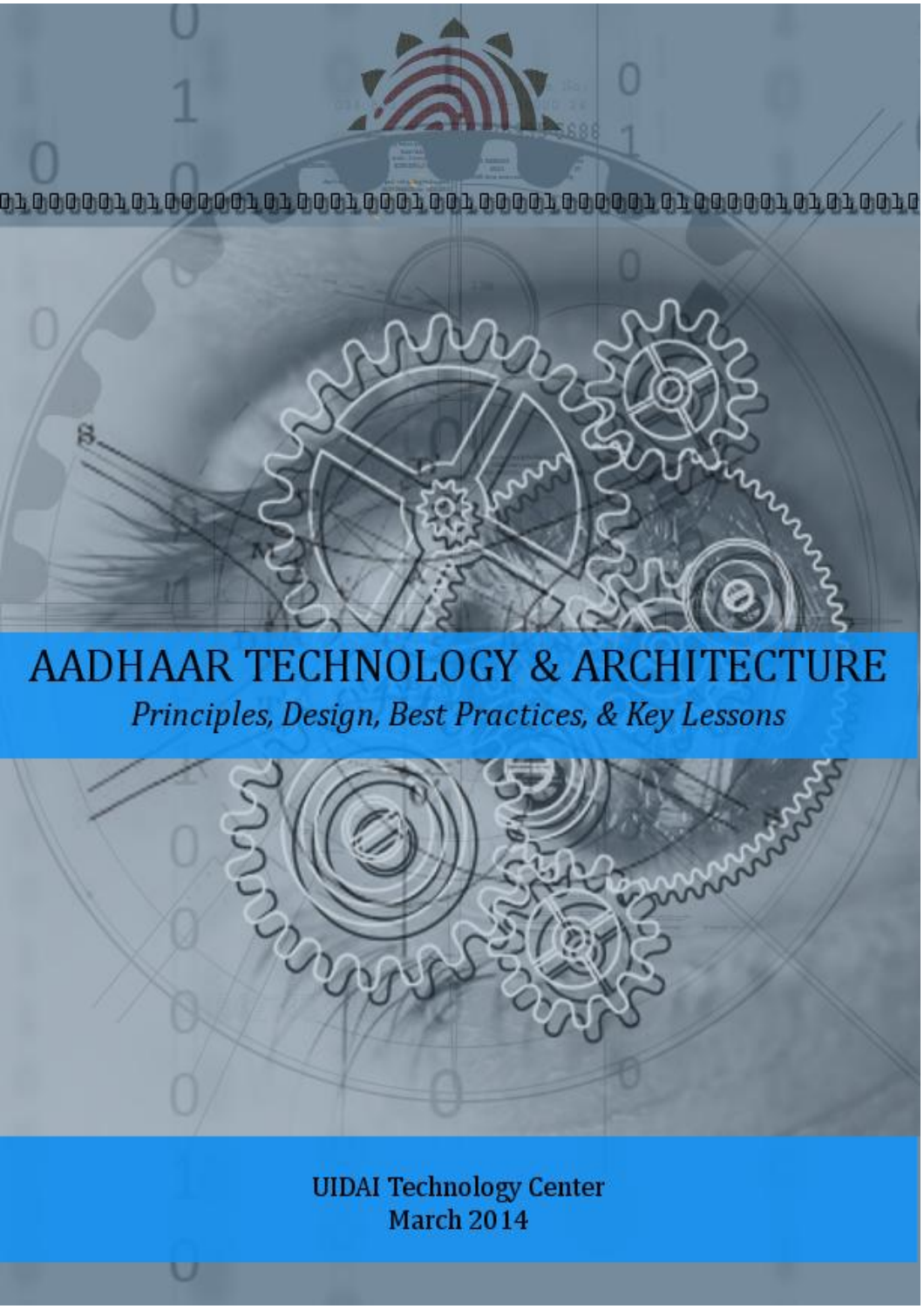


AADHAAR TECHNOLOGY & ARCHITECTURE

Principles, Design, Best Practices, & Key Lessons

UIDAI Technology Center

March 2014

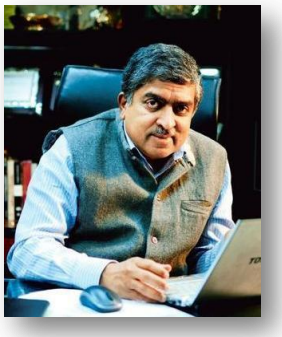
AADHAAR TECHNOLOGY & ARCHITECTURE

Principles, Design, Best Practices, & Key Lessons

MARCH 2014

UNIQUE IDENTIFICATION AUTHORITY OF INDIA
PLANNING COMMISSION, GOVERNMENT OF INDIA
JEEVAN BHARATI BUILDING, CONNAUGHT CIRCUS, NEW DELHI – 110001

Message from Chairman



UIDAI has the vision of empowering every resident of India with a unique identity and providing a digital platform to authenticate anytime anywhere. The Aadhaar system is built on a sound strategy and a strong technology backbone and has evolved into a vital digital infrastructure. Meticulous planning and execution enabled the program to be launched ahead of plan in Sept 2010 and reach the kind of scale that was never achieved in any biometric identity system in the world.

UIDAI created an ecosystem of partners for the various key components of the project and integrated them all onto a common technology backbone using open standards. Aadhaar technology system was able to succeed due to its core principles – openness, vendor-neutrality, security, and data analytics. In the absence of digital privacy laws, UIDAI took it upon itself to implement rigorous standards and measures to ensure data privacy and security. Apart from generating 60 crore (600million) Aadhaars in 4 years through this approach, the Aadhaar platform is now integrated into the financial systems of NPCI and banks, taking India towards the goal of total financial inclusion.

Documenting of the design and implementation of the Aadhaar technology, which evolved over a period of four years, was as onerous as building the system itself. The UIDAI Technology Centre has done a commendable job in compiling white papers on technology strategy, application features, and architecture. These documents place the Aadhaar system on a firm foundation and also serve as the beacons for many e-Governance projects in India and across the world.

My hearty congratulations to the UIDAI technology team!

Nandan Nilekani
Chairman, UIDAI

Message from Director General

Aadhaar project, under the Planning Commission, Government of India, is an initiative to provide a unique identification number to every resident that can be leveraged by the residents to access various services and benefits. Uniqueness of Aadhaar identity allows elimination of fake and duplicate accounts, and online authentication provides a mechanism for paperless, electronic and instantaneous verification of identity, anytime anywhere. Aadhaar platform can be utilized by various Governmental, public, and private sector agencies to efficiently deliver services to residents.

By its very nature, the Aadhaar system needed a strong technology foundation. Appropriately, the Technology Centre was the first unit of UIDAI to start functioning in 2009 in Bangalore. UIDAI technology team was able to develop the required applications and successfully generate the first Aadhaar number within a year of its formation. The architecture was rigorously tested and validated through a series of 'Proof of Concept' studies. Aadhaar generation was commenced in August 2010 and the first Aadhaar letter was formally handed over to a resident at Nandurbar in Maharashtra on 29 September 2010. Subsequently Authentication and e-KYC services were also launched.

In the course of attaining the milestone of 60 crore (600 million), the Aadhaar technology backend has become the largest biometric identity repository in the world and the first to provide an online, anytime anywhere, multi-factor authentication service. A strong technology foundation based on open architecture enabled the rapid evolution of the Aadhaar system. It was important to document all aspects of Aadhaar technology and make it available in public domain. The three white papers published by the UIDAI Technology Centre fulfil this need.

I sincerely appreciate the efforts of our technology team in publishing these.

Vijay Madan
Director General, UIDAI

From the Editor's Desk

'Aadhaar' is undoubtedly one of the most important Projects rolled out by Government of India. Ambitious, one-of-a-kind and a game-changer, 'Aadhaar' is on the path of delivery to every Indian resident, a 'national identity' and triggering thereby the much desired governance system based on social inclusion, transparency & accountability.

Considering India's population of 121 crores, it is obviously a highly ambitious project. Any Project of this magnitude prospers and grows based on the contributions from the people associated with it. Unique Identification Authority of India (UIDAI) has been blessed in this regard. Many outstanding people from both Private and Government sectors, spanning the domains of - Administration, Biometrics, Project Management, Law, Technology and Finance, to name a few - have come together and dedicated their talent, time and knowledge, besides the much needed passion & commitment.

UIDAI has also been led by competent people like Shri Nandan Nilekani, as its Chairman, Shri Ram Sewak Sharma, as its first Director General and Dr Vijay Madan, who succeeded him a year back. Shri Nilekani has provided both visionary and charismatic leadership to the Organisation as its Founding Chairman. With his experience of building one of India's internationally recognised I.T. Companies; besides his association with both State & Union Governments in different advisory capacities, he was an appropriate choice to establish and lead an organisation like UIDAI that called for an understanding of technology and a commitment to social inclusion. The yeoman services rendered by the first Director General, Shri R.S Sharma are recalled with fondness. A man of boundless energy and dedication, Shri Sharma with his attention to detail and sharp technical knowledge, ensured that the Project got the right kick-start and maintained momentum in its formative years. Dr Madan has been the perfect foil to the Chairman besides being an able successor as the current Director General. He has been able to expand the horizon of the Organisation by steering it to the next growth trajectory for quicker delivery of Aadhaar and its deployment for people-centric Applications. His focus also has

been on knitting together different components into a single and synergetic unit for this purpose.

UIDAI has always believed in documentation and sharing of information with public. Technology & processes that characterise UIDAI needed to be brought within the public domain as a matter of transparency, as also to elicit debate and discourse. Towards this end, the White Papers have been in the making for a while. I acknowledge the contributions of Shri Srikanth Nadhamuni, Dr Pramod Varma, Shri Sanjay Jain and Dr Vivek Raghavan in writing the papers. Our ADGs at the Tech Centre too have been actively involved in writing and reviewing them. The contributions of Shri Rajendra Kumar, Shri Sudhir Narayana, Shri Venkat Rao and Shri Anup Kumar are appreciated in particular.

The writers and reviewers have not lost sight of the need for a simple & straight language while putting together the highly technical and procedurally rigorous content of the Project. They have done a commendable job. Further, these documents are a manifestation of an intensely cerebral and immensely taxing effort put in by a band of dedicated domain experts who worked with the UIDAI's Tech Centre at different stages of its evolution. There are many, but some who definitely need a mention are Srikanth Nadhamuni, Raj Mashruwala, Pramod Varma, Sanjay Jain, Vivek Raghavan, and Jagdish Babu. Our colleagues including Dy. Directors General, Asst. Directors General and others serving UIDAI's cause at the Headquarters, Regional Offices and Technology Centre have contributed in multiple ways to the evolution of technology, processes & procedures and the compilation of this document. I beg pardon for not mentioning them by name. All of them are gratefully acknowledged.

It is a matter of pride and pleasure that these are getting published as the UIDAI reaches the hallmark of 60 crore Aadhaars well ahead of the targeted date and gears up to provide identification to the other half of the nation.

Ashok Dalwai
Deputy Director General, UIDAI
Technology Centre, Bengaluru

Table of Contents

Table of Abbreviations

ABIS	Automated Biometric Identification System
API	Application Programming Interface
ASA	Authentication Service Agency
AUA	Authentication User Agency
BFD	Best Finger Detection
BI	Business Intelligence
BPL	Below Poverty Line
BSP	Biometric Service Provider
CIDR	Central Identity Data Repository
DDSVP	Demographic Data Standards & Verification Process
DoB	Date of Birth
EA	Enrolment Agency
ICDS	Integrated Child Development Services Scheme
JSY	Janani Suraksha Yojana
KYC	Know Your Customer
MNREGS	Mahatma Gandhi National Rural Employment Guarantee Scheme
OTP	One Time Pin
PDS	Public Distribution System
PID	Personal Identity Data
PII	Personal Identity Information (Personally Identifiable Information)
PoA	Proof of Address
PoI	Proof of Identity
RSBY	Rashtriya Swasthya Bima Yojna
SSA	Sarva Shiksha Abhiyan
UIDAI	Unique Identification Authority of India

Table of Figures

Executive Summary

The Unique Identification Authority of India (UIDAI) was established in January 2009, as an attached office to the Planning Commission. The purpose of the UIDAI is to issue a Unique Identification number (Aadhaar number) to every Indian resident that is (a) robust enough to eliminate duplicate and fake identities, and (b) can be verified and authenticated in an easy, electronic, cost-effective way. The UIDAI's approach will keep in mind the learning from the government's previous efforts at issuing identity.

Aadhaar system is built purely as an "*Identity Platform*" that other applications, Government and private, can take advantage of. A sound strategy [1] and a strong technology backbone enabled the program to be launched ahead of plan in September 2010 and reach the kind of scale that was never achieved in any biometric identity systems across the world. Within 4 years since launch, Aadhaar system has grown in capability and more than 600 million Aadhaar numbers have been issued so far using the system [2].

Entire technology architecture behind Aadhaar is based on principles of openness, linear scalability, strong security, and most importantly vendor neutrality. Aadhaar technology backbone is built using the following principles:

- **Open architecture** – Building Aadhaar system with true openness meant use of open standards to ensure interoperability; platform approach with open APIs to allow the ecosystem to build on top of Aadhaar APIs; vendor neutrality across the application components using open and standard interfaces; and identity system designed to work with any device, any form factor, and any network.
- **Design for scale** – Aadhaar system is expected issue more than 1.2 billion identities and will continue to grow as the resident population expands. Since every new enrolment requires biometric de-duplication across the entire system, every component needs to scale to very large volumes. This

meant that system must handle hundreds of millions of transactions across billions of records doing hundreds of trillions of biometric matches every day! In addition all online services such as Aadhaar authentication, e-KYC service, and update service must work with high availability and sub-second performance. Network and data centre load balancing and multi-location distributed architecture for horizontal scale are critical to such massive scalability.

- **Data Security** – Security and privacy of data within Aadhaar system has been foundational. UIDAI has taken several measures to ensure security of resident data from the time it is captured all the way to how it is stored within CIDR. Usage of 2048-bit PKI encryption and tamper detection using HMAC ensures no one can decrypt and misuse the data, even if they are in possession of enrolment packet. Resident data and raw biometrics is always kept encrypted even within UIDAI data centres. In addition, entire Business Intelligence (BI) sub-system anonymizes all PII to ensure resident personal data is protected across all system components.

All application components are built using open source components and open standards. Aadhaar software currently runs across two of the data centres within India managed by UIDAI and handles 1 million enrolments a day and at the peak doing about 600 trillion biometric matches a day. Current system already has about 4 PB (4000 Terabytes) of raw data and continues grow as new enrolments come in. Aadhaar Authentication service is built to handle 100 million authentications a day across both the data centres in an active-active fashion and is benchmarked to provide sub-second response time. Central to Aadhaar system is its biometric sub-system that performs de-duplication and authentication in an accurate way [3] [4].

This document is meant to provide architectural patterns, design philosophy, and specific lessons from Aadhaar project for software architects building large scale systems, especially for those in various e-Governance projects. By sharing the learning and architecture details, Aadhaar team hopes to help software architects in making the right decision and apply similar architectural patterns within their application.

This is a technical document and is targeted at software architects building open, scalable systems especially in the area of e-Governance. Readers are encouraged to read the following documents to get a complete understanding of Aadhaar system.

1. UIDAI Strategy Paper [1]
2. Aadhaar Technology Strategy [5]
3. Aadhaar Product Document [6]

1 Introduction to Aadhaar

In India, inability to prove identity is one of the biggest barriers preventing the poor from accessing benefits and subsidies. Public as well as private sector agencies across the country require proof of identity before providing individuals with services. But till date, there remains no nationally accepted, verifiable identity number covering all population that both residents and agencies can use with ease and confidence.

As a result, every time individuals try to access a benefit or service, they must undergo a full cycle of identity verification. Different service providers also have different requirements in the documents they demand, the forms that require filling out, and the information they collect on the individual. Such duplication of efforts due to '*identity silos*' increase overall cost of identity verification and cause inconvenience. This is especially hard for India's poor and underprivileged residents, who usually lack documentation and find it difficult to avail various services and benefits.

There are clearly immense benefits from a mechanism that uniquely identifies a person and ensures instant identity verification. The need to prove one's identity only once will bring down transaction costs. A clear identity number can transform the delivery of social welfare programs by making them more inclusive of those communities now cut off from such benefits due to their lack of identification. It also enables the government to shift from indirect to direct benefit delivery by directly reaching out to the intended beneficiaries. A single universal identity number is also useful in eliminating fraud and duplicate identities, since individuals can no longer be able to represent themselves differently to different agencies. This results in significant savings to the state exchequer.

The Unique Identification Authority of India (UIDAI) was established in January 2009, as an attached office to the Planning Commission. The purpose of the UIDAI is to issue a Unique Identification number (Aadhaar number) to every Indian resident that is (a) robust enough to eliminate duplicate and fake identities, and (b) can be verified and authenticated in an easy, electronic, cost-effective way.

1.1 Aadhaar Strategy

UIDAI only provides identity – The UIDAI's purview is limited to the issuance of unique identification (Aadhaar) numbers linked to a person's demographic and biometric information. The UID number (Aadhaar number) only guarantees identity, not rights, benefits or entitlements.

Aadhaar number proves identity, not citizenship – As per UIDAI's mandate [7], all legal residents (anyone legally residing in India) in the country can be issued Aadhaar number. Possession of Aadhaar number is merely a proof of identity and does not confer citizenship.

A pro-poor approach – The UIDAI envisions full enrolment of residents, with a focus on enrolling India's poor and underprivileged communities. The Registrars (such as State Governments) that the UIDAI partners with helps bring large numbers of the poor and underprivileged into the Aadhaar system. Providing token-less (no cards or other physical tokens), online, anytime anywhere authentication improves service delivery to the poor.

Enrolment of residents with proper verification – Existing identity databases in India are fraught with problems of fraud due to duplicate/ghost beneficiaries. To prevent this from seeping into the UIDAI database, the UIDAI enrols residents into its database with proper verification of their demographic information based on Shri. N. Vittal headed DDSVP committee recommendations [8]. This ensures that the data collected is clean from the start of the program. However, much of the poor and underserved population lack identity documents and Aadhaar may be the first form of identification they have access to. The introducer model proposed by

the DDSVP committee allows UIDAI to ensure that the procedures do not become a barrier for enrolling the poor.

A partnership model – UIDAI approach leverages the existing infrastructure of government and private agencies across India. The UIDAI is the regulatory authority managing a Central Identity Data Repository (CIDR), which will issue Aadhaar numbers, update resident information, and authenticate the identity of residents as required. UIDAI partners with agencies such as central and state departments who are the 'Registrars' for the UIDAI. Registrars conduct the enrolment camps using UIDAI software and procedures, upload the encrypted enrolment data to the CIDR to de-duplicate resident information, and help seed the Aadhaar number into their beneficiary databases.

Enrolment is not mandated – Aadhaar strategy uses a demand-driven model, where the benefits and services that are linked to the Aadhaar number ensure demand for the number. This will not however, preclude governments or Registrars from mandating enrolment.

UIDAI issues a number, not a card – The UIDAI's role is limited to issuing the number which is communicated to the resident through a letter. This number may be printed on the document/card that is issued by various usage agencies.

Aadhaar number does not contain any intelligence – Embedding personal data into identity numbers makes them susceptible to fraud and discrimination. Aadhaar number is a random number making it purely a national identity that can be used across the country.

UIDAI only collects minimal information – Aadhaar enrolment only seeks the following demographic and biometric information:

1. Name
2. Date of birth (or Age)
3. Gender
4. Address
5. Mobile Number and Email (optional)

6. Ten fingerprints, two iris scans, and photograph
7. For children under five years old, Aadhaar number and name of the guardian (Father/Mother/Guardian)

Process to ensure no duplicates – Registrars send the applicant's encrypted data packet to the UIDAI data centres for de-duplication. Aadhaar enrolment system performs a search on key demographic fields and on the biometrics for each new enrolment, to ensure uniqueness.

Process to keep data up to date – Incentives in the Aadhaar system are aligned towards a self-cleaning mechanism. The existing patchwork of multiple databases in India gives individuals the incentive to provide different personal information to different agencies. Since de-duplication in the Aadhaar system ensures that residents have only one chance to be in the database, individuals are incentivized to provide accurate data. This incentive becomes especially powerful as benefits and entitlements are linked to the Aadhaar number. Regular usage of identity across many services naturally incentivizes the resident to keep Aadhaar system up to date.

Online authentication – UIDAI offers a strong form of online authentication. When residents wanting to avail a service require identity/address verification, agencies can compare demographic and biometric information of the resident with the record stored in the central database.

Explicit resident consented e-KYC – A balance between 'privacy and purpose' is critical to ensure convenience of online identity is balanced with the requirement to protect resident identity data. Aadhaar authentication only responds with a 'Yes' or 'No' response and no resident data is sent back. Aadhaar e-KYC service allows resident to authorize UIDAI to share electronic version of their Aadhaar letter. For every Aadhaar e-KYC request, only after successful resident authentication, demographic and photo data is shared in electronic format (via biometric/OTP authentication resident explicitly authorizes UIDAI to share electronic version of Aadhaar letter instead of sharing physical photocopies). Resident authorization is NOT used for multiple e-KYC transactions, instead, every time agencies require

electronic version of Aadhaar letter data for KYC purposes, resident must authorize the agency.

