-identifiers for relation T. and there exists another relation $D(d_1,d_2 \ldots\ldots, d_d)$ which contains identification information about the individuals belonging to relation T.
if for $i = (1, \ldots, m)$ $a_i \in Q_T$ *and* $a_i \in D$ then we can associate an identity based on the other attributes in the same tuple belonging to D.
One such example for EHR is evident with D_T as Voter List. If the pseudonymization was done on basis of YOB, ZIP, and Sex then for a particular state the total number of possible pseudonyms can be in the range of 10,000s.[3]
Which is significantly low and the actual identity can be derived using further inferences. This particular inference attack was exploited heavily and caused the creation of HIPAA (Health Insurance Portability and Accountability Act of 1996). Nevertheless, inference attacks are still prevalent as although the process of formation of pseudonyms has significantly changed but the underlying loopholes remain the same and the persistence in pseudonyms poses a wide threat to user's privacy.
Based on these facts it's obvious that intuitive pseudonymization methods are almost certain to fail in order to provide privacy. Successful pseudonymization requires a deep knowledge of the data.[4] It is necessary to design models keeping in mind that other datasets may be used in association with the existing records to derive identities.

2. Proposed Solution

The solution assumes that there exists an authorized body that regulates the identification information and provides a unique identifier for each resident. Let the identifier be represented by U_i , The patient is represented by $t_i \in T$ where T is set of all patient's identification records. The system consists of 3 Nodes namely Accession Node, Key Node and Data Node. Accession Node enrolls the user in Healthcare system only once. It extracts $Qt_i = (q_{1i}, q_{2i} \ldots, q_{ni})$ (*Quasi Specifiers for* t_i) from t_i and transmits it to Key Node. Key Node applies '*Ephemeral Pseudonym Generation algorithm - Initialize'* (EPGA-Init) on Qt_i which produces g_i (i^{th} *group*) and gu_i (*unique ID in* g_i) for t_i and initializes a

International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 7, Issue 2, March-April 2018 ISSN 2278-6856

report_schema for insertion of records in form of record IDs in Data Node corresponding to a H_i which is a hash of gu_i and g_i. Another relation is maintained for retrieval of gu_i through mapping of biometrics of patients at Key Node. Whole communication on the network is protected using ECDH (Elliptic Curve Diffie Hellman). There should exist a mapping of E_i (Ephemeral IDs) corresponding to each H_i, E_i will be used by healthcare services to insert and retrieve data for a patient. E_i will be generated for de-identification purposes in EPGA-D. Each E_i is only one time usable thus it gives a strong protection against caching of pseudonyms and makes it hard to infer the identity of an individual from records. To reassociate the identity of individuals with U_i the user must provide his consent by providing the gu_i.

2.1. Ephemeral Pseudonym Generation Algorithm - EPGA is divided in to three parts i.e. Initialize (EPGA-Init), De-identification (EPGA-D) and Re-identification (EPGA-R).

2.1.1. EPGA-Init - EPGA-Init Algorithm generates a global pseudonym H_i against which we will store the report_schema which will contain the Record IDs of the reports and other de-identified documents. In EPGA-Init *generalize_or_suppress* function returns the generalized form of an identifier else a null string if identifier should be suppressed. H_m is a highly collision resistant Hashing algorithm (e.g. SHA256). Kg_i stands for i^{th} group's key. The *getLast* function takes the argument as group id and returns the de-serialized object associated with that gid else returns Null if group id doesn't exist in Key Node's Database.