Technology undergirds the UIDAI system – Technology systems have a major role across the UIDAI infrastructure. Large scale biometric de-duplication, online authentication, data security, analytics, etc require well designed, secure, and scalable systems.

1.2 Aadhaar Value proposition

For residents – Aadhaar system provides a single source of identity verification across the country for its entire population. Once residents enrol, they can use the number multiple times using electronic means. This eliminates the hassle of repeatedly providing supporting identity documents each time a resident wishes to access services such as opening a bank account, obtaining driving license, etc. By providing a clear proof of identity, Aadhaar system also facilitates entry for poor and underprivileged residents into the formal banking system providing large scale financial inclusion and provides the opportunity to avail services offered by the government and the private sector. Aadhaar system enables mobility to millions of people who migrate from one part of the country to another.

For Governments – The UIDAI provides Aadhaar to residents only after de-duplicating their records against entire database. Eliminating duplication under various schemes is expected to save substantial money for the government exchequer. It also provides governments with accurate data on beneficiaries, enable direct benefit programs, and allow government departments to coordinate and optimize various schemes.

For Service Agencies – Uniqueness characteristic of Aadhaar number helps agencies such as banks, telecom companies, insurance companies, etc clean out duplicates from their databases, enabling significant efficiencies and cost savings. For agencies focused on cost, Aadhaar online authentication and e-KYC services greatly help lower KYC costs. For agencies focused on social goals, a reliable identification number enables them to broaden their reach via easy identity

verification. The strong authentication that the Aadhaar number offers can improve ease of delivering services in a convenient fashion, leading to better customer satisfaction.

1.2.1 Aadhaar in service delivery

The key goal of Aadhaar program is to provide an identity infrastructure for delivery of various social welfare programs and for effective targeting of these services. While enabling better government welfare delivery is the prime focus of Aadhaar, it can also be utilized by enterprises and service providers such as banks, telecom companies, and others for improving their service delivery.

The potential of Aadhaar can be realized only upon use of the infrastructure as an ID proof and as a unique key by various state departments, central ministries, PSU's, and private sector entities to provide service delivery to residents in an integrated fashion. There are many benefits associated with such integration for the various stakeholders that range from better compliance management to significant savings in leakages and increased efficiency and accountability in service delivery.

1.2.2 Aadhaar Usage Types

Aadhaar and identity authentication can be used by the service delivery provider mainly for the following 3 broad usage types.

1.2.2.1 Establishing Proof of Presence

1. **Confirming Beneficiary** – Various social sector programs, where beneficiaries need to be confirmed before delivery of the service, are expected to be the most common users of the authentication service. Examples of applications include subsidized food and kerosene delivery to PDS beneficiaries, health service delivery to RSBY beneficiaries, registering job applications by MNREGS beneficiaries, etc.
2. **Attendance and Muster Rolls** – Another key usage of authentication is attendance tracking for programs such as SSA (for students and teachers),

MNREGS where wages/outlay is linked to actual number of days the beneficiary reports for the program, etc. Similarly, pension system requiring yearly verification may use Aadhaar biometric authentication the doorstep of beneficiary using a handheld thus providing convenience to older residents.

3. **Financial Transactions** – Examples include banks that authenticate a customer using Aadhaar as well as bank-related identity information (account number or user id along with password/OTP, etc.) before enabling banking transactions such as funds transfer, funds withdrawal, etc. Several banks and Department of Post have started offering these services using Aadhaar enabled Micro-ATMs in villages and remote locations.

1.2.2.2 Establishing Know-Your-Customer (KYC) Credentials

1. **KYC for various services** – Identity and address verification is a key requirement for enrolling a new customer or opening a new account for an individual. Examples are the issuance of a new PAN card, passport, telephone connection, bank account, or an Internet service account for an online business. The service provider in all such cases can verify applicant identity and address using Aadhaar authentication and e-KYC online services instead of asking for paper copies of identity and address documents. This secure, electronic, paperless KYC process is expected to substantially reduce the cost of KYC and provide convenience to customers.
2. **General Proof of Identity (PoI)** – General use of identity document is commonplace in many areas such as entry to airports, hotels, train travel, offices, education institutions, etc. Various Internet, social networking and e-commerce related websites also use identity verification for offering their services. Online authentication and e-KYC services of Aadhaar may be used in many of these systems in an electronic, paperless mode to make entire process secure, convenient, and fast.
3. **Demographic Data verification** – Customer demographic data in service delivery databases may also be verified using Aadhaar authentication to clean and unify customer master database. For example, a bank may allow customers to simply go on their website and provide new address which can be instantly verified against Aadhaar system thus avoiding customer

having to provide paper proofs every time. Similarly, systems requiring age verification (e.g. senior citizen discounts) can easily take advantage of Aadhaar demographic authentication to allow senior residents to obtain discounts without having to provide paper proofs.

1.2.2.3 As a Unifier for Resident-centric Information

Aadhaar number may also be used as a common, unique identifier to clean up and link customer/beneficiary accounts. Applications of these can be:

1. State view of residents across schemes e.g. number of schemes accessed by resident; potential linkage of JSY, ICDS, and SSA to track health and education for every child.
2. Creation of national health care platform and portable patient records systems.
3. Credit bureaus for customer rating information.
4. National skills registry and enable individuals working in informal sectors (drivers, maids, electricians, plumbers, etc) to carry their experience as they move from one job to another and from one place to another.
5. Large entities that need to implement single customer view across services provided such as banks, insurance companies.

For details, refer to Aadhaar Enabled Service Delivery White Paper [9].

1.3 Aadhaar Design Considerations

Aadhaar system captures demographic data along with biometrics from residents as part of enrolment process. This data is used to de-duplicate against existing database and if uniqueness is established, a new Aadhaar number is assigned to that enrolment application. There were several fundamental decisions that were taken when deciding the details of data elements within Aadhaar system.

1.3.1 Minimalistic Approach to Data

Purpose of Aadhaar is only to provide a biometric attached identity that is verifiable online to a billion people. For success of such as large program, it was critical to keep the whole system simple and purpose driven. Since this touches everyone in the country once, it was possible to easily collect some more information regarding residents (such as marital status, place of birth, religion, caste, BPL status, etc).

But, since UIDAI's purpose was only to provide unique identity to an individual, DDSVP committee [8] took a firm decision to keep it minimal just enough to establish individual identity. Inclusion being one of the key goals of Aadhaar, it was critical that data be minimized to avoid elimination of people who may not be able to provide such details. Hence it was decided that Aadhaar system should capture only four key demographic elements (name, gender, date of birth, and address) with couple of additional optional attributes (mobile, email) in addition to biometrics data which is core to establishing uniqueness of an individual.

1.3.2 Federated Model & One Way Linkage

While the UIDAI has the largest repository of identity data, one must resist the temptation to add other resident attributes, especially links to other identities that resident may have, to this central system. For example, it was decided not to capture and link existing identities such as Passport number or Driver's License number or Income Tax number (PAN), etc within Aadhaar. Those application systems may add Aadhaar number within their database and link it "one way" to Aadhaar rather than other way around.

That means that Aadhaar system has no knowledge of any applications, transactions, or domain specific identities within its database. This is designed so that information about the resident (transaction history of a resident) is not centralized into one system and is kept in distributed fashion. This federated model makes collation and cross domain analysis of resident transactions difficult since it requires massive effort of bringing many federated databases together to

derive knowledge of a resident. Any such transfer or collation of data across these distributed information silos should follow due course of law.

Fundamentally, Aadhaar is designed as a "*root*" identity which is "*unique*" to an individual and is "*valid for life*" whereas all domain specific identities (Driver's License number, Income Tax number, etc) all other identities may be thought of as "derived" identities which are issued for specific purposes.

1.3.3 Designing for Inclusion

One of the key goals of Aadhaar system is inclusion. When the system needs to scale to a billion people with diverse cultural, economic, and educational background, it is essential that the system be made simple from the perspective of data, processes, and its structure. While standardization of name, address, etc are absolutely essential to create a common, electronically verifiable, national identity for all, it is also critical that these standards does not end up excluding sections of people.

Structure of the "Name" field of the resident is a classic example where a decision had to be made to use whether a western style naming convention where first name, middle name, and last name are captured separately or to use a single field name concept to accommodate all kinds of names from single word to multiple words. Considering the fact that, in India, several people have single word name (even 1 character names) to names having more than 4 or 5 words, it was decided by the DDSVP committee to use a single field to capture name.

Similarly, in India, we neither have a standardized address format nor have well defined geographic boundaries beyond villages. This creates issues when trying to map addresses in a standard way. Various forms issued by existing systems vary greatly when it comes to capturing addresses. It was felt that a standardized address structure must be created to ensure structured, electronic matching but at the same time ensuring it caters to rural, semi-urban, and urban addresses.

It was also essential to ensure a full Date of Birth (DoB) with proof is not mandated since many people in India have no knowledge of exact date of birth or have no

documentary evidence. To ensure no one is excluded due to this, Aadhaar system allows either age to be captured or a full date of birth to be captured when available.

Another issue related to establishing identity is the fact that residents need to have some proof of their name and address while enrolling for Aadhaar. Although a large list of Proof of Identity (PoI) and Proof of Address (PoA) documents are allowed, it was essential that the enrolment process also included residents who may not have any of the valid documents with them. To ensure inclusion, a concept of introducer was created that allowed residents with no PoI or PoA documents to be introduced by a valid listed introducer. In such cases, Aadhaar system captures the Aadhaar number of the introducer along with the enrolment. The Introducer concept was later extended to enrolment of family members through Head of Family (HoF).

1.3.4 Privacy by Design

Security and privacy of personal data has been fundamental in design of Aadhaar system without sacrificing utility of the national identity system. When creating a national identity system of this scale, it is imperative that privacy and security of personal data are not afterthoughts, but designed into the strategy of the system from day one.

Aadhaar system addresses these issues at its core. Following key aspects are results of this.

1.3.4.1 Aadhaar Numbering scheme

Aadhaar number is a random number with no built-in intelligence or profiling information. A 12-digit number was chosen based on the identification needs of the population in the next couple of centuries [10].

It was also decided that, to ensure privacy, the date-of-birth and other attribute information should not be embedded in the number. Similarly, place of birth or residence using administrative boundaries (state/district/taluk) should also not

be embedded within the number. When state/district IDs are embedded, the number faces the risk of becoming invalid and misleading the authenticator when people move from place to place. It can also lead to profiling/targeting based on the region/state/district that a person is from. The approach of storing intelligence in identification numbers was developed to make filing, manual search and book-keeping easier prior to the advent of computers. This is no longer necessary, since centralized database management systems can index the records for rapid search and access without having to section data by location or date of birth.

1.3.4.2 Minimal Data with No Linkage

Since Aadhaar system has data of all Aadhaar holders of the country in a central repository, it was essential that data be kept to a minimum so as to provide identity related functions (issuance and authentication) and nothing else. This design philosophy is derived directly from the fact that UIDAI respects privacy of the residents and not hold non-essential data within its systems.

In addition to having minimal data (4 attributes – name, address, gender, and date of birth - plus 2 optional data – mobile, email), this central database does not have any linkage to existing systems/applications that use Aadhaar.

This essentially creates a set of data islands containing resident data across various applications/systems (a federated model for resident data) rather than a centralized model eliminating the risk of a single system having complete knowledge of resident and his/her transaction history.

1.3.4.3 No Pooling of Data

Aadhaar system is not designed to collate and pool various data and hence does not become a single central data repository having all knowledge about residents. It has no linkage information (such as PAN number, Driver's License Number, PDS card number, EPIC number, etc) to any other system. This design allows transaction data to reside in specific systems in a federated model. This approach allows resident information to stay in distributed fashion across many systems owned by different agencies. Any transfer or collation of data across these distributed information silos should follow due course of law.

1.3.4.4 Yes/No Answer for Authentication

Aadhaar authentication responds only with yes/no answer. For example, authentication answers questions such as "*resident claims his/her name is ..., is this correct?*" whereas it does not provide any scheme to ask questions such as "*what is the address of resident whose Aadhaar number is ...?*". Aadhaar authentication allows applications to "verify" the identity claim by the resident while servicing them while still protecting their data privacy.

1.3.4.5 Explicit Resident Consented e-KYC

A balance between 'privacy and purpose' is critical to ensure convenience of online identity is balanced with the requirement to protect resident identity data. External user agencies do not have access to the Aadhaar database. Aadhaar e-KYC service allows resident to authorize UIDAI to share electronic version of their Aadhaar letter. For every Aadhaar e-KYC request, only after successful resident authentication, demographic and photo data is shared in electronic format (via biometric/OTP authentication resident explicitly authorizes UIDAI to share electronic version of Aadhaar letter instead of sharing physical photocopies). Resident authorization is NOT used for multiple e-KYC transactions, instead, every time agencies require electronic version of Aadhaar letter data for KYC purposes, resident must authorize the agency.

1.3.4.6 No Transaction History

Aadhaar authentication does not have any knowledge of the transaction resident is performing. It only knows that resident is authenticated by this agency at this time. Aadhaar system has no knowledge and is not designed to keep track of specific transaction details such as depositing money or obtaining pension or anything else. This has been consciously designed to ensure resident transaction history is not part of a central system to ensure privacy of the resident.

1.3.5 Ecosystem Approach

National identity projects are run in several countries by a single monolithic agency mandated to enrol its population. It was evident from the very beginning

that given India's diversity – urban/rural, rich/poor, literate/illiterate etc. – the Aadhaar system would be best implemented by a set of cooperating partners or stakeholders.

Given the federal nature of the governance with strong state governments that implement many of the flagship schemes, it was important to enlist the state governments to enrol the residents of their respective states. The project needed enrolling agencies that would actually perform field enrolments on behalf of the registrars. These enrolling agencies also needed to procure standard enrolment kits. It was important to create a cadre of trained enrolment operators who work for enrolling agencies. This, in turn, required training and certification agencies. The ecosystem also needed device manufacturers and suppliers who would provide Aadhaar compliant devices for enrolment; this in turn needed a device testing and certification agency. A similar approach has been followed on the authentication and e-KYC services to ensure that the entire system can derive efficiencies from the Aadhaar system.

The UIDAI has also engaged in continuous dialog with standardization bodies, with technology providers, and other partners to ensure a high degree of uniformity in the quality of data collected and resident experience.

Such an ecosystem approach necessitated that the interfaces between these partners and systems were well defined and standardized. Hence, the UIDAI also needed to build a technology backbone that would hold together this partner ecosystem.

At the data centres where the enrolments were processed, UIDAI needed a set of suppliers to provide pieces of the Aadhaar backend system, most importantly, a set of Biometric Service Providers (BSPs) to provide multi-modal biometric de-duplication software solution that can de-duplicate the incoming enrolment requests and ensure that they are truly unique. In addition, UIDAI needed one or more agencies for application software development, 24x7 data centre operations, and security monitoring.

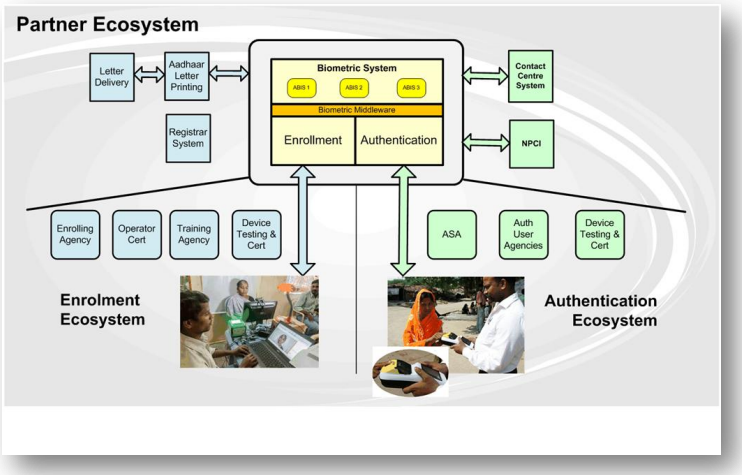


Figure 1: UIDAI Partner Ecosystem

1.3.6 Identity as a Platform

One of the key considerations is to keep the Aadhaar system purely focused on identity and nothing else. The Aadhaar system only collects minimal data just enough to provide unique identity, issue the Aadhaar number after biometric de-duplication, manage lifecycle changes of that identity record, and provide a secure Application Programming Interface (API) for verifying the identity (online authentication) for various applications requiring identity verification.

Designing the Aadhaar system as pure identity platform allows clear separation of duties and leaves usage of identity to other partners, and their various applications which may be built on top of the Aadhaar platform.

For details, refer to program strategy document "UIDAI Strategy Overview" [1] *and more recent "Aadhaar Technology Strategy"* [5].

2 Aadhaar Application

This chapter provides a high level overview of these modules. It is important to understand the application functionality and requirements for scale so that architecture that is implemented can be better appreciated.

2.1 Aadhaar Application Overview

Below figure depicts the Aadhaar application system at a high level.

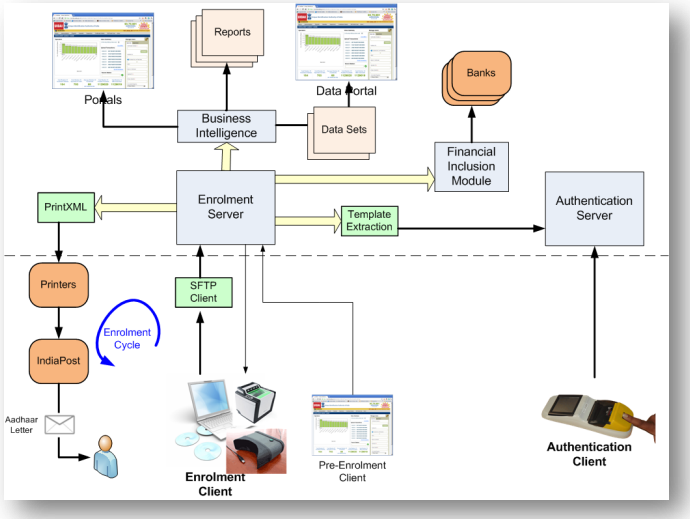


Figure 2: Aadhaar Application Overview

Aadhaar application has three primary sub-systems or modules – Enrolment module that handles Aadhaar lifecycle changes, Authentication & e-KYC modules that provide online identity verification services, and common modules such as business intelligence & reporting, fraud detection, resident and partner portals.

Following sections introduce key features of these modules. Later chapter "*Application Architecture*" discusses architecture and implementation details.

2.2 Use of Biometrics in Aadhaar

The Aadhaar biometric system design has followed global best practices. UIDAI has reviewed existing state-of-the-art biometric systems, consulted with the world's top biometric experts, conducted a proof of concept study and has built a biometric system that is currently considered to be the state-of-the-art.

UIDAI constituted a committee to come up with recommendations. Biometric Standards Committee was constituted to study the benchmarks for biometric de-duplication accuracy in other large systems around the world and recommend the approach to be followed by UIDAI to achieve a high de-duplication accuracy to scale to a population of 1.2 billion.

The committee report [11] found that while in the global context, 99% de-duplication accuracy had been achieved using fingerprints alone in a database size of 50 million, 95% de-duplication accuracy could be reasonably expected using 10 fingerprints and face at a database size of 1.2 billion in the Indian context. Hence, the committee recommended that UIDAI explore the use of the iris biometric in addition to fingerprint and face biometrics to achieve de-duplication accuracy in excess of 99% and also ensure total inclusion. Based on the recommendation, the UIDAI commissioned a PoC study to determine whether multi-modal de-duplication could improve the de-duplication accuracy. The results of the enrolment PoC showed that an order of magnitude better accuracy could be achieved using a multi-modal de-duplication scheme including both fingerprints and iris.

UIDAI decided to accept the recommendations of the PoC report and capture the following, biometrics for the enrollment:

- 10 finger prints
- 2 iris scans
- Photograph of face

2.2.1 Multi-ABIS De-duplication System

Since de-duplication at this scale (1.2 billion residents) had not been previously attempted anywhere in the world, UIDAI decided to procure 3 ABIS (Automatic Biometric Identification System) software solution to perform biometric de-duplication as a risk mitigation strategy.

Aadhaar is the first ever multi-ABIS system implemented in the world, and brings significant advantages:

- It ensures that there is no vendor lock-in. If one of the ABIS solutions needs to be replaced (for any reason – technical or contractual), it can be done without bringing the entire system to a grinding halt.
- The three ABISs compete for work based on their accuracy and throughput. Since payment is based on successful de-duplication (not an upfront payment for software), solution providers compete to improve accuracy and throughput aiming for a higher volume of de-duplications.
- The deployment of multiple ABISs improves the accuracy of de-duplication. If any ABIS identifies a potential duplicate, it is sent to the other ABIS for verification. By combining the results of all 3 ABIS systems the overall biometric de-duplication accuracy goes up.
- The utilization of three different de-duplication engines with different implementations and fusion strategies also helps to detect various kinds of software or data collection errors. In certain enrolments (for example in suspected duplicates and enrolments with poor quality biometrics) the enrollment data is sent to more than one ABIS to minimize the chance of an identification error.

- The use of multiple ABIS engines also permits continuous monitoring. Each duplicate found is used as a "probe" to test the accuracy of the other two ABIS systems. This is critical to maintain the accuracy of the system on an ongoing basis.

2.2.2 Biometrics in Authentication

Fingerprint and iris are the biometric modalities that are being used by UIDAI to allow residents to authenticate themselves. Online biometric authentication is a 1:1 verification of the biometric(s) presented at the time of authentication with templates generated from the data collected during enrolment or biometric updates.

Biometric matches are not exact matches, and such system may fail in the following two ways:

- **False Reject** – The authentication system incorrectly rejects a genuine claim of identity (authentication attempt). The False Reject Rate (FRR) is defined as ratio of the number of false rejects to the number of genuine authentication attempts.
- **False Accept** – The authentication system incorrectly accepts an imposter claim of identity. False Accept Rate (FAR) is defined as ratio of the number of false accepts to the number of impostor authentication attempts.

Similar to enrolment the accuracy of authentication may also be expressed using an ROC curve that shows the trade-off between FAR and FRR. One important difference between enrolment biometrics and authentication biometrics is while the resident presents all ten fingerprints and two iris images during enrolment only one or two fingerprint or iris is presented during authentication.

To attain the best results, the UIDAI makes the following recommendations to application developers:

- **Capture Quality**: Measure the quality of captured biometrics, and request the resident to re-present their biometrics for poor quality captures.

- **2-Finger Authentication**: Fusion allows 2-finger authentication to have lower false reject rate. Use it!
- **Multiple Attempts**: Allow the application to have multiple attempts, before rejecting the business transaction.
- **Backup Authentication**: Have an alternate form of authentication to allow users who may be unable to use biometrics for authentication.
- **Best Finger Detection**: Use Best Finger Detection to guide users to use the best fingers for authentication.

2.3 Enrolment Module

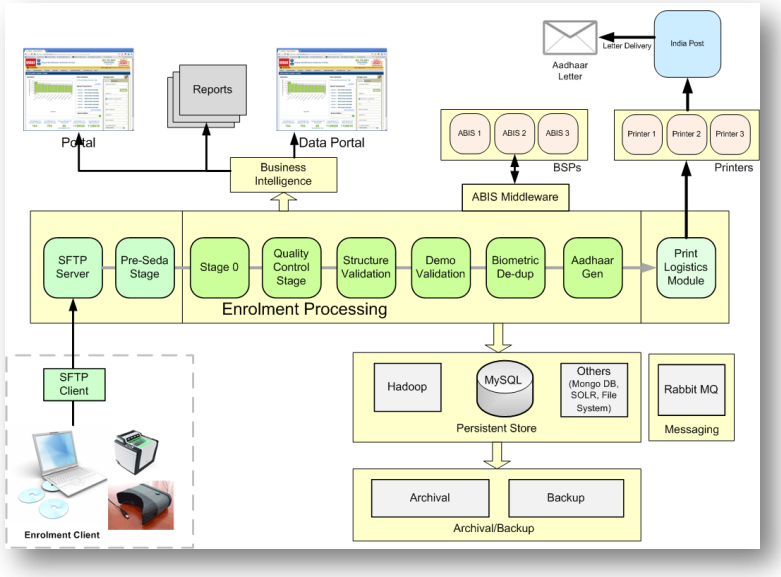


Figure 3: Enrolment Module Overview

Enrolment module handles entire Aadhaar lifecycle including initial enrolment, corrections, subsequent demographic/biometric updates, back-end workflow

related to handling enrolment exceptions, etc. Diagram above depicts the high level picture of the entire enrolment system.

Residents are expected to enrol once and go through multiple data updates during their lifetime. This module is offline, batch-oriented, workflow based and mainly concerned about its overall throughput (number of enrolments/updates that can be processed through the system in a day) rather than individual transaction response time.

2.3.1 Enrolment Client

Aadhaar enrolment strategy is based on a multi-registrar model [1]. This means that for uniformity of data capture, process, and security, it was essential that standardized "Enrolment Client" (EC) software be created and given to all Registrars to be used by their appointed Enrolment Agencies (EAs). Enrolment client software is provided by UIDAI to use it in the field for first-time enrolment and for subsequent data updates.

Enrolment client works mostly offline and has all the features necessary to capture resident demographic details, do local validation, local transliteration, biometric data capture, biometric data quality check, and capturing of necessary audit details such as operator biometrics, location & time. Enrolment client software also has built-in security features such as in-memory data encryption, encrypted data storage, export, etc. It also connects at regular interval to the server at CIDR and exchanges necessary meta-data and master data as part of a "*smart sync*" scheme to constantly be in sync with server. Enrolment client works on Windows and Linux laptops and can connect any of the certified biometric capture devices, standard scanners, and printers.

Enrolment client has the following key features:

1. **Standardized Data Capture**: Enrolment client captures data based on the Aadhaar data standards for resident name, address, etc. This software validates the structure of the data and ensures as much as codified data is used. For example, when capturing address, operator needs to just key in the postal pincode and rest of the information about village, district, state, etc.

are automatically populated from the common geographic region codification master data. Similarly, names and addresses are "*transliterated*" to local language using automated transliteration software built into the client software package. Operator can then make appropriate edits or corrections in transliterated data as required.

2. **Biometric Capture**: Aadhaar enrolment captures resident biometrics as part of the process. This includes all fingers, both irises, and a face photograph. This multi-modal biometric capture require an automated quality check, ability to handle exceptions such as missing fingers, ability to capture a quality face photograph under Indian conditions, etc. These features of Aadhaar enrolment client have been built to balance quality and practicality to ensure best possible biometrics impressions are captured without excluding people whose biometrics are hard to capture. Every time biometric exception is marked, an additional photo showing the exception is captured and the packet is additionally signed off by a supervisor to ensure rogue operators are not violating processes.
3. **Resident Data Validations**: Several Government systems are plagued by correctness of resident data such as name and address due to the fact that electronic data entry is not typically done in the presence of the resident. Although Aadhaar enrolment software allows pre-loading resident data to minimize demographic data entry, actual enrolment is always done in the presence of the resident. This ensures that all resident data (demographic and biometric) is captured together and resident gets a chance to validate the data before it is saved to an encrypted file. Aadhaar enrolment process mandates that enrolment agencies have a second screen (computer monitor) facing the resident at the time of enrolment so that resident can view his/her data and make corrections/suggestions at the time of capture. There is also an explicit step before saving data and a review screen is shown to resident to ensure data correctness in both English and local language. If the resident cannot read at all, operators are expected to read the key data (name, gender, date of birth, and address) back to the resident. Although at this large scale, there is always possibility of mistakes due to a few careless operators, such schemes are built into the software ensuring maximum data correctness.

4. **Corrections & Updates**: Residents who realize any mistake in data capture after enrolment process is done still have a mechanism to approach any operator at the enrolment centre and make necessary corrections to the data within a stipulated time window. In addition, this software allows residents to request updates to name, address, etc as per UIDAI update policy at any time after obtaining Aadhaar number. Several registrars are now setting up permanent Aadhaar centres to facilitate data update need of the resident. Since Aadhaar is a national identity, residents can go to any of the nearest centres to get this done anywhere in the country. While corrections are meant to handle any data entry error by the operator and always linked to original enrolment, update requests are independent at any later point in time and are tracked separately.
5. **Synchronizing with Server**: Information collected by Enrolment client as part of enrolment process is synchronized with server at a regular interval. There are primarily three reasons for regular sync – (1) to ensure all packet metadata is sent to server for tracking enrolments and ensuring the data is received and processed successfully before deleting from enrolment client database, (2) to ensure any server side rules and policy related to enrolment process are enforced on the client, and (3) to ensure master data updates related to location codification, operator activation and de-activation, etc are updated at the client side at regular interval.
6. **Operator/Supervisor Security**: Aadhaar enrolment client only allows approved operators to enrol residents. Activation requires all operators and supervisors to have Aadhaar numbers and also have appropriate certification. Only after they are activated, they can operate the enrolment client software. During enrolment, every packet is biometrically signed by operators and if there are any exceptions such as missing biometrics, it is also signed by supervisor. This happens at the end of each enrolment and then only the electronic packet containing resident data and operator/supervisor biometrics along with other audit data is encrypted and saved. During enrolment packet processing, enrolment server validates the operator and supervisor to ensure only valid operators are enrolling residents. This scheme is built to support non-repudiation and traceability for all enrolments.

7. **Resident Data Security**: Security and privacy of data within Aadhaar system has been foundational and is clearly reflected in UIDAI's strategy, design and its processes throughout the system. UIDAI has taken several measures to ensure security of resident data from the time it is captured all the way to how it is stored within CIDR. Standardized enrolment client ensures that the resident data including raw biometrics is encrypted before even saving to disk. It is to be noted that raw biometrics is never stored anywhere without encryption. Enrolment client uses standard, best in class, cryptographic techniques while storing resident data on field enrolment stations. It uses, encryption based on 2048-bit PKI (which is an asymmetric public/private key encryption scheme) which means that no one can decrypt and misuse the data, even if they are in possession of encrypted enrolment packet. Every enrolment data packet is "*always*" stored in PKI encrypted, tamper evident files and are never decrypted, accessed, or modified during transit until it is reaches within secure production zone of UIDAI data centres.
8. **Process Monitoring**: In general, UIDAI believes in data-driven analytics and continuous improvement of its processes. To enable this for the enrolment process, UIDAI has built-in several features within the enrolment client providing metadata related to the enrolment. For example, every enrolment packet is reviewed by a supervisor for data quality (review audits are captured electronically) and signed as required which means every enrolment is traceable in terms of "*who*", "*when*", "*where*", "*under which agency*", "*under which registrar*", "*who reviewed it*", etc. In addition, several metadata elements such as "*how long operator spent on demographic data screen*", "*how many times a fingerprint was captured*", etc. are collected as part of every enrolment packet for analysis of operator/supervisor actions and performance. This data is used for providing continuous improvement feedback on data quality to the registrars and enrolling agencies using UIDAI's business intelligence platform.
9. **Multi-platform Support**: Since enrolment client software is used by various Enrolment Agencies for Registrars across the country and since the laptop and devices are deployed by the agencies, it is essential that enrolment client software runs on multiple operating systems such as Windows, Linux, etc.

10. **Plug-n-play Device Support**: Entire Aadhaar system is built to be based on open standards and Aadhaar strategy has been to enable the ecosystem to provide best products at best price. By defining common APIs, standards, and certification, Aadhaar system allows multiple vendors to provide devices and components that are certified and create a healthy ecosystem. Enrolment devices such as fingerprint slap scanners, iris scanners, etc. are classic examples of this. There are many vendors who have certified devices complying with Aadhaar enrolment device specification allowing these devices to work with enrolment software in a plug-n-play fashion.
11. **Administration & MIS**: Enrolment client is an offline smart-client having quite a lot of features. Administrative tasks for managing the software configuration including new operator on-boarding etc are managed through the administrative module accessible only to administrative users. Also, various data management features such as export of enrolment packets, import of pre-enrolment data, server synchronization, end of day activities of review, reporting, etc. are all provided as part of this.
12. **Ability to work offline**: Aadhaar enrolment is conducted through camps across the country including cities, towns, villages, and remote locations. Due to the high-end capabilities built-in to software as described above, and due to the nature of connectivity, it was necessary to have enrolment client as an offline, smart-client software with connectivity requirements only for administration aspects.
13. **Master Data Management**: The enrolment client is designed to be a highly configurable system in order to address a wide variety of on-the-field scenarios. Such data include region master data containing postal pin code, state, district, village codifications and mappings, validated and active operator/supervisor data, pre-enrolment data, etc.

2.3.2 Enrolment Server

Enrolment server handles all the functionality necessary to manage enrolment packets, validate them, do necessary quality checks, do demographic match for elimination of trivial duplicates, and most importantly orchestrate biometric de-duplication using multiple ABIS (Automated Biometric Identification System)

solutions to ensure uniqueness before assigning Aadhaar number to the residents. This module also handles lifecycle update processes such as address changes, any manual workflows, and cancellations.

Aadhaar enrolment server components are highly scalable to handle more than 1.5 million enrolments every day and are built to manage data stores in hundreds of terabytes. Entire enrolment workflow is broken up into many logical stages that may take anywhere from minutes to several days depending on the packet validation status. Key stages of enrolment server are described below.

2.3.2.1 Packet Upload and Management

All enrolment packets generated using the enrolment client are fully encrypted from the time of data capture in the field and made tamper evident. Individual enrolment packet contains resident data – demographic and biometric – and other audit data such as operator, date & time, etc. Because of the size of biometrics raw data, each packet is around 3-5 MB in size. Once exported from enrolment client software, these packets are then uploaded (data is always kept encrypted using 2048-bit PKI scheme) to UIDAI data centres, located within India, for processing.

Enrolment agency administrators export these encrypted packets and upload them to UIDAI data centre using a secure upload tool provided by UIDAI. These uploaded packets are tracked, checked for virus/malware, and checked for any tampering before it is moved to inside the production zone of data centre. All valid packets are then decrypted and moved forward through the enrolment workflow whereas invalid packets are rejected. Packet upload and tracking component provide MIS reports and analytics related to this to enable Registrars and Enrolment Agencies so that they can handle any exceptions, re-export and re-upload as necessary.

2.3.2.2 Enrolment Packet Archival

All enrolment and update packets are required to be stored in a permanent archival area across both data centres. Packets are always stored encrypted even within the archive and this module provides services to access, replicate, and monitor archival and replication process. It is important to note that once Aadhaar number is allocated, Aadhaar (UID) master is created, and biometric templates (for

authentication) are extracted, there is no real need for accessing the raw enrolment packet file. Hence entire enrolment packet (original packet in encrypted state as it came from field) is archived and is not accessed until for any manual inspection or investigation. This archival system is loosely coupled and separated from enrolment sub-system. Following are the business requirements:

- Archive system should allow archiving of "all" original enrolment and update packets as-is "forever" (few billion files of roughly 3-5 MB each) and provide high availability and zero loss of data.
- Archive system should ensure data is secured and separated away from core enrolment and authentication systems and should support deployment within a separate sub-network within CIDR.
- It should be designed to be physically disconnected and should allow on-demand data retrieval with appropriate access control and approvals.

Once the data is archived, any temporary copies kept in caches can be purged from operational data stores and backups can be taken from the archive copy.

2.3.2.3 Data Validation

Once the packets are uploaded and verified for viruses or tampering, next step is to validate the data contained within the packet to ensure authenticity and correctness. Following are some of the key validations:

1. **Metadata Validations**: Every packet contains several metadata elements such as station ID, registrar code, EA code, timestamp of enrolment, biometric devices used, etc. This stage of enrolment server validates these against master data and business rules.
2. **Operator/Supervisor/Introducer (OSI) Checks**: Every enrolment packet also captures several metadata (process data) related to enrolment such as who is the operator, who is the supervisor, if introducer was used or not, etc. and corresponding biometrics of operator/supervisor/introducer as required. This is to ensure that all enrolments are done by "authentic and active" operators/supervisors/introducers. As a best practice, in addition to enrolment client side validations, enrolment server also re-validates these. These validations include validating if they have Aadhaar, if they belong to a valid enrolment agency, if they are certified, and they are not black-listed.

3. **Demographic Data Validations**: In addition to above metadata validations, server also validates resident demographic data based on the business rules setup in the system. These include phonetic matching of name, age valid range check, and address structure check against postal pin code and geographic boundary master data.
4. **Biometric Data Quality Check**: Various checks on biometric data quality are also performed at this stage including biometric exception check and face photo check.

2.3.2.4 Demographic Match & Reduced Biometric Check

Demographic matching ensures "trivial" duplicates are filtered out early in the stages so as to not waste compute for doing a full biometric de-duplication. It is observed that residents, without any fraudulent intention, re-enrol themselves when there is a delay in getting their Aadhaar. Similarly, some of the new operators may simply re-enrol residents without cancelling their original enrolment to handle corrections or data entry mistakes. Several of these "trivial" duplicates are caught by this module by doing a combination of demographic and reduced biometric matching.

This module has the capability to configure various demographic searches (exact, partial, and fuzzy). This is done for every incoming enrolment as part of the flow. Once the potential list of matches is created, a 1:1 biometric match is performed against them to detect these trivial duplicates. Because of the 1:1 biometric match facility, unlike pure demographic de-duplication systems, Aadhaar application has the advantage of being "loose and broad" when doing pure demographic match since false matches anyway get filtered while doing automated biometric matching.

This combined approach of using demographic and 1:1 biometric match scheme allows Aadhaar system to detect and eliminate trivial duplicates upfront. If a duplicate is not found, enrolment packets are sent to full biometric de-duplication system and hence there is no real risk of misses at this stage.

2.3.2.5 Manual Data Quality Check

If specific data quality checks fail during automated data quality checks described above, those packets are sent to manual data quality with specific data quality "hints". In addition to failed or suspicious packets, an auto sampling is also implemented to check quality of operators. For example, enrolment packets created by new operators, or operators who have higher percentage of past quality check issues are also subjected to stricter quality checks compared to other packets. At this stage itself, enrolment may get rejected if a significant quality issue is found. This stage is meant to catch and eliminate data entry related issues and other regular operator issues.

2.3.2.6 Biometric De-Duplication

Aadhaar system deploys 3 independent Automated Biometric Identification Systems (ABIS) from 3 different solution providers adhering to common ABIS API [12] for multi-modal biometric de-duplication. All ABIS solutions are "multi-modal" (handles fingerprints, iris, and face) and use their internal matching and fusion scoring algorithm to de-duplicate. Accuracy of these solutions is monitored constantly using measurements of "*False Matches*" and "*False Non-matches*". In addition to actual data, automated probes are also sent constantly every day to do these accuracy measurements.

Enrolment server needs to therefore integrate with all 3 solutions using the standard interface API (ABIS API) and allocate de-duplication requests as per UIDAI policy of dynamic allocation (which uses on-going accuracy and performance data to decide which solution gets maximum de-duplication requests and which one the least). See Aadhaar Product Document [6] for details on how multi-ABIS systems work.

ABIS middleware component of enrolment server handles all the complexities of multi-ABIS orchestration ensuring de-duplication is done using one or more of the ABIS solutions. Middleware, in addition to normal de-duplication request orchestration, also addresses automated probe based testing, accuracy measurements, and error/failure handling.

2.3.2.7 Automated and Manual Adjudication System

All duplicates identified by ABIS systems are sent to the adjudication module. Purpose of this module is to ensure no resident is rejected due to potential occasional false matches of the ABIS systems. This module, in turn, uses an automated scheme to auto dispose sure duplicates based on additional SDK based matching. For example, if ABIS-1 declares an applicant duplicate, Aadhaar system has the capability to send the same request to other two ABIS solutions and also uses additional SDK based matching. This allows high percentage of duplicates to be disposed as rejects in an automated fashion. For a small percentage of applicants with poor quality biometrics or matching scores that are in the grey area, a manual inspection is performed by UIDAI adjudication team. A maker-checker pattern is implemented to ensure "acceptance" of a system-declared duplicate as genuine by adjudication operator requires additional supervisor level verification and approval. Daily performance reports are generated for system monitoring.

2.3.2.8 Aadhaar Number Generation and Allocation

Aadhaar number is a 12 digit number. First 11-digit of the number is purely random and last digit is the checksum based on Verhoeff algorithm [10]. This module handles Aadhaar number generation, ensuring uniqueness of the number (that same number is not allocated to two different residents), and allocating to successful enrolments. Only after successful biometric de-duplication, number is assigned and Aadhaar master record is created. Enrolments that are rejected do not get any number and are stored in reject master database.

2.3.2.9 Print & Logistics Integration

This module, part of enrolment server, handles Aadhaar letter printing, allocation to appropriate printing vendor, tracking print status, and ensuring printing is completed through a defined workflow. This module also has the capability to re-print specific letters, handle letter printing during Aadhaar update lifecycle, rejection letter printing, etc. Notice that UIDAI has multiple printing vendors selected through public procurement and all these are integrated using a common integration layer. All print documents are sent as encrypted, digitally signed XMLs

to appropriate printing vendor based on language, geography, and other allocation rules. After printing Aadhaar letters, letter bags are handed over to India Post for letter delivery.

2.3.2.10 E-Aadhaar Service

E-Aadhaar is an online service for residents to print their own Aadhaar letters from UIDAI resident portal. This service allows residents to download digitally signed PDF letter once Aadhaar is generated and not wait for letter to come via post. E-Aadhaar site requires full 28 digit enrolment id (enrolment number with date timestamp all the way to seconds), name, and pin code to be entered before it can be accessed. In addition, security features such as captcha, One-Time-Pin (OTP) on mobile, etc are implemented. Once input is verified, resident is allowed to download digitally signed PDF which can be shared with agencies and/or printed for use.

It is important to recall that Aadhaar is just a number and is designed to be authenticated (verified) online rather than giving significance to physical cards/letters, etc. That means, residents are allowed to print any number of letters or have colour copies of their letters! There is no special security mechanism or significance attached to this letter/card. Unless the number is authenticated online using demographics/biometric/OTP factors, this letter/card mean nothing. Once e-KYC service is widely adopted by various agencies, need for any physical letter will cease to exist, instead, residents can simply approach agencies for a service, use their biometrics/OTP to authenticate and have agency obtain digitally signed electronic version of Aadhaar and completely eliminate cards, photo copies, etc.