- **EPGA-Init(Qt_i):**

1. $gQt_i \leftarrow generalize\_or\_suppress(q_i: q_i \in Qt_i)$
2. $Kg_i \leftarrow$ '\0'
3. $Kg_i \,||\, q_i : q_i \in gQt_i$
4. $g_i \leftarrow H_m(Kg_i)$
5. $count \leftarrow getLast(g_i)$
6. $gu_i \leftarrow randomize\_count(count)$
7. $H_i \leftarrow HMAC(g_i \,||\, gu_i, key = Kg_i)$
8. return H_i , gu_i

To define *getLast* function we assume that there must exist a cQueue associated with each group id in Key Node's database which stores the counts of revoked gu_i corresponding to g_i to avoid overflow in group unique ID's counts. *randomize_count* takes *count* as seed and maps the count to another number within a defined prime number's range. It only introduces randomness in generated group unique IDs.

- **getLast(g_i):**

1. retrieve g_i row from database.
2. if g_i doesn't exist in database:
   2.a. $g_i.count = 0$
   2.b. return 0
3. $cQueue \leftarrow deserialize(g_i.cQueue)$
4. if cQueue is null:
   4.a. return $g_i.count+1$
5. else:
   5.a. $count \leftarrow dequeue(cQueue)$
   5.b. serialize(cQueue)
   5.c. update g_i.cQueue in database
   5.d. return count

2.1.2. EPGA-D - EPGA-D Algorithm de-identifies the streaming data and fulfills the purpose of real-time de-identification of streaming data. If the data is being produced by a producer on a stream processing platform e.g. Kafka in a predefined format e.g. FHIR (Fast Healthcare Interoperability Resources) then we can apply EPGA-D on producer-end if the producer is reliable else on consumer-end on Data Node to de-identify data in real-time. The de-identification of a patient report is partially influenced by *safe harbor method*[5] which suggests suppression of 18 identifiers like Names, Locations, Dates directly relating to an individual, Telephone numbers, Fax numbers etc. The key difference being that EPGA-D assigns a short-lived pseudonym as the report's ID called Ephemeral ID (E_i) along with suppression of identifiers suggested in safe harbor method. The Ephemeral ID is generated by user's consent on report producer's end after providing gu_i. Upon receiving the pseudonymized data with E_i on Data Node, the Data Node generates a random identifier RH_i and replaces E_i with RH_i. RH_i is updated in the report_schema corresponding to the patient's H_i who generated the E_i.
In order to generate E_i patient can send the request for the generation of E_i to Key Node through an authenticated medium by providing his gu_i and U_i.

- **createEi(gu_i , U_i):**

1. retrieve Qt_i from identification body through U_i.
2. $gQt_i \leftarrow generalize\_or\_suppress(q_i: q_i \in Qt_i)$
3. $g_i \leftarrow H_m(\ concat(q_i) : q_i \in gQt_i)$
4. creates a random identifier E_i and associate it with the H_i.
5. return E_i

We further subdivide the EPGA-D algorithm into two parts i.e. @Producer and @Consumer where Producer is the segment that should be used on the stream's end which produces the de-identified report and Consumer is the stream's end which receives the de-identified report i.e. Data Node.

@Producer

- **EPGA-D(Report):**

1. Request patient to generate E_i.
2. $E_i \leftarrow createEi(gu_i , U_i)$
3. $gReport \leftarrow generalize\_or\_suppress(Report)$
4. $gReport.id \leftarrow E_i$
5. Stream gReport on data pipeline.

International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org, editorijettcs@gmail.com
Volume 7, Issue 2, March-April 2018 ISSN 2278-6856

@Consumer

- **EPGA-D(Report):**

1. *create random identifier RH_i.*
2. *$E_i \leftarrow Report.id$*
3. *Request H_i corresponding to E_i from Key Node.*
4. *On receiving H_i request Key Node deletes E_i from the map and returns corresponding H_i.*
5. *$Report.id \leftarrow RH_i$*
6. *Save RH_i in report_schema corresponding to H_i*

2.1.3. EPGA-R - EPGA-R Algorithm re-associates the identity of an individual with a Report with the explicit consent of the patient. The patient generates a short-lived one-time usable Ephemeral Group Unique ID (Egu_i) by providing his gu_i, U_i and Lifetime of Egu_i. In case the patient does not provide the lifetime of Egu_i a default timeout must be set up to prevent misuse of Egu_i through malevolent attempts.

In order to generate Egu_i patient can send the request for the generation of E_i to Key Node through an authenticated medium by providing his gu_i , U_i *and* optionally the time to live *(ttl)* for Egu_i.

- **createEgui(gu_i , U_i, ttl = default_time):**

1. *retrieve Qt_i from identification body through U_i.*
2. *$gQt_i \leftarrow generalize\_or\_suppress(q_i: q_i \in Qt_i)$*
3. *$g_i \leftarrow H_m(concat(q_i) : q_i \in gQt_i)$*
4. *creates a random identifier Egu_i and associate it with the H_i.*
5. *Set ttl of Egui.*
6. *return Egu_i*

Let us assume there exists a 'Service' which wants to re-identify the patient.

- **EPGA-R(gu_i , U_i):**

1. *Service requests patient to generate E_i.*
2. *$Egu_i \leftarrow createEgui(gu_i , U_i, optional\_ttl)$*
3. *Service requests patient to provide U_i.*
4. *Service sends U_i and Egu_i to DataNode.*
5. *Data Node requests Key Node to return H_i corresponding to Egu_i and U_i .*
6. *KeyNode returns the H_i to Data Node and deletes Egu_i.*
7. *DataNode returns requested data associated with H_i to the Service.*

3. Conclusion

EPGA can be used to implement real-time de-identification of healthcare data. It provides the patient information self-determination as EPGA-D and EPGA-R both revolve around the group unique ID gu_i which is exclusively known to user. gu_i works as a proof-of-consent for the algorithm.

EPGA-D provides a fairly complex relation between report ID and H_i which makes it hard to find a straight relation between reports and patient pseudonyms making inference attacks less effective. To reduce the effect of inference attacks even more we can split report_schemas on distributed resources which will require inference from multiple sources making it even harder to identify patient through inference attacks.

The stored pseudonyms are never shared with any of the third-party services in the whole mechanism instead a short-lived pseudonym is shared which makes caching of pseudonym corresponding to U_i ineffective.

4. Future Work

Based on the algorithm we can create an architecture for scalable EHR using appropriate messaging queues and stream processing platforms. Although the proposed solution provides a robust mechanism for de-identification of data but it lacks the safe storage of data. An adversary's malevolent attempt can be aimed at destroying the integrity of the data which would render the de-identified data useless for the patient. Perhaps a blockchain based immutable storage can address this problem but the proposed solution lacks it.

Appendix

T - Relation containing all patients.
D - Relation containing de-identification information of all patients.
P - Relation containing pseudonyms for all patients.
t_i - i^{th} patient belonging to relation T
U_i - Basic identity information of t_i.
g_i - Group ID of t_i.
gu_i - Unique ID in group for t_i.
Qt_i - List of Quasi Specifiers for t_i.
gQt_i - Generalized or suppressed list of Quasi Specifiers for t_i.
Egu_i - Ephemeral Unique ID in group for t_i.
H_i - Globally Unique ID for t_i to map Report IDs.
RH_i - Unique Global ID for i^{th} report.
H_m - Highly collision resistant Hashing algorithm
|| - Concatenation symbol.
E_i - Ephemeral ID for i^{th} report.
HMAC - Hash based Message Authentication Coding function.
Kg_i - Key for creating H_i through HMAC for i^{th} patient.

References

[1] IHE IT Infrastructure Technical Committee, Integrating the healthcare enterprise (IHE IT Infrastructure Book), June 6,2014, pp. 170.

[2] Aris Gkoulalas-Divanis Grigorios Loukides, Overview of patient Data Anonymization, September 13, 2012, pp. 9-11.

[3] Latanya Sweeney, Only You, Your Doctor, and Many Others May Know, Sept. 29, 2015.

[4] Phil Factor, Pseudonymization and the Inference Attack (Redgate Hub), August 01, 2017.