2.3.2.11 Update Services

Update module handles Aadhaar lifecycle update requests such as address update, mobile number update, photo update, etc. Aadhaar system allows residents to update demographics data and biometric data based on a set of published rules. All

updates require resident authentication to protect Aadhaar data from unauthorized updates. UIDAI currently provides both assisted and self-service modes of updates after due authentication of the resident and validation of appropriate documents. This module ensures all rules are validated, resident authentication is done, request origination is validated, and after the update, a notification is mandatorily sent to resident.

To provide multiple options to resident to manage any data updates, UIDAI has created the following update channels:

1. **Permanent Update Centre**: Residents may go to a permanent update centre managed by a registrar where an operator helps resident with data process. There are two formats for the update centre – one allowing a complete data update including biometrics, and the second allowing only demographic data updates. During the complete update process, packets are securely generated and uploaded quite like initial enrolment.

   For demographics only updates, a lighter version of the enrolment client software is used. It is expected that the registrars will setup a larger number of demographic update centres to cater to the higher volume of these requests as compared to the lower volume of biometric update requests. In CIDR, these packets are scanned, validated, and processed via the update flow.

2. **Self-service Update**: Residents having registered mobile with Aadhaar system can go to online update portal managed by UIDAI and use OTP authentication to request data update. After the approval workflow, update packets are generated and passed on to automated update flow. This mode allows residents to request for any demographic data update. In future, newer self-service channels such as mobile application may be provided.

3. **Contact Centre and/or Postal**: A few limited fields are allowed to be corrected or updated by calling UIDAI Contact Centre and sending supporting documents via post. After manual inspection and approval, update packets are created by UIDAI and passed onto automated update flow.

4. **Update API**: People may change mobile numbers and address frequently as compared to other fields and it is critical that UIDAI provide maximum

touch points for residents to be able to easily update their data. UIDAI intends to provide an "Update API" to select agencies (which are already authentication agencies) where resident can authenticate and request data updates.

Update features use common services that are part of enrolment server and most validations such as station validation, registrar/EA validation, operator/supervisor validation, etc. are identical to that of initial enrolment flow.

2.4 Authentication Module

Aadhaar authentication is the process wherein Aadhaar number, along with other attributes, including biometrics, are submitted online to the CIDR for its verification on the basis of information or data or documents available with it.

Following diagram depicts the high level overview of Aadhaar authentication.

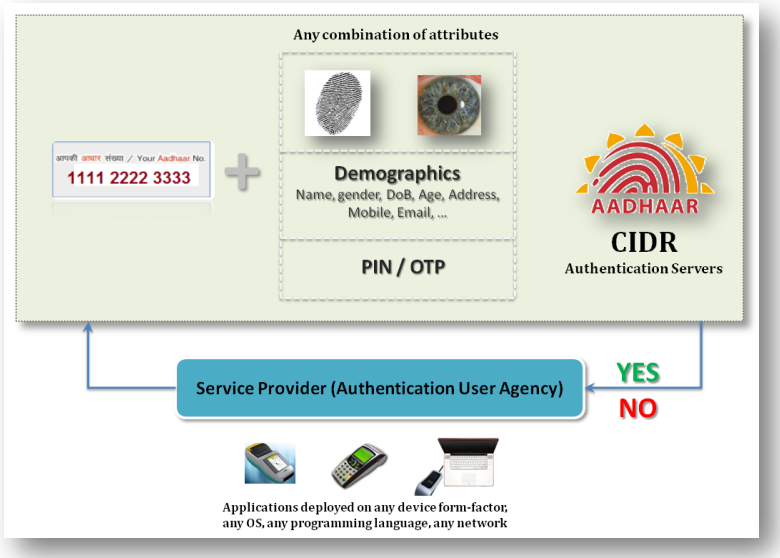


Figure 4: Aadhaar Authentication Overview

Authentication module handles online resident authentication from various authentication user agencies. In addition to the main authentication service which offers multi-factor demographic/biometric/OTP authentication in an end to end secure fashion, this module also implements related services such as best finger detection, OTP request, etc.

Aadhaar number along with certain demographic information such as name of the resident, date of birth, etc. being matched can cater to simple authentication needs. Combination of Aadhaar number and biometrics deliver online authentication without needing a token (such as a smartcard). During biometric authentication, agency collects the Aadhaar number along with one or more biometric impressions (e.g., one or more fingerprints, or iris impression alone, or iris impression along with fingerprints) which then encrypted and sent to Aadhaar authentication server for authenticating the resident.

For example, when MNREGS beneficiary is enrolled and given a job card, resident could be biometrically authenticated against Aadhaar system to verify his/her Aadhaar number along with name and age. Similarly, while disbursing wages, beneficiary may be asked to provide Aadhaar number and one or more fingerprint impressions to authenticate before cash is disbursed.

Authentication focuses on matching a person's identity based on the reliability of a credential offered. Various agencies have different requirements for the degree of assurance required while authenticating beneficiaries/customers. When authenticating a resident, multiple factors may be used to strengthen the authenticity of the request.

In general, following are the 3 categories of factors:

1. ***What you have***: Something the user uniquely has (e.g., a card, security token, mobile phone, tablet/laptop computer accessing email, etc.).
2. ***What you know***: Something the user knows that is not public (e.g., a password, PIN, secret question, etc.). Demographic details such as date of birth may also be classified in this category although they are generally considered weak factors.

3. ***Who you are***: Something the user individually is or does (e.g., fingerprint pattern, iris pattern, signature, handwriting, etc.).

Aadhaar authentication provides multi-factor authentication based on "*what you have*" using mobile phone and "*who you are*" using resident Fingerprints and Iris. By combining one or more factors, authentication of the resident could be strengthened. In addition, Authentication User Agency (AUA) specific factors such as ATM cards or smart cards or passwords may also be used in conjunction with Aadhaar authentication to further strengthen user authentication.

2.4.1.1 Federated Authentication Model

UIDAI offers Aadhaar authentication as a federated model. This implies that Aadhaar authentication usually works in conjunction with and strengthens an AUA's existing authentication system (as opposed to replacing an AUA's existing authentication system).

Most authentication systems could be described as "local" (i.e., pertaining to and/or valid for a few services, situations or entities) and "revocable" (wherein an existing identity factor could be revoked and reissued as a result of expiry, compromise or other valid reasons). Such revocable, local authentication systems come with set of strengths and limitations. Aadhaar authentication system, on the other hand, could be described as "global" (because of its applicability across situations, AUAs and services) and "non-revocable" (because Aadhaar identity factors such as fingerprints and iris scans cannot usually be revoked/replaced). Global, revocable/permanent authentication systems come with their own set of strength and limitations.

In the federated authentication model envisaged by UIDAI, the global-irrevocable authentication Aadhaar authentication system co-exists with and strengthens the local-revocable authentication systems of AUAs. It is expected that such a federated approach would eventually result in authentication systems which are stronger and more reliable than those that are based either only on global-irrevocable systems or only on local-revocable systems.

The following are some types of situations where an AUA could use Aadhaar authentication either on a federated authentication mode or on a standalone basis:

1. **One time usage**: When enrolling a new customer or creating a new service account for an individual. Examples are the issuance of a new PAN card, a new passport, creation of a new bank account or an internet service account for an online business. The AUAs in all such cases could authenticate an applicant's identity using the applicant's Aadhaar PID before issuing their own authentication factors.
2. **Periodic Usage**: AUAs can also use Aadhaar based federated authentication system for periodic update of their customers' (or employees' or associates') identity information. Examples are using Aadhaar authentication as a basis for renewing an Aadhaar holder's KYC data, the address of a bank account holder, etc.
3. **Transactional Use**: AUAs can also use Aadhaar based federated authentication system for carrying out any of their other business transactions. Examples include banks that authenticate a customer's Aadhaar PID as well as bank-related identity information (account number/user id along with password/OTP, etc.) before enabling banking transactions such as funds transfer, funds withdrawal, etc.

It is important however, to note that the federated model does not mandate the existence or use of an AUA's local-revocable authentication system in conjunction with Aadhaar authentication system. If an AUA so wishes, they could use only Aadhaar authentication to enable their services.

2.4.2 Authentication APIs

Aadhaar authentication is an online service offered by UIDAI to approved organizations to verify identity claim by the resident. This service is offered via an Application Programming Interface (API) that allows organizations to integrate Aadhaar authentication within their applications. Aadhaar Authentication API [13] is an open, published specification and works with applications written in any programming language, running on any computer or device, using any network including mobile networks. In addition to core authentication API, there are two

more supporting APIs – Aadhaar Best Finger Detection (BFD) API [14] and Aadhaar One-Time-Pin (OTP) Request API [15].

2.4.2.1 Best Finger Detection (BFD)

During the initial authentication POC studies, it was identified that the quality of fingerprints may vary considerably between fingers of the same resident. This could be due to many reasons, including wear and tear on the finger, ability to present a finger to the sensor, or errors during enrolment.

It may thus be useful to appraise each resident of finger providing the best accuracy and successful matching results. We shall refer to this finger with best accuracy as the best finger. Resident may possess one or more best fingers. This knowledge allows the resident to provide his/her best finger(s) during authentication thereby increasing the chances of successful match.

Since a resident would normally not be aware of the best fingers to use for authentication, a Best Finger Detection process was defined where the resident would send a set of fingerprints (as captured from an authentication device) to the UIDAI server which would respond with the list of fingerprints that the resident could use for the purpose of UIDAI authentication. The value of this method was proved through additional POC studies that were carried out by the UIDAI [4], where the use of BFD brought down the False Reject Rate for residents significantly and thereby improved accuracy.

The UIDAI has provisioned the Best Finger Detection as an API [14], which is available to all user agencies. UIDAI has mandated that this be used by user agencies to improve speed, and accuracy and overall customer satisfaction.

It is further recommended that residents be required to go through this process if they are unable to successfully authenticate themselves in a repeatable manner, and that all user agencies provision an appropriate application on authentication devices for this purpose.

2.4.2.2 One-Time-Pin (OTP)

OTP authentication allows AUAs to verify the possession of the mobile phone by resident by sending the OTP to his/her mobile phone and re-verifying it. One Time Pin (OTP) is a mechanism to do achieve this.

In a nutshell, OTP request can be initiated by the resident (via IVR or SMS) or by the application on behalf of the resident. OTP API [15] specification allows "Application Initiated" OTP request. Irrespective of how OTP request was initiated, a 6-digit OTP is always delivered on the resident's mobile/email and application is expected to capture and validate using Authentication API. Usage of OTP is particularly useful for web applications such as online insurance renewals, e-Governance portals, and so on.

2.4.3 Authentication Server

Aadhaar authentication server implements all the logic required to match incoming authentication request against the authentication database where demographics and biometric data of Aadhaar holders are stored. Authentication server does the necessary matching of demographics, biometric, and OTP values and respond with appropriate response as per API specifications. Authentication server also implements server side logic for BFD and OTP request APIs. In addition to matching logic, it also implements inline fraud detection, auditing, SMS/Email resident notifications, and event publishing for use by Business Intelligence (BI) and offline fraud analytics.

Authentication server is currently benchmarked to handle 100 million authentication requests in 10 hrs with sub-second average response time. Authentication service, being online and available 24x7, runs in active-active configuration across both data centres with load balancers and mutual fail-over scheme. It is built to scale horizontally and can be scaled further simply by adding application server nodes to the cluster. 100 million authentications a day also means that several billion BI events and audit records are created every few days which necessitates both BI and audit data stores to be able to handle huge number of data records.

For every authentication request, at a high level, authentication server logic is as follows:

- Validate the ASA code and license key
- Validate the input for strict XSD compliance
- Validate AUA code, license key, and digital signature
- Validate input as per API specification
- Do inline fraud check
- Do necessary demographic/biometric/OTP matching
- Publish BI event
- Form response and digitally sign it
- Write encrypted, signed audit containing input and output
- Send response

2.5 E-KYC Module

Verification of the Proof of Identity (PoI) and Proof of Address (PoA) is a key requirement for access to financial products (payment products, bank accounts, insurance products, market products, etc.), SIM cards for mobile telephony, and access to various Central, State, and Local Government services. Today, customers provide physical PoI and PoA documents. Aadhaar is already a valid PoI and PoA document for various services in the financial, telecom, and Government domains.

The Aadhaar e-KYC API [16] provides a convenient mechanism for agencies to offer an electronic, paper-less KYC experience to Aadhaar holders. The e-KYC service provides simplicity to the resident, while providing cost-savings from processing paper documents and eliminating the risk of forged documents to the service agencies. UIDAI has published its policy on e-KYC [17] and e-KYC API is implemented as per the policy.

Following are the salient features of the service.

1. **Paperless** – The service is fully electronic, and document management can be eliminated.
2. **Consent based** – The KYC data can only be provided upon authorization by the resident through Aadhaar authentication, thus protecting resident privacy.
3. **Eliminates document forgery** – Elimination of photocopies of various documents that are currently stored in premises of various stakeholders reduces the risk of identity fraud and protects resident identity. In addition, since the e-KYC data is provided directly by UIDAI, there is no risk of forged documents.
4. **Inclusive** – The fully paperless, electronic, low-cost aspects of e-KYC make it more inclusive, enabling financial inclusion.
5. **Secure and compliant with the IT Act** – Both end-points of the data transfer are secured through the use of encryption and digital signature as per the Information Technology Act, 2000 making e-KYC document legally equivalent to paper documents. In addition, the use of encryption and digital signature ensures that no unauthorized parties in the middle can tamper or steal the data.
6. **Non-repudiable** – The use of resident authentication for authorization, the affixing of a digital signature by the service provider originating the e-KYC request, and the affixing of a digital signature by UIDAI when providing the e-KYC data makes the entire transaction non-repudiable by all parties involved.
7. **Low cost** – Elimination of paper verification, movement, and storage reduces the cost of KYC to a fraction of what it is today.
8. **Instantaneous** – The service is fully automated, and KYC data is furnished in real-time, without any manual intervention.
9. **Machine readable** – Digitally signed electronic KYC data provided by UIDAI is machine readable, making it possible for the service provider to directly store it as the customer record in their database for purposes of

service, audit, etc. without human intervention making the process low cost and error free.

10. **Regulation friendly** – The service providers can provide a portal to the Ministry/Regulator for auditing all e-KYC requests. The Ministry/Regulator can establish rules for secure retention of e-KYC data (eg. storage method, period of storage, and retrieval among other things).

2.5.1 Compliance with the IT Act, 2000

The data provided to the service provider is fully in compliance with the Information Technology Act (IT Act), 2000 as follows:

1. The e-KYC electronic record provided by the UIDAI is equivalent to the Aadhaar letter (Section 4 of the IT Act, 2000);
2. A cryptographic hash of the KYC data is computed and attached. The SHA-2 digital hash function algorithm is used. Hashing ensures that any tampering of the data in transit is detected (Section 3 of the IT Act, 2000);
3. The KYC data along with the computed hash are encrypted using a combination of AES-256 symmetric key and RSA-2048 PKI encryption, form a secure electronic record. The encryption ensures that only the intended service provider can view the data provided by the UIDAI (Section 14 of the IT Act, 2000); and
4. The encrypted data and hash are digitally signed by the UIDAI using RSA-2048 PKI. The secure digital signature of UIDAI can be verified by the service provider to ensure the authenticity of the source (Section 15 of the IT Act, 2000).

The e-KYC service is compliant with the latest standards notified in the Information Technology (Certifying Authorities), Amendment Rules 2011 – 25th October 2011(GSR 782(E) & GSR 783(E)-Standards (Hash & key Size), usage period of private keys, and verification of Digital Signature Certificate.

2.5.2 E-KYC API

Aadhaar e-KYC is an online service offered by UIDAI to approved organizations to obtain digitally signed electronic Aadhaar record. E-KYC works only with resident's Aadhaar authentication and explicit consent. This service is offered via an Application Programming Interface (API) that allows organizations to integrate Aadhaar e-KYC within their applications. Aadhaar e-KYC API [16] is an open, published API specification and works with any application written in any programming language, running on any computer or device, using any network including mobile networks.

Broadly speaking, e-KYC API can be used under the following two scenarios:

1. New customer/beneficiary:
    a. The KUA captures resident authentication data and invokes the Aadhaar e-KYC API through a KSA network;
    b. The KYC data returned within the response of the e-KYC API is digitally signed and encrypted by UIDAI; and
    c. Using the resident data obtained through this KYC API, the agency can provision the service instantaneously.
2. Existing customer/beneficiary:
    a. The KUA captures resident authentication data and invokes the Aadhaar e-KYC API through a KSA network;
    b. The KYC data returned within the response of the e-KYC API is digitally signed and encrypted by UIDAI;
    c. Since the resident is already a customer/beneficiary, the KUA can use a simple workflow to approve the Aadhaar linkage by comparing data retrieved through the e-KYC API against what is on record (in paper or electronic form); and
    d. Once verified, the existing customer/beneficiary record can be linked to the Aadhaar number. The Aadhaar e-KYC API returns digitally signed data allowing electronic audit.

2.5.3 E-KYC Server

Aadhaar e-KYC server implements all the logic required to serve incoming e-KYC requests. E-KYC server, in turn, uses Aadhaar authentication service for resident authentication to ensure only resident authenticated and authorized requests are serviced. In addition, it also implements inline fraud detection, auditing, SMS/Email messaging as required, and event publishing for use by Business Intelligence (BI) and offline fraud analytics.

At a high level, e-KYC server implements the following logic:

- Validate the KSA code
- Validate the input for strict XSD compliance
- Validate input as per API specification
- Invoke authentication API
- Invoke Common Service API to fetch resident demographic data and photo
- Publish BI event
- Form response and digitally sign it
- Write encrypted, signed audit containing input and output
- Send response

2.6 Common Modules

Quite like any other large scale applications, Aadhaar system also is built using common components that can be reused across Enrolment and Authentication. Common modules consists of business intelligence & reporting, fraud detection, resident and partner portals, and other common platform capabilities such as scalable data stores, data encryption/decryption, high volume messaging, etc.

2.6.1 Technology Platform

This module consists of all common technology components. This includes data source access, file system access, rules engine, messaging, encryption/decryption, number generation, data caching, data synchronization, etc. All common components used across various other modules are part of this.

2.6.2 Internal APIs

There are several internal APIs that are accessed by other applications within CIDR. Encapsulating common functionality into a reusable service (REST style service or Java API) ensures functionality and business rules are centrally managed and not repeated. Examples of such internal APIs/services include:

- Enrolment Status Service (ESAPI)
- Common Search Service (CSAPI)
- Advanced Search Service (ASAPI)
- Internal OTP API
- Common Update Service

2.6.3 Business Intelligence & Reporting

At UIDAI, Analytics and Reporting has been a constituent of the Aadhaar implementation strategy from inception. A cross-functional analytics and continuous improvement team was created at an early stage to suggest and oversee usage of analytics across the organization. Business Intelligence (BI) systems provide the extensible infrastructure platform, framework and associated tools to help meet goals of the Analytics and Reporting function.

The BI module provides comprehensive analytics and reports for:

- Partners of UIDAI for effective field management and process improvement via partner portal
- UIDAI internal users for program monitoring, management, and continuous improvement

- Public at large via public portal for providing transparency
- Statisticians and researchers to allow anonymized aggregate data sets for purposes of field studies and advanced research

2.6.4 Application and Information Portals

Portals provide a single window for residents, Registrars, Authentication User Agencies (AUAs), other partners in the ecosystem as well as UIDAI officials to view and manage various types of data. These portals provide static and dynamic data, manages access privileges, and manages data that are real-time and periodic in nature. Primarily these fall into 4 categories:

2.6.4.1 Resident Portal

All resident facing services such as enrolment status tracking, e-Aadhaar print service, self-service update service, etc are provided via this portal.

2.6.4.2 Partner Portal

All partner features are made available via partner portal. Partner document repository, partner level administration (which can be done by partners), reporting and BI services, etc go into this portal. These include enrolment, authentication, and e-KYC related features.

2.6.4.3 Internal Portal

UIDAI users have access to internal portal which allows approved users within UIDAI organization to access and manage various business rules, reports, and watch the overall performance of the system.

2.6.4.4 Data Portal

UIDAI also publishes machine readable datasets (fully anonymized analytics data) through Data Portal as part of 'Government Open Data Initiative' [18]. UIDAI Data Portal has a Catalogue of data sets. Various data sets related to enrolment and authentication are available in machine readable format.

2.6.5 Application Monitoring

The primary purpose of the monitoring solution is to proactively identify potential issues in system in terms of service availability, throughput, and response time. Aadhaar application monitoring module is non-intrusive, near real-time, and graphical. This helps various business metrics to be monitored visually and alerts are created based on system defined rules. Monitoring module also allows hourly, daily, and weekly analysis of issues.

2.6.6 Fraud Detection

Since Aadhaar system only deals with identity management and identity authentication, UIDAI concerns itself only with identity fraud. Since Aadhaar identity is verified online against strong factors such as biometrics or OTP, using fake numbers, forged Aadhaar letters, or usage of someone else's number will not work. UIDAI has setup a fraud detection system to detect, investigate, and manage any potential cases. Since Aadhaar system uses biometrics to de-duplicate every enrolment, enrolling under a fake name or address does not help since residents can only get one identity that is valid for life. If an Aadhaar record is found to be fraudulently obtained, Aadhaar system has the ability to cancel that Aadhaar, thus instantly disabling any subsequent authentication and usage.

For details, refer to Aadhaar Product Document [6].

3 Architecture Principles

This chapter covers architecture principles used in building Aadhaar system ensuring openness, vendor neutrality, scalability, and security. Before introducing Aadhaar architecture principles, this chapter looks at software system architecture trends over the last couple of decades and looks at some of the high impact changes in recent years. This understanding is critical in appreciating Aadhaar architecture and the reasoning behind those decisions.

3.1 Architecture Evolution & Trends

Software architecture has evolved from mainframe era to cloud computing era and as part of that change, computing, storage, and programming technologies also have evolved. Monolithic architecture has given way to large scale distributed computing architectures, proprietary storage and compute has given way to commodity computing and large scale low cost storage, user interface has changed from character based fixed green screens to highly interactive gesture based mobile interfaces, and nearly no connectivity to having pervasive connectivity. Massive increase in the amount of data managed within applications from mere in kilobytes and megabytes to petabytes and hexabytes have forced architects to rethink on design choices for computing and data store within applications.

These changes have had huge impact when building next generation, large scale applications. Subsequent sections in this chapter explore these trends, changes, and set the context for understanding Aadhaar architecture strategy.

3.1.1 Scale-up, Scale-out, and Open Scale-out

Over the last four decades, software system architecture has evolved from a monolithic, single large server deployment to highly distributed, heterogeneous technology deployment. This has resulted in significant architecture and design shift from a vendor locked-in system to highly open and distributed system.

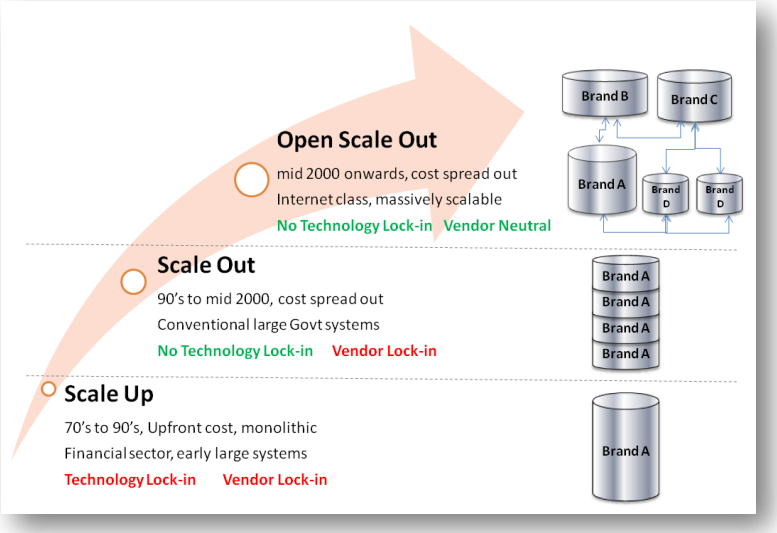


Figure 5: Architecture Evolution

3.1.1.1 Scale-Up Architecture

In the 70's and 80's, large scale systems were designed to be monolithic and deployed in a large mainframe class machine. These systems are built using specific technologies provided by one vendor who also provide the large hardware for deploying these applications. Once you decide to build using technology provided by one mainframe (or similar large scale computing platforms) vendor, then for several years following it, application scaling, system upgrades, maintenance, etc all completely depends on that particular vendor.

Such architecture is referred to as scale-up architecture where entire system needs to scale within one machine (by adding more computing power within the machine) and that too using the technology provided by that vendor. Most of the time, significant upfront investment needs to be made planning for future growth and scaling instead of upgrading the systems at a later stage when it is really required to do so.

Several financial systems built in the 80's and even in 90's were completely designed this way and it is extremely hard for these applications to move out of this to an open architecture of today. Even when systems were built in the non-mainframe era using languages such as C/C++ running on a Unix platform, several applications entirely depended on large scale vertically scalable machines with enormous single central database systems. Most of these applications completely depend on the platform provider to upgrade compute, memory, maintenance, etc and have no choice to choose an alternate provider.

Such lock-in makes the entire business at the mercy of one of two vendors. Enterprise having such applications could never take advantage of the sharp decrease in prices of compute, storage, etc and constant increase of compute speed around them as technology evolves rapidly.

When building large scale e-Governance application, it is more so important not to lock entire system to one or two technology provider. This ensures vendor neutrality, ability for Government departments to make open procurement at best prices when required, and completely avoid upfront investment.

3.1.1.2 Scale-Out Architecture

From the 90's, with the advent of client-server architecture, application architecture changed to support different components of an application to be deployed in different machines and allowing them to scale independently. Most of these systems were built to run on Unix or Windows environments and middleware technologies allowed components to communicate with each other across machines over a network. Large scale applications deployed within enterprises and early Internet applications followed this architecture.

Although capability to scale horizontally (across machines) at the application server layer exists in these applications, most of them depended on a large database node that can only scale-up since use of distributed data stores were not at all common. Interestingly, several of these applications eventually had lock-in to a particular application server or database server vendor. Also, even with horizontal scaling (scale-out), uniformity of the computing environments such as use of same hardware or OS platform across all application server nodes was assumed. For example, application depending on a large RDBMS vendor has no choice to depend on a uniform, expensive, high-end storage below to gain performance and scale.

Although scale-out architecture allowed scaling as the systems grew, use of heterogeneous (different OS, different storage, different sized machines, etc all within same application deployment) environment was rare. It is important to note that most large scale e-Governance applications got into this trap and eventually is at the mercy of specific vendors for upgrades, maintenance, etc not having capability to take any advantage of crashing prices and significant technology changes.

3.1.1.3 Open Scale-Out Architecture

With significant maturity of cloud platforms and very large scale Internet class applications over the last 5 years, application architecture also evolved into a highly scale-out (distributed across 1000's of machines), highly open, heterogeneous taking constant advantage of newer high speed, low cost hardware and storage.

Open scale-out architecture does not depend on specific computing platform, specific storage, specific OS, specific database vendor, or any specific vendor technologies to scale. Such applications are built completely using open source or open technologies and architected to address scalability in a vendor neutral fashion and allow co-existence of heterogeneous hardware within same application. Most importantly, such architecture also allows delayed procurement of hardware from any vendor as the systems grow. Every component of the core system must follow this principle in order to ensure entire system gains these advantages.

3.1.2 Commodity Computing

Commodity technology refers to hardware and software technology that are completely commoditized and are cheaply available from a variety of providers. Applications that are built on such technologies are built on commodity computing architecture.

Whenever building massively scalable systems requiring large compute and storage capability, choice of computing architecture become extremely critical. With Aadhaar applications needing trillions of computations every day and storage of petabytes of data, this was one of the most important architecture decisions. Choice existed from custom computers with massive parallel computing capability, mainframes, special chipsets, and use of commodity x86 platforms. With the maturity of cloud computing and recent success of massively scalable Internet class applications such as social networking have clearly demonstrated that applications can indeed be built on top of open source taking advantage of large deployment of commodity hardware.

Applications that are architected to only use commodity hardware fully benefit from using best technologies at a very cheap cost and allow applications to not be tied to a proprietary and vendor specific technology. Large e-Governance applications should always choose this instead of custom alternatives. Such applications also benefit best when technology evolves at a rapid pace.

3.1.3 Distributed Platforms & Data Stores

As Google, Yahoo, Amazon, and other early dominant Internet giants built their backbone to cater to massively distributed applications handling petabytes of data, a new way of looking at commodity, open, massively distributed computing platforms and data stores emerged. Google demonstrated how cheap compute can be used and still achieve massive scalability for applications and data.

3.1.4 Ubiquitous Connectivity

Mobile connectivity revolutionized Indian communication landscape from a very limited landline penetration to nearly a billion mobile connections across the country. Mobile networks are now used for all kinds of communication in payment, entertainment, and other verticals and use of SMS as an effective two way communication scheme. While traditional broadband and Internet use continued to grow very slowly over last several years, in 2012, Indian mobile Internet usage exceeded all other channels.

With nearly all of India now covered under mobile networks, Telcos continue to aggressively expand their 2G and 3G coverage across country, and with really cheap tariffs, being online and connected is now taken for granted. Several applications that are being built now take advantage of this pervasive connectivity and allow users to access massive amount of information online and share with other users.

3.1.5 Mobile, Tablet, and Handhelds

One of the most transformational technologies that completely changed the face of India is the massive adoption of mobile phones. Desktop usage, especially in India, was very low and nearly stagnated without mass scale adoption. From nearly no phone access, Indians went straight into using mobile phones in a massive way. A combination of regulatory approach and open market approach allowed many companies to compete and provide best value to end customers. Cheap phones, extremely affordable tariffs, and most importantly a massive distribution network to handle pre-paid plans and recharges allowed an explosion of user base to a billion people.

Desktops, quite like landlines, never penetrated into daily lives of masses due to its price, complexity, interaction model, and its inability to be mobile. Whereas smart phones, with its affordable prices, simplicity, easier touch based interaction, and mobility has caught the imagination of masses. Smart phone adoption in India is exploding at a rapid pace and will surely become the de-facto computing device for

millions of people that can be used for communication, games, entertainment, and business purposes anywhere, anytime.

Along with smart phones, larger form factors such as tablets directly address all the issues that plagued desktop adoption. With natural touch based interaction, connectivity, and mobility, tablets will allow seamless content access and social interactions across many verticals such as education, entertainment, and healthcare.

3.1.6 Impact of Technology Trends in Aadhaar

Aadhaar system is built to last and assuming key trends described in above sections. Use of open scale-out architecture, use of commodity hardware, online APIs that can work on mobiles and tablets, and use of large scale distributed data stores for petabytes of data management and large scale analytics within Aadhaar application are reflection of embracing these trends.

Aadhaar application is built using a completely open, heterogeneous, and scale out architecture ensuring no vendor or technology lock-in throughout the system. Aadhaar application already runs on hardware (compute, storage, and network) from different vendors procured at different time as the system grew thus taking advantage of cheaper and better hardware available from any vendor. Choice of large scale distributed data store and analytics systems within Aadhaar application takes advantage of recent developments in big data architecture. Aadhaar application uses simple commodity storage for large scale data storage and ensures redundancy and disaster recovery through external means.

Open APIs and technology dependency of Aadhaar enabled applications are built in such a way that these applications can work from small handheld devices to smart phones to tablets to laptops. This ensures broad adoption of these APIs from a simple attendance style embedded system all the way to larger applications running on a full-fledged laptop. In addition, these APIs work over simple 2G network or through any other network across the country. Instead of offline technologies, processes, and frameworks, Aadhaar system is built from ground up as an online system taking advantage of ever increasing connectivity.

3.2 Aadhaar Architecture Principles

At the beginning of this mass scale project, it was critical that fundamental principles be clearly laid out based on which all subsequent technical decisions are made. These principles ensured entire team understood the axioms by which technology choices were made. For any large e-Governance project, it is critical to write these rules down clearly so as to enable current and future teams to make right decisions as an on-going basis.

3.2.1 Key Assumptions

1. **E-Governance applications should look for transformational opportunities** – Instead of imitating paper process in electronic form, applications should look to fully embrace mobile adoption, digital signature, online authentication, etc. to transform the processes completely and offer wider choice and no/low touch point for residents to interact directly. It is critical that project design are aligned to larger trends and designed for next decade rather than past.
2. **Technology rapidly evolves and requires continuous adoption** – Technology evolves way too fast and Government projects with its rigid vendor selection and long procurement cycles do not align naturally to adapt to this trend. Also, making changes to existing implementations require contract changes, new RFP (Request for Proposal), etc. Hence it is very essential that entire system is built to be open (standards, open API, plug-n-play capabilities), components coupled loosely to allow changes in sub-system level without affecting other parts, architected to work completely within a heterogeneous compute, storage, and multi-vendor environment.
3. **Ability to select best product at best rate in as and when required** – Large e-Governance applications should be designed to get best cost and performance advantages of natural technology curve (constant increase of speed and decrease of cost) and still aligned to open procurement practices of the Government. For this to happen, architecture should be open, use commodity hardware, have no vendor lock-in, and designed for horizontal

scale. This allows buying of commodity compute, storage, etc only when needed at best price.

4. **Incentive aligned design** – Whenever significant changes are made, solution must naturally cater to the incentives of the "participants" within that system. In any transformational e-Governance project, if design is not aligned to natural incentives of residents, Government, and partners, there is a high chance that the project will fail.

Given the above assumptions, Aadhaar technology team laid out a list of architecture principles based on which all design decisions were made. Following sections describe these principles in detail.

3.2.2 Openness and Vendor neutrality

Avoiding vendor lock-in was essential to Aadhaar architecture due to two primary reasons. Firstly, this is a Government program built as the national identity platform and need to sustain openness in the long run and secondly a program of this scale has never been attempted before.

Recall the section above on key assumptions. As we discussed in that section, rapid pace of technology change, ability of the Government to use best products available in the market at any point in time, and openness required to ensure best cost benefit during open procurement are key drivers to ensure vendor neutrality. When a system is architected using open hardware or software components available from multiple vendors, naturally Government benefits from healthy competition among vendors and allows adoption of best technology available in the market.

In addition, a project of this scale has never been attempted before and hence it was critical to ensure complete openness to system architecture. This allows UIDAI to use best-in-class components at cheapest rate and most importantly de-risk the project by using multiple providers when required. Openness also allows UIDAI to replace a particular component if required without affecting whole system.

All key components of Aadhaar system are fully open and are built using open source components running on top of commodity hardware. This is achieved through adoption of open standards, extensive use of open source, and use of commodity and heterogeneous components at software and hardware level. In addition, whenever required, certification is mandated to ensure vendors and providers are indeed compliant to standards and specification laid out by UIDAI.

3.2.2.1 Use of Standards, APIs, and Open Source

Keeping entire system completely open without use of proprietary interfaces and technologies is the only way a program like Aadhaar can have longevity and ensure continuous adoption of best-in-class technology. Openness comes from use of standards, open source, and creating vendor neutral APIs and interfaces for all components.

Aadhaar system is entirely built using open source components and takes heavy advantage of international open standards such as ISO biometric standards, data representation standards such as XML, JSON, Protocol Buffers, security standards such as 2048-bit PKI, AES-256, LDAP, messaging standard AMQP, open protocols such as HTTP, and so on. Aadhaar system is built using widely adopted open source components and uses Java as the primary programming language to build all applications.

Use of open APIs addresses two primary goals – loose coupling of components allowing independent evolution of each component without affecting the other, and having a vendor/provider neutral layer allowing use of one or more providers and replacement of a provider with another without affecting other parts of the system. In addition to the above goals, having API driven approach allows test automation for automated regression testing, continuous re-factoring and tuning within an implementation, and better component level versioning and lifecycle management.

3.2.2.2 Use of Commodity Hardware

When building a very large system requiring massive compute and storage, there are two ways to try achieving it. One way to meet the above goal is to use

proprietary or specialized hardware tuned for specific purpose. This approach, although may give faster short term benefits, completely locks architecture in that specific hardware and technology and will not allow adoption of newer and cheaper technologies in the long run. Also, typically such specialized or proprietary technology forces upfront investments and on-going maintenance from the same vendor forever. On the other hand, if a system is built using open architecture and commodity hardware (one that is cheaply available from several vendors) that allows adoption of newer technologies and cheaper hardware whenever required. In the long run, such open systems work out to be most efficient and cost effective for changes, upgrades, and on-going maintenance.

Aadhaar system is completely built using an open commodity hardware components using several cheap blade/rack servers on x86 platform running 64-bit Linux. Such open scale-out architecture allows UIDAI to procure latest blade servers from any vendor at the best price whenever required. Similarly, storage layer also does not depend on any specialized hardware and takes advantage of heterogeneous storage arrays having cheap SATA disks from multiple vendors. Network backbone and other hardware deployed with UIDAI data centres are based on open standards having multiple vendors capable of providing them at competitive rates.

3.2.2.3 Use of Multiple Providers

Another principle Aadhaar system has used across the system is to bring multiple providers using a common API/Standard to avoid single vendor lock-in and ensure heterogeneous architecture across. This is reflected at hardware level, having servers of different capacity from different vendors work a common unfired compute grid, having storage of different types from vendors, network and security appliances from multiple vendors, and so on. Same multi-provider concept is also used at software level, having multiple biometric matching engines, multiple local language engines, having multiple data stores, having multiple certified vendors providing biometric capture equipments, etc all using common APIs defined for each of those components. Use of a common API and having multiple providers allow constant performance comparison, cost comparison among providers, and eventually entire system of UIDAI and various departments and agencies benefiting from such healthy ecosystem.

3.2.3 Security and Privacy

Security and privacy of data within Aadhaar system has been foundational and is clearly reflected in UIDAI's strategy, design and its processes throughout the system. UIDAI has taken several measures to ensure security of resident data, spanning from strong end-to-end encryption of sensitive data, use of strong PKI-2048 encryption, use of HSM (Hardware Security Module) appliances, physical security, access control, network security, stringent audit mechanism, 24x7 monitoring, and measures such as data partitioning and data encryption.

It is very important that all personal data (any PII) collected for the purpose of UIDAI be provided significant protection across UIDAI and its ecosystem. A National data privacy and data protection legislation will indeed help in this effort and UIDAI has been strongly in support of such a law. However, in the absence of such legislation, the UIDAI has ensured that the resident data is handled with the utmost care within its own and partner domains and follows some of the major principles of data privacy/protection recognized by countries that have already enacted such laws.

At the strategy level, following design considerations are given:

1. Keeping minimal data that is required to uniquely identify a person and to provide identity based services.
    a. The UIDAI restricts itself to the collection of the minimal amount of personal information (as decided by the DDSVP Committee chaired by Shri. N. Vittal [8]) just for the purpose of issuing a unique identity.
    b. The UIDAI does not collect or store any additional personal information or linking data, such as PAN number, Driver's License numbers, caste, income, etc..
2. System is designed to allow domain specific applications and database to be built as a layer on top of Aadhaar by the respective owners of those systems.
    a. This design eliminates Aadhaar system having all domain specific transaction data and hence supporting data federation across

many provider databases rather than centralization into a common database.

b. Allows clear separation of duties and keeps individual databases specific to its purposes instead of one large monolithic all-inclusive database.

3. Aadhaar Authentication "only" responds with a "yes/no" answer, not revealing any Personal Identifiable Information (PII) [13]. Similarly, e-KYC service works only under resident authorization through online Aadhaar authentication (authentication is required every time) hence ensuring residents are in control of their personal data.

3.2.4 Scalability

Aadhaar system requires providing unique identity to more than a billion people. With an aggressive target of reaching 600 million Aadhaars in a short span of 4 years meant that about 1 million Aadhaars need to be generated every day. Processing of every enrolment requires matching 10 fingerprints, both irises, and demographics with every existing record in the database. Currently, with the Aadhaar database at 600 million, processing 1 million enrolments every day roughly translates to about 600 trillion biometric matches every day. On the other hand, authentication is expected to perform within sub-seconds even when authentication volume is few 100 million requests every day.

All this requires entire system and supporting processes to scale to massive levels. Scalability comes from fundamental design and architecture of the system rather than an afterthought. Aadhaar system has already demonstrated the scale by reaching 600+ million Aadhaars, processes 1 million enrolments every day, manages about 4000 terabytes of data, and has deployed authentication services to handle 100 million authentications every day.

3.2.4.1 Scalable Technology

For achieving such massive scale and that too within the constraints of e-Governance systems, it is critical that technology choices are kept simple, open,

multi-vendor, and standards based. Following are key considerations given at architecture level from the beginning to ensure technology scale:

- **Horizontal scale for compute and storage** – Architecture must be such that all components including compute and storage must scale horizontally to ensure that additional resources (compute, storage, etc) can be added as and when needed to achieve required scale. This also ensures that capital investments can be made only when required.
- **Data partitioning and parallel processing** – For linear scaling, it is essential that entire system is architected to work in parallel with appropriate data and system partitioning. Data partitioning (or sharding) is integral to ensure as data and volume grow, system can continue to scale without having bottlenecks at data access level. Choice of appropriate data sources such as RDBMS, NoSQL data stores, distributed file systems, etc must be made to ensure there is absolutely no "single point of failure" in the entire system.
- **Loose coupling through open API and messaging** – Whenever the logic is long running (taking potentially hours or days, as in the case of enrolment), it is critical that it is broken into small components and wired them through an asynchronous workflow. Such design allows each component to do its job fast, release resources, and handle failures at micro level. It also allows each of these components to be run across a cluster of machines and allow horizontal scaling. But, such asynchronous design requires each component to be designed through a published open API and loosely couple them through a messaging layer. Such API wrapped, black-box style approach also allows component level tuning and re-factoring to achieve required performance and scale.
- **Use of open source and commodity hardware** – Keeping entire system completely open without use of proprietary interfaces and technologies allows component level modifications when faced with scaling issues. If a component is found not performing, instead of depending on a vendor to fix proprietary technology, UIDAI can go ahead modifying either existing open source component or use an alternate one. Since these are wired through APIs, local testing (for regression) is enough to ensure functionality is intact. Similarly, when building a very large system requiring massive

compute and storage, use of commodity hardware (one that is cheaply available from several vendors) and horizontal scalability allows scaling by adding cheaper hardware into the cluster whenever required.