[5] Guidance Regarding Methods for De-identification of Protected Health Information in Accordance with the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule , September 4, 2012.

[6] Peterson, R.L., Encryption system for allowing immediate universal access to medical records while maintaining complete patient control over privacy. US Patent Application Publication, No.: US 2003/0074564 A1 , 2003.

[7] Daniel slamanig, Christian stingl , 'Privacy aspect of e-health' the 3rd international conference on availability, reliability and security, IEEE computer society, 2008.

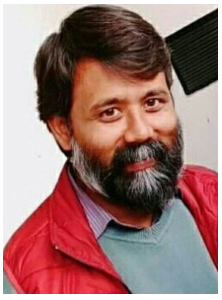
[8] Bipin Kumar Rai, Dr. A.K. Srivastava, Pseudonymization Techniques for Providing Privacy and Security in EHR, IJETTCS, July, 22, 2017.

AUTHORS

Ashish Shukla is an undergraduate Computer Science & Engineering student pursuing B.Tech at ABES IT, Ghaziabad. His primary area of interest is Information Security and Data Sciences. (ash2shukla@gmail.com)

Mohit Kumar Sahni is an undergraduate Computer Science & Engineering student pursuing B.Tech at ABES IT, Ghaziabad. His primary area of interest is Big Data and Data Analytics. (mohitkumarsahni@gmail.com)

Sourav Aggarwal is an undergraduate Computer Science & Engineering student pursuing B.Tech at ABES IT, Ghaziabad. His primary area of interest is Deep Learning and Data Science. (srvaggarwal96@gmail.com)



Bipin Kumar Rai, received the B.Tech(CSE) from UPTU (BIT Muzaffarnagar) Lucknow, UP and M.Tech(CSE) from RGPV Bhopal, (SSSIST, Sehore) MP in 2004 and 2009, respectively. During 2004-2006 & 2008-2014 he taught in different engineering colleges. He is with ABES IT as Associate Professor now. His primary area of interest is Information Security.(*bipinkrai@gmail.com*)

REFERENCES

1. IHE IT Infrastructure Technical Committee, Integrating the healthcare enterprise (IHE IT Infrastructure Book), June 6,2014, pp. 170.

2. Aris Gkoulalas-Divanis Grigorios Loukides, Overview of patient Data Anonymization, September 13, 2012, pp. 9-11.

3. Latanya Sweeney, Only You, Your Doctor, and Many Others May Know, Sept. 29, 2015.

4. Phil Factor, Pseudonymization and the Inference Attack (Redgate Hub), August 01, 2017.

5. Guidance Regarding Methods for De-identification of Protected Health Information in Accordance with the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule , September 4, 2012.

6. Peterson, R.L., Encryption system for allowing immediate universal access to medical records while maintaining complete patient control over privacy. US Patent Application Publication, No.: US 2003/0074564 A1 , 2003.

7. Daniel slamanig, Christian stingl , 'Privacy aspect of e-health' the 3rd international conference on availability, reliability and security, IEEE computer society, 2008.

8. Bipin Kumar Rai, Dr. A.K. Srivastava, Pseudonymization Techniques for Providing Privacy and Security in EHR, IJETTCS, July, 22, 2017.

9. Python Documentation, Python Wikia Biggener's Guide.

10. Advantages and Disadvantages of Python Programming Language, Minfire, Medium, 24 Aug 2017

11. Devopedia Django.

12. Advantages and Disadvantages of Django, Steven Hansen, Hackernoon, May 23 2017.

13. 4 Benefits of Apache Kafka in lieu of AMQP or JMS, Christy Wilson, 27 May 2017, SyncSort.

14. What is BigChainDB? Bruce Pon, Ocean Protocol, BigChainDB, 14 Feb 2016.

15. What is React Native?, Bonnie Eisenman, React-Native-Docs, 13 Jan 2013.