3.2.4.2 Scalable Ecosystem

In addition to technology at the backend, it is necessary to scale field operations and application adoption to ensure Aadhaar enrolments and authentication take off. As described in first chapter (see section '*Ecosystem Approach*'), UIDAI has created a large ecosystem of providers to manage the overall scale and cost of this entire project. UIDAI ecosystem consists of registrars, enrolment agencies, biometric device providers, certification agencies, training agencies, field operators & supervisors, authentication agencies, IT product & service providers, and application developers.

Such multi-provider ecosystem cannot be implemented at large scale efficiently unless two key areas are addressed: "*Technology & Process Standardization*" and "*Measurement, Monitoring, & Continuous Improvement*". UIDAI has defined technology, people, and process standards across the board and enabled training and certification agencies to train and certify to those standards to ensure entire ecosystem works uniformly. When working with 3rd party organizations that are part of ecosystem, it is essential that entire system is measured using data, decisions are made based on data, and monitoring and continuous improvement schemes are put in place. Highly granular metadata (or process data) must be collected throughout the system to ensure quality is measured systematically and feedback is given to improve any specific issues that are identified.

3.2.5 Interoperability

Because Aadhaar system is conceived as an 'Identity Platform' on which many applications will be built, it is critical that all 3rd party interfaces be fully interoperable without any affinity to platforms, programming languages, network technologies, and such. Such open interoperability is an absolute requirement for Aadhaar to be widely adopted as a national identity platform.

In addition, even within the Aadhaar application, all components must be loosely coupled using open interfaces (APIs) ensuring interoperability across components and sub-systems. This is especially critical with respect to biometric de-duplication engines and biometric matching engines where such specialized components are procured and used as plug-in-play components within the overall application. Also, given the fact that entire enrolment and authentication devices on the field are procured and managed by 3rd party agencies, it is critical that any device that is certified is able to interoperate with Aadhaar application seamlessly.

3.2.6 Manageability

Aadhaar application is expected to handle millions of enrolments leading up to a billion, 100's of millions of authentications every day, a few 1000 terabytes of data all in a reliable and manageable way. It is inevitable that in such large scale compute environment, something or other fails regularly; be it a hardware failure, network outage, or software crashes. Assuming otherwise (that nothing fails) is naive and it is essential that the application architecture handles these failures well, be resilient to failures and have the ability to restart, and make human intervention minimal.

The entire application must be architected in such a way that every component of the system is monitored in a non-intrusive fashion (without affecting the performance or functionality of that component) and business metrics are published in a near real-time fashion. This allows data centre operators to closely watch the application behaviour and performance through a Network Operations Centre (NoC) at a granular level.

Application architecture should also allow specific components to be watched very closely through a component level debugging scheme. Such debug logging should be limited to specific components and for a very short time so as to enable engineering team to analyze any specific issue arising in production and troubleshoot.

3.2.7 Data Driven Decision Making

A large multi-provider ecosystem created by UIDAI can only be managed efficiently by measuring process data at a high degree of granularity, creating well defined metrics from this process data, and creating feedback loop for these insights and learning to be shared back to the ecosystem for continuous improvement. When working with 3rd party organizations that are part of ecosystem, it is essential that entire system is measured using data and decisions are made completely based on data. Highly granular metadata (or process data) must be collected throughout the system to ensure quality is measured systematically and feedback is given to improve any specific issues that are identified.

Thus the objectives for UIDAI are as below, keeping in mind the large ecosystem, comprising primarily of external partners and a very lean UIDAI structure:

1. **Drive decision making based on data analytics**: The analytics module within Aadhaar system should be such that stakeholders can easily include data and insights in their operations on a regular basis. Processes must be in place to drive a feedback loop to the overall organization including partner ecosystem to drive continuous improvement.
2. **Empower self-improvement**: The analytics function should also help stakeholders to improve by themselves. Tools, data and platform should be created to be able to help stakeholders analyze their own performance and operational metrics themselves.

3.2.8 Platform Based Approach

It is critical that a platform based approach is taken for any large scale application development. Building an application platform with reusable components or frameworks across the application suite provides a mechanism to abstract all necessary common features into a common layer. Open APIs designed to be used for internal and external purposes form the core design mechanism to ensure openness, multi-user ecosystem, and specific vendor/system independence.

4 Application Architecture

This chapter covers application architecture and design for each of the modules described in chapter 2 ("*Aadhaar Application*") based on principles described in chapter 3 ("*Architecture Principles*"). Design details cover how each of the key components are architected, technology used within each of those, and the reasoning behind the design.

4.1 Overall Architecture

Aadhaar application is primarily written in Java™ language using open source components and frameworks. Application is built in tune with all the architecture principles described in earlier chapter.

Aadhaar system currently has already issued more than 600 million (60 crores) Aadhaar numbers, processes 1+ million (10 lakhs) enrolments per day amounting to 600 trillion biometric matches every day within its system, deployed authentication services capable of handling 100 million (10 crores) authentications every day, and has over 4000 Terabytes (4 Petabytes) of data across UIDAI's data centres, all using an open commodity computing architecture and built entirely using open source software components.

At infrastructure level, application runs on commodity multi-core blade servers with 8086 64-bit architecture on a 10Gbps network backplane. Entire application runs on 64-bit Linux OS. Most of the large scale storage is SATA, with some specific smaller storage having SSDs. Both compute and storage infrastructure is

heterogeneous (having different capabilities procured at different times from different vendors based on RFP) and works uniformly as a compute/storage cluster.

As described before, Aadhaar application can be divided primarily into its core modules – enrolment, authentication, e-KYC, and supporting modules (portals, BI, fraud detection, CRM, etc). Rest of the chapter describes how these modules are architected to meet their specific functional and non-functional needs.

4.2 Enrolment Module

As described earlier, at a high level, this module handles initial enrolment of residents into Aadhaar system, validations, biometric de-duplication, Aadhaar generation, printing of letters, and all subsequent lifecycle changes such as demographic data updates, biometric updates and other related workflows.

Aadhaar system currently has about 30,000 enrolment stations deployed across the country by various registrars. Currently there are about 100,000+ certified operators and supervisors who are trained and certified to operate the enrolment station. On an average, each station enrols about 50 people per day amounting to 1 million enrolments per day on the field.

4.2.1 Enrolment Client

Enrolment client works mostly offline and has all the features necessary to capture resident demographic details, do local validation, local transliteration, biometric data capture, biometric data quality check, and capturing of necessary audit details such as operator biometrics, location & time. Enrolment client software also has built-in security features such as in-memory data encryption, encrypted data storage, export, etc.

The enrolment client is implemented as a desktop application with a rich user interface and deployed to a large number of computers on the field (enrolment stations). It works on Windows and Linux laptops and can connect any of the certified biometric sensors, standard scanners, and printers via Aadhaar Biometric Capture Devices Interface Specifications [19]. Enrolment agencies use this Enrolment Client software to enrol residents into the system on the field and upload data onto the server as a batch.

Following sections cover logical and physical implementation of various components within enrolment client so as to meet all business requirements.

4.2.1.1 Client Architecture and Implementation

Diagram below represents the logical component diagram for enrolment client.

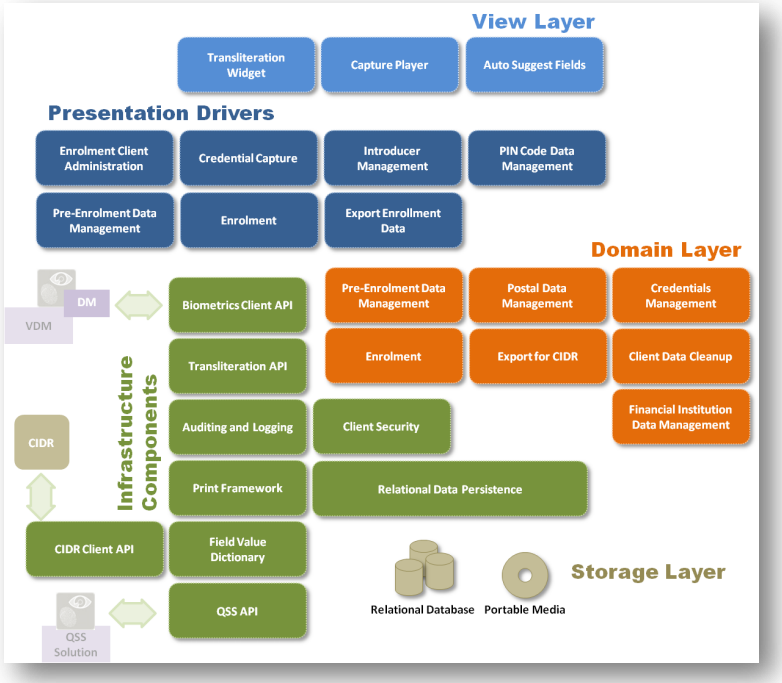


Figure 6: Enrolment Client Component Diagram

Entire client is written in Java and single client code base is certified on both Windows and Linux stack. Although most agencies use Windows laptops, a few states such as Kerala use Ubuntu Linux based laptops. UIDAI ensures that single client version runs on multiple OS platforms. In terms of design, each of these components conforms to a well defined interface that serves as the contract for collaboration between other components within the client.

4.2.1.2 Storage Layer

The storage layer comprises of multiple, logically distinct, data stores implemented using a combination of relational database tables and local file-system entities. In order to reduce the overheads during application setup and maintenance, the relational database must be an in-process database. This means that the database must exist within the memory space of the enrolment client and not as a separate process or service.

In the interest of security, the resident's profile is maintained as in-memory objects during the life-cycle of an enrolment session. This includes all demographics, biometrics, user credentials (standard data types and biometrics) and audit information associated with the resident's profile. These in-memory objects are also protected via HMAC values to ensure no unwanted alterations/tampering are done while in-memory and integrity is maintained.

Once the enrolment session is complete for a resident, the whole in-memory object tree is checked for integrity and saved as encrypted enrolment packet. Each enrolment packet is encrypted using a 2048-bit asymmetric key and written to a single file on the local file system. The internal format of the enrolment packet is identical to the import format expected by CIDR. This is necessary since the encrypted enrolment packet, once created, cannot be opened on the client side or during transit for further modifications. During export to CIDR, each enrolment packet is moved to the removable media using opaque file operations without further interpretation and re-packaging of file contents.

4.2.1.3 Domain Layer

The domain layer incorporates logic that pertains to capture of residents' profiles and the storage, retrieval and cleanup of such information against persistent storage. Elements of the domain layer are not reusable across business scenarios. There are also no visual considerations (with respect to application behaviour at user interfaces) that go into the design of this layer.

Pre-Enrolment Data Management

This module works in conjunction with suitable presentation drivers and other helper entities (e.g. CSV reader, HTTP reader) to import pre-enrolment data. Once loaded, this data cannot be modified on the enrolment client. Basic validations are performed at the time of addition to ensure type consistency, check for junk data, removal of duplicate entries, etc.

Management of Region Code Data, Credentials, and FI Data

This module helps to import master data information into relational database tables. Once loaded, this data cannot be modified on the enrolment client. Basic validations are performed at the time of addition to ensure type consistency, check for junk data, removal of duplicates, etc. The module allows search and retrieve operations to be performed on previously loaded region code data based on defined parameters (e.g. Postal pincode, district code, etc).

Enrolment Component

This component provides behind-the-scenes capabilities to securely store resident profile information including both demographic and biometric data. A unique enrolment ID is generated for each resident profile stored. The profile information is distributed across file system storage and application database as described in storage section above. Note that the enrolment module does not directly interact with any of the biometric devices attached to the enrolment client. This glue (control and data flow) is provided by corresponding presentation driver entities.

Enrolment Data Export

This module provides the necessary capabilities to export encrypted enrolment packets (maintained by the enrolment module) into removable media for physical transfer and upload to CIDR. The export process comprise largely of opaque file-

level operations (copy, move, delete) with additional data transformations into alternate packet structures.

Enrolment Client Data Cleanup

The enrolment client data cleanup module is responsible for removal of resident's profile from the enrolment client databases subsequent to export and receipt of the corresponding data by CIDR. This module makes use of the CIDR Smart Sync API component of the infrastructure layer for communication and data exchange with the CIDR.

4.2.1.4 Infrastructure Components

Infrastructure components are entities incorporating common features that are reused across multiple domain layer entities and presentation drivers. These also include components that allow integration with external devices (e.g. biometric devices) and backend systems (e.g. CIDR).

Biometrics Client API

The Biometrics Client API reduces all interactions with the biometrics layer to the following high-level procedures, data objects and callback interfaces:

- Method to ping the Device Manager at intervals defined in Device manager's Connect response.
- Register a callback interface to receive notifications on device availability and auto capture of biometric samples.
- Method to request for video stream capture on any device.
- Method to query device capabilities and capture biometric samples.

All other entities across logical layers within the enrolment client interact with the biometrics layer only through this client API conforming to and implements the client-side specifications for the Aadhaar Biometrics Capture Device API [19]. Note that the UIDAI biometrics capture API itself provides a common abstraction for interacting with any biometric device independent of the solution provider and type of data captured.

QSS API

The QSS API (biometric quality check) reduces all interactions with a third-party QSS library to the following high-level procedures and data objects:

- Given a biometric data in standard image formats, retrieve a list of quality parameters along with scores and feedback to improve overall quality.
- Given a biometric data in standard image formats, segment the image into relevant portions (as rectangular co-ordinates) with qualitative information on each segment.
- Given a biometric data in standard image formats, retrieve information about the left or right sidedness of the image.

Transliteration API

The transliteration API defines a standard service provider interface that must be implemented by any external solution capable of providing transliteration between English and the officially recognized Indian languages. It is primarily used by the transliteration widget of the view layer to provide side-by-side view and capture of resident's profile in both English and a local language of choice.

From a functional perspective, the transliteration API requires the following capabilities of any implementing service provider:

- Ability to configure the transliteration engine to output in a target Indian language using data dictionaries, language models, etc.
- Given a text string in English, convert the same into a Unicode text corresponding to the Indian language specified.
- Given a text in English, provide a list of transliterated texts in the target Indian language. This list can be used to select an alternative transliterated text, in case where the default transliteration is not appropriate. The list is sorted based on an internal scoring system for relevance and accuracy.
- Reverse transliteration from a local language to English. This includes providing the best match as well as a list of possible alternatives. The list of alternatives is sorted based on an internal scoring system for relevance and accuracy.

- Edit distance algorithms to validate pre-filled local language fields (e.g. from pre-enrolment data) against transliterated alternatives.
- Provide an on-screen keyboard that allows typing and character-by-character correction of the transliterated text.

GPS Client API

The GPS Client API is used to interface with an attached GPS device to capture the approximate geographical co-ordinates of enrolment operations. This GPS information is associated with additional mapping between an enrolment drive and a geographical location and used to perform business analytics.

Auto-Suggest Dictionary

This key-value dictionary component is used to provide auto-suggest capabilities for frequently used fields on the demographics capture form of the enrolment client. Note that this component in itself does not appear anywhere on user interface screens but are used within the auto suggest fields of the view layer. Specific capabilities of this component are as follows:

1. Maintain a dictionary of frequently used terms on a per-field basis. The data in this dictionary is not-preloaded but derived from previously entered values in the corresponding field.
2. Data in the dictionary is temporary in nature and should be cleaned up on a periodic basis. This should be a manual operation to be performed using suitable configuration screens.
3. The dictionary should have a maximum size with respect to the number of terms stored and follow a 'most recently used' (MRU) policy to retain frequently used terms.
4. Each term must have an associated weight that determines its position within the list of auto-suggest options. This weight is ascertained based on the number of times the term has been entered previously.
5. The dictionary is maintained for text in both English and local languages. For local language fields, auto-suggest values from the name dictionary and from the transliteration API must be combined and visually displayed in one drop-down.

Smart Sync and other Client to CIDR APIs

The Client Sync API (Smart Sync) is used by the enrolment client to communicate with the CIDR as and when the client application comes online within mandated sync frequency. This API is primarily used to cleanup exported resident's profiles that are still residing with the client application. A subset of this protocol is the enrolment station registration API that is used to register the machine with the server as part of the post-installation station configuration activities.

Online communication between enrolment client and CIDR takes place over HTTP and accesses remote services that expose a REST interface. SSL is used as the transport layer encryption mechanism. The POST method of HTTP is used to transmit data from the client end. For communication pertaining to client registration, an XML document containing unique machine identifier, enrolment agency code and station number is used as the body of the POST message. In case of successful registration, the CIDR should respond with a set of security tokens to establish trust relationships for all future client-server data exchange.

In addition, smart sync has the mechanism to enforce rules from the server such as force upgrade of client versions, when maximum allowed un-exported packet count is reached, lock it until data is exported, remove black-listed operators from credential data, etc.

Print Framework

This module provides support for print-related capabilities for all modules across all layers within the enrolment client. Printing in both English and local languages are supported.

The on-paper layout of printed content is template based: a separate template exists for each type of print operation (e.g. print receipt after successful enrolment). These templates form a part of the application executable. Changes to the template contents are centrally distributed as incremental updates and incorporated into the application via the automatic update process.

Client Security

This module deals primarily with certificates and key management to encrypt the enrolment packets that are exported. Key management makes use of PKI technologies with two sets of public keys being maintained for data exchange with CIDR and registrars respectively. CIDR packets are secured by encrypting them using a 2048-bit UIDAI public key. Each such public key is made available in the format of a valid X509 certificate. The validity of these certificates is separately checked on the client-side prior to extraction of the corresponding public key.

During packet encryption, a single public key is randomly selected from a set of valid public keys. The corresponding key store is embedded into the enrolment client and forms a part of the application runtime binaries.

Note that no private key is ever stored or maintained in the client. Client only has the ability to encrypt the resident data and has no mechanism to decrypt until the packet is securely transported to within UIDAI's data centre.

Auditing and Logging

This module is used to maintain a runtime log of the enrolment client for subsequent debugging purposes in case of application crash or poor performance. Audit information, pertaining to application usage, is also maintained separately. Some of this audit information is exported as part of enrolment packets. The purpose of the audit information is to perform on field usage analytics.

Aadhaar system implements data-driven analytics and continuous improvement of its processes. To enable this for the enrolment process, UIDAI has built-in several features within the enrolment client providing metadata related to the enrolment. For example, every enrolment packet is reviewed by a supervisor for data quality (review audits are captured electronically) and signed as required which means every enrolment is traceable in terms of "*who*", "*when*", "*where*", "*under which agency*", "*under which registrar*", "*who reviewed it*", etc. In addition, several metadata elements such as "*how long operator spent on demographic data screen*", "*how many times a fingerprint was captured*", "*how many corrections were done*", etc. are collected as part of every enrolment packet for analysis of operator actions and performance. This data is used for providing continuous improvement

feedback on data quality to the registrars and enrolling agencies using UIDAI's business intelligence platform. Audit module captures all these data including screen transitions, crash reports, number of times it restarted, etc and added to audit metadata which is then sent to CIDR and fed into analytics sub-system.

Local Biometric Verification (LVS) API

This module simplifies integrations with the third party services used for local biometric verification of operators, supervisors or introducers. The local verification service also wraps the interaction with the server in order to verify a user. LVS uses "Aadhaar Authentication" to "on-board" operators and supervisors within its DB so that local offline verification can be done after on-boarding whenever required.

4.2.1.5 Views and Presentation

The View Layer comprise of visual elements that together constitute the end-user interfaces of the enrolment client. Most of the visual elements (e.g. text fields, radio buttons, tables and buttons to name a few) are standard in nature and supported by most windowing toolkits such as Java Swing.

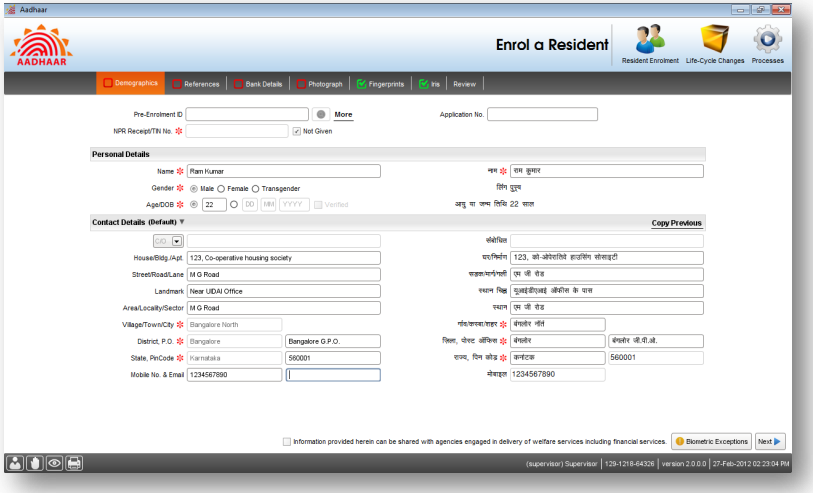


Figure 7: Enrolment Client Screenshot

The following are a few custom widgets that are specifically required to address the user experience requirements of the enrolment client.

Transliteration Widget

The transliteration widget makes use of the Transliteration API to provide a side-by-side view of text data in English and in a local language of choice. It also provide local language keyboard, suggestions, etc. Screenshot of the keyboard is given below.

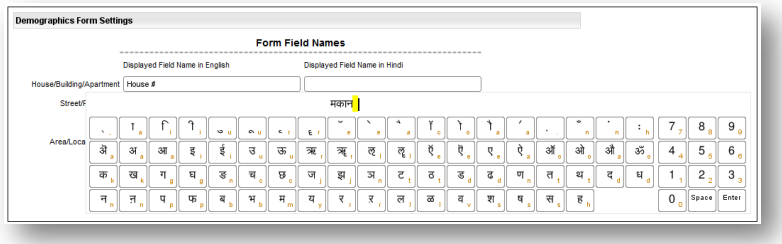


Figure 8: Local Language Keyboard

Capture Player

Figure 9: Biometric Capture Screenshot

Above widget is a lightweight video player to render frames obtained from any biometric device. Dynamic feedback received from the device (e.g. 'move the finger' 'press harder', etc.) is shown on the video rendered. Dynamic quality of biometric sample is displayed alongside the video display area.

4.2.1.6 List of Technologies Used

Technology	Intended Usage
Java Standard Edition 6.x	Programming language and runtime environment.
Apache Derby 10.5.x	Embedded relational database to host client-side data (master and process data). Encrypted resident data including biometrics is NOT stored here.
Apache iBatis 2.3.x	Lightweight relational data persistence framework to move client data between relational database tables and in-memory objects.
Spring Framework 2.x	Dependency injection framework to integrate with each other the entities within different layers of the enrolment client.
Java Swing Framework	Windowing toolkit that is used to create the enrolment client user interface screens.
Aadhaar Biometric Capture API compliant VDMs	Vendor Device Manager (VDM) that is compliant to Aadhaar Biometric Capture API is provided by specific biometric device vendor. These are certified by STQC for quality and compliance.
Cognirel's and CDAC Transliteration and Matching Library	Transliteration engines based on Aadhaar Transliteration API. Any engine compliant to the API can be used as a plu-n-play engine and configured for one or many languages.

4.2.2 Enrolment Server

Enrolment server module handles all the functionality necessary to manage enrolment packets, validate them, do necessary quality checks, and most importantly orchestrate biometric de-duplication to ensure uniqueness before assigning a unique identifier to the residents.

4.2.2.1 General Architecture Strategies Used

The enrolment server solution architecture follows certain architecture principles and strategies that have been adopted considering the architecture goals and constraints of the system.

Stateless nature of Aadhaar generation flow stages

Stateless nature of application components helps in addressing scalability and availability needs of the architecture. All stages of the Aadhaar generation flow are designed to be stateless and message driven in a request-response model i.e. each stage is given complete data that it requires for execution. The stages follow the "*Command*" design pattern. Each stage is autonomous and connected to the next one by the infrastructure and not hard-wired in code. This approach to software design that decomposes a complex, event-driven application into a set of stages connected by queues is based on the "*Staged Event Driven Architecture*" (SEDA).

All stages in the flow are designed and implemented as Plain Old Java Objects (POJO's). This approach serves purposes of independent testing of each flow stage and removes many dependencies on the execution runtime. Each stage can be instantiated and messages passed to it using method constructs of the POJO during unit testing and thereby even decimate need for queues that link stages together.

Re-submission of data within stages

The Aadhaar generation flow stages are multiple in numbers and involve execution of many tasks. This spans OS processes, machines, and module boundaries. It is therefore quite likely that some of the execution stages might become unavailable due to machine failures or software exceptions. Failures may also occur due to large and sustained loads on the system.

The architecture takes this possibility into consideration and follows the design strategy of permitting re-submission to all stages in the flow. This approach permits stages to fail fast and resume execution when recovered. Re-submission strategies normally adopt a check-pointing approach where each stage saves required state into a data-store which is then to resume execution when recovery happens. The Aadhaar generation flow uses an RDBMS schema as the persistent store for check-point data. The first stage in the Aadhaar generation flow initiates the check-point record for the flow and subsequent stages update this data as the flow progresses. A sweeper job scans the check-point database for stale and zombie records and resubmits them to the next stage of the flow i.e. the stage succeeding the last one that was successfully completed and check-pointed.

Definition of vendor neutral API for integration with COTS solutions

Aadhaar generation process is dependent on third-party and often COTS products for realization of certain stages in the flow. Aadhaar enrolment server follows a multi-vendor strategy (as articulated in architecture principles) for servicing such needs. However, not all interfaces have industry standards and standard APIs for integration. The architecture therefore follows the following strategies for such integration needs:

- Use of messaging middleware as the integration channel – Asynchronous messaging and use of message correlation identifiers enables loose coupling with the vendor solution and isolates the stages from the actual deployment and implementation of the required task. This approach also helps tune queue depths and consumption rates depending on the throughput available from the vendor solution deployment.
- Use standardized API published by UIDAI that will be supported by all participating vendors – Messages passed to the vendor solutions follow an XML structure as standardized by UIDAI in consultation with the vendors. The API is standardized and published as XML schemas (XSD) that are then implemented by the various vendors over the messaging middleware transport interface.

Location and network agnostic execution

The approach of connecting stages using message queues has another benefit – that of using location agnostic message queues that can span data centres. A geographically scaled out deployment of such nature helps in disaster recovery and load distribution across the entire logical cluster of machines. Location unaware execution requires stateless behaviour as mentioned above. The messaging middleware supports routing tables that are shared across the entire cluster, and delivery of messages published at one node to a queue residing at another node is seamless.

Solution design to handle failure of Nodes, Processes and Storage disks

The deployment infrastructure for the enrolment server components is expected to grow up to hundreds of nodes. Given the high loads on the infrastructure, it is but natural for nodes and processes to fail. Solution strategies like stateless nature

of generation flow stages, re-submission of stages and location and network agnostic execution defined above ensure that the system is functional in the event of large scale infrastructural failures.

However, it is imperative that failed instances recover, are restarted at the earliest so as to resume processing of the flow stages. Recovery requires the messaging infrastructure to either support discovery of nodes or for nodes to join logical topologies and named clusters.

In addition, the architecture also needs to address data storage needs. During peak enrolment, the system receives 1 million enrolment packets in a single day. The large size of each resident's enrolment data adds up roughly to 30 TB of I/O for a whole day's load. The nature and structure of large volume of encrypted binary files indicates need for alternative stores to the RDBMS. A DFS (Distributed File System) such as Hadoop that works with commodity grade storage disks is a viable option in spite of the network latency involved in data block distribution across the disks and the cost of maintaining and running a file system over and above the operating system's file system. The DFS deals with the issue of potential storage disk crashes and failures by maintaining multiple copies of data blocks. In the DFS, each file is broken into multiple data blocks which are then distributed across storage disks – often in multiple copies.

Loose coupling of integration interfaces

Integration interfaces are isolated using messaging middleware where possible as described in the solution strategies above. However not all interfaces fit into the middleware based asynchronous integration pattern. An example is the callback from biometric de-dup servers for enrolment packet data i.e. API request for reading the biometric data packet for the reference number submitted for de-dup and insert into the vendor solution's gallery. Interfaces of this nature follow a synchronous pattern as opposed to asynchronous model typical of messaging. HTTP based implementations are used for such interfaces as opposed to providing Java API that runs inside the vendor solution's process space. This approach, while adding IPC overhead, provides process separation, cleaner separation of responsibilities and better management of code changes to vendor solution vis-à-vis the interface.

4.2.2.2 Enrolment Server Logical View

Aadhaar enrolment server components are highly scalable to handle million+ enrolments every day and is built to manage data stores in 1000's of terabytes. Entire enrolment workflow is broken up into many logical stages.

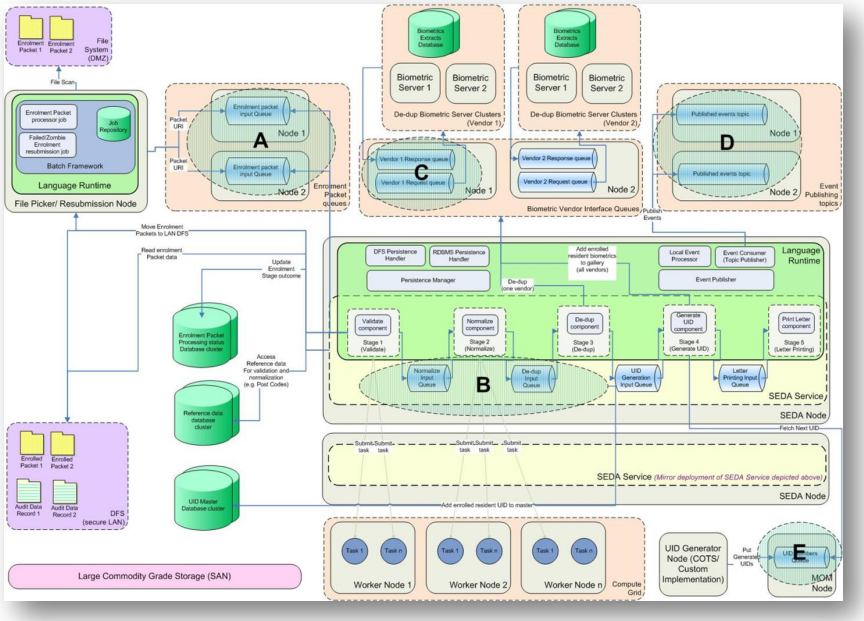


Figure 10: Enrolment Server Logical View

High level flow across the various components within enrolment server is as follows:

- Enrolment data packets are collected from the various Aadhaar enrolment clients and uploaded by authorized personnel to designated folders in the DMZ through the Upload Client software provided by UIDAI. These are scanned for viruses before making it available for subsequent processing.

- A file picker batch job periodically scans the DMZ and queues the URIs of enrolment packets to a load balanced set of dedicated queues.
- Multiple SEDA nodes that run the enrolment processing flow stages are deployed to process incoming enrolment packet URIs. The first stage of the flow is registered as a listener to the enrolment queues and therefore picks up the file URIs. This stage inserts enrolment and its metadata to the following data stores:
    - A record into the enrolment packet processing status RDBMS for the URI being processed.
    - A record into the demographics and photo tables in RDBMS which acts as a cache for subsequent stages of the SEDA.
    - A copy of the entire encrypted enrolment packet into the archival system and a copy to DFS which acts as file cache. The packet is then deleted from the DMZ. While archiving checksum of the file is computed and verified to ensure entire encrypted enrolment packet file is moved in full.
- The various stages in the SEDA flow listen to designated queues for incoming messages that trigger and provide data for stage processing. Each stage is implemented as a POJO and may in turn define a set of tasks to be executed to complete the stage. The tasks are dispatched to the compute grid and execute there.
- Integration with the multi-ABIS de-dup servers is via the ABIS middleware. The ABIS middleware is a SEDA node that has a couple of queues – one outbound and another inbound. All messages have a unique reference number populated by the Enrolment server which is replayed back in the response by the biometric server. The reference numbers are used for message correlation. De-dup (Identify) and add-to-gallery (Insert) requests containing the reference numbers are queued to the biometric server queue by the ABIS middleware which has policies for routing and response handling. On receipt of the enrolment message, the biometric server makes a call to an HTTP service hosted by the Enrolment system for retrieving biometric content from within enrolment packet identified by the reference number in CBEFF format.

- The biometric server sends a response message with status indicator for the request i.e. de-dup or add-to-gallery. The SEDA stage receiving the biometric server response routes the message to Aadhaar number generation stage. Error or manual verification flows are handled by separate stages and using an exception handling workflow.
- The Aadhaar number generation stage interfaces with a queue containing generated Aadhaar numbers and picks the first available number.
- The generated Aadhaar number is then linked to the enrolment record identifier and inserted into Aadhaar (UID) master (a 100-way sharded RDBMS store).
- Processing then moves to the next stage of letter printing. In this stage the letter is marked for printing. Actual letter generation is done by a scheduled batch job.
- Persistence of operational data like enrolment status records, demographic information of resident and audit records are initiated via the flow framework. Data may be stored in either of RDBMS, document database, columnar database or DFS depending on the persistence strategy for respective data model.
- The architecture also supports an Event framework that allows stages and its constituent implementation classes to publish events relevant to the processing stage and the data being processed. In asynchronous event consumption, event data is published to messaging server topics for distribution to subscribers. The BI sub-system is one such subscriber of enrolment server events.

The enrolment process is automated from receipt of enrolment packet leading to letter printing post Aadhaar number generation. However manual intervention is needed in stages that are integration points with third-party solutions. One such example is the Manual Adjudication after biometric de-dup stage where candidate duplicates may be returned from the biometric servers requiring manual intervention for de-dup. This requirement indicates the following solution needs:

- Aadhaar generation flow must support re-submits to any of the stages. Each stage must therefore write checkpoint data that may be read, tracked or used otherwise.
- Ability to determine the count and reference numbers of enrolment packets being processed in each stage. This information may then be used for manual intervention.
- Ability to pause execution of the Aadhaar generation flow to permit administrative intervention.
- Ability to collect exception and failed enrolment data from the respective stage in the Aadhaar generation process which may then be presented for manual analysis and action

Each stage in the enrolment process generates useful information that may be used in Business Intelligence (BI) – for e.g. quality and response time information for de-dup requests from the various biometric vendor solutions. This requirement indicates the following solution needs:

- Provide an event framework that enables various stages to easily publish defined events with relevant event data.
- Isolate event publishers from event consumers to enable easy administration of different types and instances of event consumers.
- Define destination end-points for each event type using URIs that may be used to connect event consumers with publishers.
- Select a suitable transport for delivering events that is location agnostic and can preferably work across data centre networks.

The Aadhaar BI sub-system and the audit framework are two primary consumers of events published from the enrolment flow. Aadhaar enrolment server architecture provides the required data needed by these sub-systems using the event framework.

4.2.3 Enrolment Biometric Subsystem

Aadhaar system deploys 3 independent ABIS solutions adhering to common ABIS API [12] for multi-modal biometric de-duplication. Enrolment server needs to therefore integrate with these solutions using the ABIS API and allocate de-duplication requests as per UIDAI policy of dynamic allocation (which uses on-going accuracy and performance data to decide which solution gets maximum de-duplication requests).

Architectural requirements of multi-ABIS solution are as follows:

- Define integration channel that is generic enough to be supported by multiple solutions and have detailed configurability in defining allocation policies.
- Ability to make optimal read-write of enrolment packet data over the network and across processes considering large data volume of the enrolment packets (3-5MB per resident having a volume of 1+ million enrolments per day).
- Suitable isolation of integration components into processes and nodes – maintain clear interfaces and implementation responsibility between Enrolment server and vendor provided ABIS solution servers.
- Design solution to handle non-uniform success-failure ratios among biometric server solutions during API invocation.
- Collect performance metrics for each biometric server solution and publish the same for analysis for continuous accuracy and performance metrics calculation.

The ABIS Middleware is implemented as a separate SEDA node and is therefore independent of the enrolment server's SEDA node. The ABIS middleware at its core implements behaviour for request-response tracking, handling retries and routing behaviour based on dynamic allocation policies of UIDAI that may be administered on demand. ABIS middleware uses RDBMS to track requests and responses from multiple ABIS providers and orchestrates the de-duplication activity. Once all ABIS solutions have responded, middleware responds with one message back to

enrolment SEDA process which continues to either Aadhaar allocation stage or reject handling stage. In the case of rejection of a de-duplication request by one of the ABIS providers, middleware sends the same to other ABIS providers for re-confirmation. If more than one ABIS provider rejects the enrolment as duplicate with high degree of confidence, it is automatically rejected. Otherwise, it is sent to manual adjudication workflow where a semi-automated process handles this small number of adjudication requests.

Following diagram depicts the multi-ABIS architecture with ABIS middleware orchestrating the insert/identify flows between 3 different ABIS solutions as per UIDAI dynamic allocation policy.

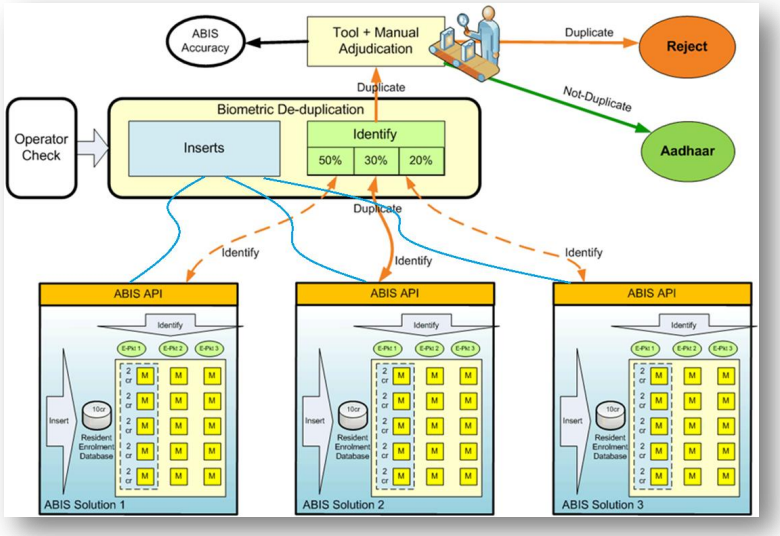


Figure 11: Multi-ABIS De-Duplication

The messaging queues used in this integration are persistent i.e. messages are persisted in order to permit guaranteed delivery even in case of machine failures. Message delivery to the biometric servers requires guarantee as the ability of the

biometric server to handle duplicate add-to-gallery requests might vary between vendor solutions.

All read-writes to the DFS are performed by the enrolment server components. The biometric servers receive only a reference number for the enrolment packet that must be de-duped or added to the gallery. The message to the biometric server also contains an HTTP URL that needs to be de-referenced by invoking an HTTP service which implements reading of enrolment packets for requested reference numbers. This HTTP service provides clear separation of responsibilities between the enrolment server and the vendor biometric de-dup solutions.

4.2.4 Aadhaar Number Generation and Allocation

Aadhaar number follows a well defined format and size as defined by UIDAI [10]. Multiple number generation options exist for implementing this module. Number generation module is independent of the rest of the application and is integrated via a messaging queue where pre-generated Aadhaar numbers are available for the SEDA flow stage to consume on demand. Benefits of this design are:

- 12-digit Aadhaar number (11 digit random number plus a Verhoeff checksum digit) generation throughput is independent of the enrolment packet arrival rate. Random numbers may be generated well in advance instead of on-demand if the implementation is considerably slow.
- Message queue provides a level of isolation between the number generator and the consumer. This allows the generator to be replaced when required.

Number generation uses a pseudo-random number generation scheme (PRNG) with high degree of randomness and generates random numbers derived from a number space of 100 billion (11 digits). As per number generation scheme, UIDAI filters numbers starting from 0 and 1 (kept for future expansion) making the number space to 80 billion. In addition, specific number patterns are eliminated as per UIDAI policy. Filtered patterns include repeating numbers (more than 4 digits), simple sequence numbers, numbers with special sequences (such as '666'), reverse sequences, repeating blocks of 4, etc. Number generator uses a series of configured filters to ensure Aadhaar numbers follow all defined policies.

Note that PRNG algorithm does not guarantee uniqueness across all Aadhaar numbers generated across many months and years. Hence it is important to ensure uniqueness explicitly. Generator component uses an embedded Java database to store and check number uniqueness before pushing the number to the queue where all generated numbers are stored and consumed from. Even with this, to ensure any failure resilience, when finally inserting the Aadhaar number to Aadhaar (UID) Master database, uniqueness is enforced via database unique index. But, this optimistic scheme makes sure 99+% of time, there is no failure. In case of any last minute failure, that number is simply discarded and another number from the queue is picked up and used. In practice, number generator Java Spring batch program is executed in regular intervals (every few days) and generates about 30-40 million at a time and pushes into the queue for enrolment module to consume.

4.2.5 Enrolment Packet Archival

Once Aadhaar number is allocated, Aadhaar master entry is created, and biometric templates (for authentication) are extracted, there is no real need for accessing the raw encrypted enrolment packet file.

Design and architecture needs to cater primarily to handle:

- **Scalability** – System should be able to handle more than 4 billion files (enrolments, corrections, lifecycle updates, and future enrolments for next 50+ years) each of size 3 to 5 MB. This means archive system should handle about 10000 to 15000 TB raw data. Considering the requirement of high availability (HA) and no data loss, 2 copies of data needs to be on disk across geographically separate data centres and 2 copies on tapes (20000 to 30000 TB on disk and same on tape). This storage should allow random access to individual packets based on a unique file ID. This system should be able to support easy replication (across data centres) and file backup.
- **Security** – Packets should always be encrypted on disk. Decryption (PKI-2048 and AES-256) is done only by authorized applications using a Hardware Security Module (HSM) network appliance which is protected. System administration should not have any mechanism to reverse engineer

and deduce packet details by looking at the physical files/data on disk. Access to the system should be easily controlled via standard ACLs. From a network access perspective, entire archive sub-system should be not accessible directly and access to individual files should be controlled and audited.

Aadhaar application uses a concept of unique alias, called RefID, for all enrolments and update requests which ties (links) various partitioned database and files within file system. RefID is a 128-bit UUID generated for every packet, represented as Hex format [20] [21]. Encrypted enrolment packets are stored as files with UUID as the file name. This ensures that administrators who have access to file system for backup and other activities do not have any knowledge from the file name and is completely anonymized and since it is PKI encrypted, has no access to file content either.

Archive module uses XFS file system to store these encrypted files in directory structure based on the UUID as the hash bucket. First two levels of the directories are using the first 4 hex characters and hence the directories are uniformly filled. All mount points are 2 TB LUNs coming off SAN boxes which can hold roughly 500,000 to 600,000 files considering each file is about 3 to 5MB in size. At any instant of time, many 2TB LUNs can be "active" where files are archived in round-robin fashion and when these LUNs get full, application moves onto set of active ones. Full LUNs are then made read-only. Archival batch program keeps track of UUID to LUN mapping through an RDBMS table. Every rack server running archive module roughly holds about 128 such LUNs mounted through fibre channel. Currently Aadhaar application uses about 15 such archive servers to store all such files. These archive files are also replicated to similar setup across the data centre and also backed up in each data centre, making totally 4 copies within the system.

Application access to archival system is via an HTTP server (Tomcat) running on all archive nodes. These "archive readers" serve specific files given the UUID and if not found, it generates a specific event and provide a specific error code and application can retry after configured number hours. This specific event can be listened by automated provisioning system and on-demand provide access to mount point.

Following architecture diagram depicts the archival system as described above.

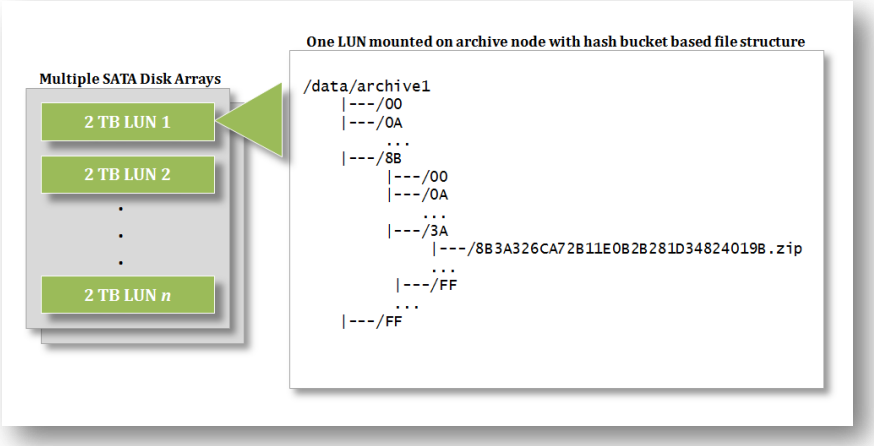


Figure 12: Enrolment Packet Archival Architecture

4.2.6 Print & Partner Integration

As 1+ million enrolments are processed every day, print letter job is also expected to scale up fast. With various language wise printing capabilities, it is essential that a highly configurable, multi-threaded, parallelizable job is created to handle the volume.

Print job scans the SEDA enrolment records database to identify records that are ready for letter printing. Then it invokes the Common Search API (CSAPI) internally to retrieve resident data from Aadhaar (UID) Master to generate XML files of suitable format. XML data contains resident demographic data, photograph and a 2D bar-code containing a subset of the demographics details. This batch job uses the common batch framework of the underlying technology platform. This batch depends upon the data available for print and is triggered based on the CRON timing configuration available in application properties file.

Once, the print letter xml is generated then, the letter status type of the enrolment record will be changed to 'Letter Generated' and entry will be inserted into tracking table for postal tracking.

All UIDAI interfaces with external partners including print/logistics partners are digitally signed to provide non-repudiation and encrypted using partner public key. Partners must decrypt the file using their private key. Digital signature of UIDAI can be verified using the public key of UIDAI to ensure integrity of data.

4.2.6.1 E-Aadhaar Service

E-Aadhaar service, exposed via resident portal, is built as an online web application backed with authentication and data retrieval APIs within the CIDR. It is deployed as a web application with HTML/JavaScript user interface. E-Aadhaar application in turn invokes two key internal APIs – one for OTP authentication and verification and second for demographic data retrieval. Both are internal APIs secured behind the firewall which abstracts OTP service and resident data retrieval service.

Although the service is accessible to public, it is secured through mandatory 28 digit EID (up to seconds field of timestamp), postal pincode of enrolment, full name as in enrolment, and a mobile number where OTP is sent before e-Aadhaar can be accessed. E-Aadhaar audits every request including IP address, mobile number where OTP is sent, and watches metrics such as number of times a particular EID was requested, number of requests originated from same IP, number of requests for which same mobile number was used for OTP delivery, etc. to track any abnormal patterns.

4.2.7 Aadhaar Update Services

UIDAI allows data update (demographics and biometrics updates) to Aadhaar holders post initial enrolment based on its update policy [22]. Update services are integral part of enrolment server.

For permanent physical update centre operations, UIDAI provides update software, integrated to enrolment client, which is an offline Java application. Registrars using enrolment agencies use this software quite like enrolment client. All security features, process monitoring features, data management features, and sync features are also available within update client.

UIDAI also offers Self Service Update Portal (SSUP) for demographic data updates. This module (SSUP) is a web application hosted within CIDR and exposed via Internet to residents. Update portal ONLY allows residents with registered mobile number to access the system using One-Time-Pin (OTP) authentication. In either of these cases, an encrypted, digitally signed update packet is created which, in turn, is picked up by SEDA flow for update/correction processing.

From an architecture perspective, update flow is very similar to enrolment based on various business rules of update [22] laid out by UIDAI except for some fundamental differences:

1. Update request is always authenticated to ensure only the resident (or guardian in case of children) can update his/her record. This is to ensure subsequent Aadhaar lifecycle updates are carefully protected from any invalid or unauthenticated updates happening to resident record.
2. Only exception to above rule is in the case of biometric update (first time) once the child with Aadhaar number turns 5 years. In that case, a full biometric de-duplication is done and in the case that Aadhaar (since the child before 5 years old was provided Aadhaar number on the basis of demographics alone) is found duplicate, that number is cancelled (through a separate cancellation flow).
3. Since Aadhaar number (or EID in the case of correction) is always available as part of update request, no biometric de-duplication is done. Instead, only biometric/OTP authentication is performed.

The update packet processing follows the SEDA architecture of enrolment server. Most of the processing flow is same as that of a new enrolment processing. There are additional validations performed for update/correction packets as compared to new enrolment processing.

- In case of Demographic update/correction, the packets are sent for resident's biometric verification (authentication) instead of de-duplication.
- In case of Biometric update the packets are sent to ABIS middleware for inserting the changed biometric to ABIS solution databases after biometric verification (authentication) of the resident.
- In case of a biometric update for an infant at the age of 5, new biometrics are inserted to biometrics servers and a full biometric de-duplication is performed with the biometrics servers to eliminate any duplicates.

All other validations (structural validations, metadata validations, rules validations, etc), exception workflows, and resubmissions (in the case of long failures or holds which are time bound) are handled quite like normal enrolment.

4.2.8 Information Privacy & Security

Application security for the above architecture cuts across all places where an un-trusted source or destination is used. The encrypted enrolment data file is uploaded in the DMZ to ensure against Trojans or malwares.

The enrolment/update data packets are encrypted by the client using public key cryptography with each data record having an HMAC which can identify any integrity violation of the data. Master keys are stored and managed within HSM (Hardware Security Module) appliance. It must be noted that the enrolment packet is constructed in memory and encrypted prior to writing of the file. All of the data, including biometrics and demographic data, is never stored in unencrypted form. The packet is encrypted with a randomly generated AES-256 symmetric session key and the key itself is encrypted with a 2048 bit public key, selected from a bank of UIDAI public keys.

Following should be noted:

- Every enrolment station, registrar, enrolment agency, operator, and supervisor are registered and authenticated.
- Every packet is biometrically signed by operator (and supervisor in various cases) and contains complete process data including station id, timestamp, location (pin code, GPS). This allows strong validations and traceability at packet level.
- Every enrolment data packet is "always" stored in PKI encrypted, tamper-evident files and are never decrypted or modified during transit.
- Enrolment data is "never" decrypted until it is reached within UIDAI's data centre's secure production zone.

Usage of strong 2048-bit PKI encryption technologies ensures that no agencies or persons can access, modify, or misuse the resident data during field enrolment or in transit to the UIDAI data centres.

In addition to enrolment packet, resident data in Aadhaar master database and BI data store is protected through various security measures. These include:

- **Encryption** – used to ensure data is encrypted in database and not available to administrators and other users in plain text format.
- **Anti-Tampering** – used to ensure data is only altered by authorized applications and NOT via command line SQL scripts.
- **Data Partitioning** – data is partitioned vertically (some attributes about resident is in one database while others in separate ones) and across multiple databases with a random alias (UUID based RefID) being the only link to ensure there is no central database table where all resident data is available.
- **Anonymization** – hashing techniques are used for anonymizing data in BI/Reporting data store, still having the ability to match and do analytics.

Other than application techniques as described above to protect resident data, UIDAI has implemented data centre best practices and technologies such as firewalls, IPS systems, zoning and access control, centralized security policy management, audits, 24x7 monitoring through Security Operations Centre (SoC), and strong security procedures used to ensure CIDR is protected.

4.2.9 Data Model and Technology Stack

Enrolment module uses various data stores in a loosely coupled fashion for managing entire enrolment and lifecycle processes. There are 4 core data stores that provide single source of truth and these core data stores are wrapped with APIs that provide access from other modules. These are:

1. **Active Enrolment Master** – This is where core data about resident and some key process metadata (operator id, registrar/EA codes, timestamp, station id, etc.) related to active enrolments and update request updates are stored. This is typically about 30-40 million. EID and RefID are the unique keys.
2. **Aadhaar (UID) Master** – Once Aadhaar number is allocated to the record or if updates are done, this data store is populated with core resident data including photograph, and key data related to that particular enrolment/update request. This data model also includes update history with timestamp attached to the main UID master table. Because this is permanent and need to handle 2-4 billion in eventual state, data is sharded 100-way using the first 2 digit of Aadhaar number as the shard key. To handle photo blob storage, within each UID master shard, another 16 way shard is used using 3rd digit ensuring no table is growing beyond manageable size and is still accessible by the Aadhaar number without alternate index lookup.
3. **Reject Master** – This database stores all rejected enrolments. Data model is identical to UID Master except the "Aadhaar Number" column. It is not necessary to shard this since number of rejects are not large. In addition, access to this is not required to be super fast or not that frequent. Since this is identical to UID Master, photos are sharded 16-way within this database.
4. **Packet Archive** – This is the permanent archive of "all" enrolment/update packets ever received in the system. This is primarily a large pool of file system storage with random access indexes maintained within RDBMS.

Following diagram depicts enrolment data model:

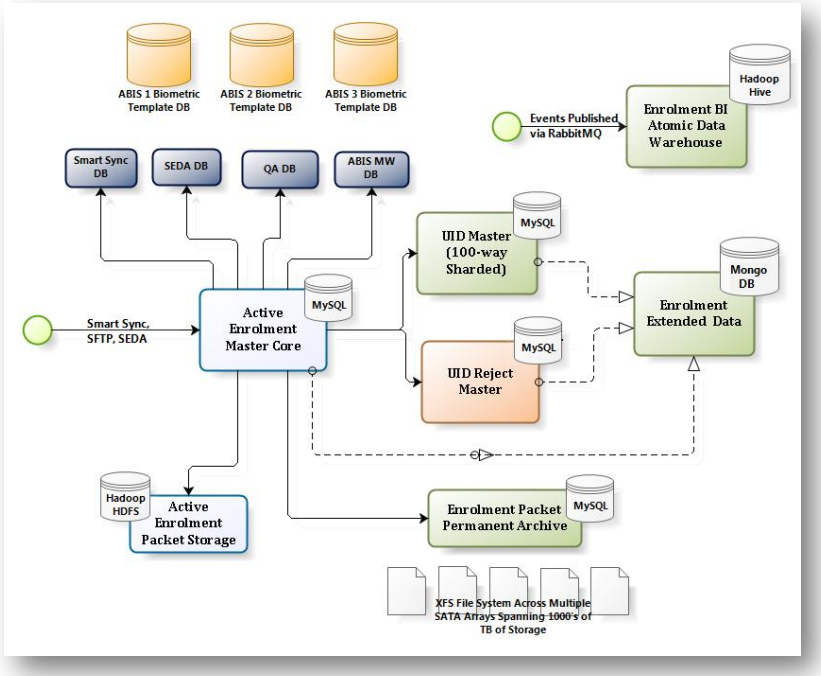


Figure 13: Enrolment Data Model

Other modules such as upload, ABIS middleware, Data Quality, Manual Adjudication, etc use an extended database of their own with link to the above core data via either EID or RefID. They can also be linked via Aadhaar number itself in the case of components that provide features at Aadhaar level. These extended databases are typically component level audits that can be archived away after a period. Analytics (BI) module is used for long term analytics/reporting where all events are stored within Hadoop Hive Atomic Data Warehouse (ADW) for long period of time. Various Hive and map-reduce jobs are run on that atomic data warehouse to derive aggregate metrics.

APIs wrap the core data models and provide uniform way to access this data via EID/RefID/UID irrespective of the state they are in (being processed, Aadhaar allocated, or rejected). Status tracking API, Common Search API, Advanced Search API are these core APIs. Other services such as e-KYC API, e-Aadhaar service, etc all built as wrappers on top of these core internal data access APIs.

Below diagram depicts technology stack used within enrolment server module at a high level. In addition to those depicted above, there are several open source libraries used throughout the system. Following table lists the technology stack of Aadhaar enrolment server.

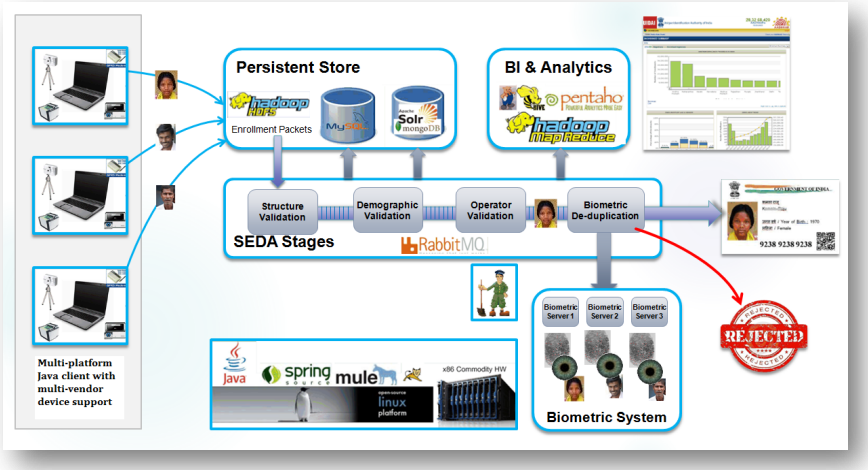


Figure 14: Technology Stack of Enrolment Server

Architecture element	Technology mapping
Operating System	Any Enterprise Linux (RHEL is used as of now)
Application language	Java 1.6 or above
Messaging Platform – Publish/Subscribe, Queues	RabbitMQ – provides high throughput messaging, distributed deployment across data centres and high availability, reliability options like message persistence and transaction support

Application Container	Spring Framework – provides lightweight container for Java objects, Security via Acegi, AOP, Remoting, and implementation of Dependency Injection for better maintainability
Enterprise Service Bus	Mule ESB – Integrates well and runs inside Spring, Implements the SEDA model suitable for event driven applications like UID enrolment server
Data store mapping layer	Hibernate plus custom wrapper layers for handling multiple types of data sources, sharding, etc
RDBMS Persistence – storing relational data	MySQL – SQL compliant and replaceable with equivalent open source or commercial alternative
Distributed File System persistence	Hadoop Distributed File System (HDFS) – supports deployment on Linux and on commodity grade disks. Cloudera and MapR distributions are used.
Large storage	Large SAN storage (multiple storage boxes from vendors like EMC) using commodity SATA disks, archive data done within XFS file system.
Batch Processing – execution of scheduled and repeating jobs	Spring Batch – elaborate framework managing jobs, steps within jobs and tasks within. Provides support for file system read-write jobs
Application monitoring	In-memory metrics collectors implemented in Java
Enrolment extended data stores	MongoDB is used to store extended enrolment data that are unstructured and document oriented. HBase is used to store biometric template extracts from the enrolment packets. The templates are used by SEDA stages, when required and also replicated by the Authentication servers.
Various web app UIs (Portal, NoC, etc)	Liferay enterprise portal HTML, Javascript, CSS, Apache Tomcat calling Spring services
BI, Analytics, Reporting	Apache Hive (Hadoop family) for atomic warehouse and various metric computations, Pentaho and MySQL for derived metric storage, reporting, etc
Encryption/Decryption, PKI key storage,	HSM (network appliance)
Transliteration and Indian language data matching	Cognirel's and CDAC Transliteration and Matching Library complying to Aadhaar Transliteration API

4.3 Authentication Module

Authentication module handles online resident authentication from various authentication user agencies. In addition to the main authentication service which offers multi-factor demographic/biometric authentication in an end to end secure

fashion, this module also contains related services such as best finger detection, one-time-pin request, etc. Unlike enrolment, authentication is an online service and is mainly concerned about response time per transaction.

As described in chapter 2, Aadhaar authentication supports demographic, biometric, and OTP (and combinations of these) based authentication as an online service. It is deployed as a secure HTTP(S) service accessible only through secure private network from authorized agencies. Biometric sub-system of authentication uses multi-modal biometric SDKs from multiple agencies, each tuned and cal liberated for best match at specified thresholds.

Authentication server implements logic as per Aadhaar Authentication API [13] and various server side parameters such as biometric match thresholds, SMS templates, which SDK to use for particular modality, etc. are configurable as per UIDAI policy. Documents related to Authentication Operating Model [23], Authentication Framework [24], Authentication Security [25], etc are available on UIDAI website.

As per UIDAI authentication security policy [25], Aadhaar authentication and related online services (Best Finger Detection and OTP Request) are deployed as an HTTPS service and are only accessible over a private network (MPLS or leased lines). Authentication user agencies (AUAs) use network services of Authentication Service Agencies (ASAs) to access UIDAI's production setup. Authentication is deployed across both data centres in active-active fashion.

Following diagram depicts the network architecture of authentication and related online services.

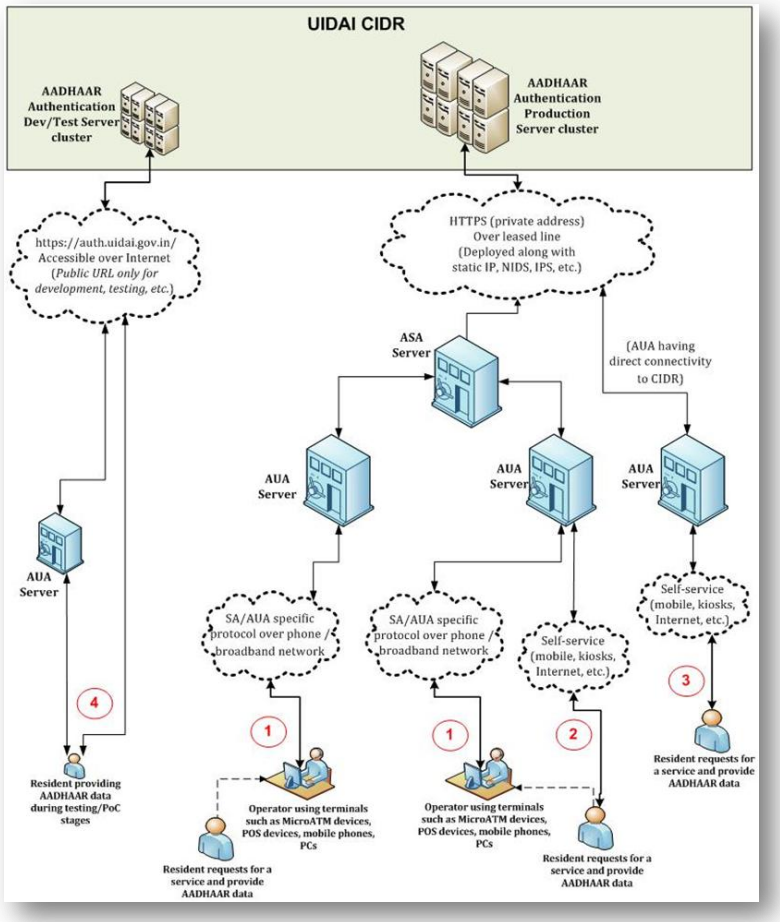


Figure 15: Authentication Network Architecture

4.3.1 Authentication API

Aadhaar Authentication API [13] is designed in such a way that applications that use the API can be implemented in any programming language, can run on any operating system, can work on any device form factor, can work over any network, and can be integrated easily into any application. Such flexibility is critical given the fact that Aadhaar identity authentication may be used in all sorts of applications running on handheld devices to larger systems over low bandwidth mobile networks to high bandwidth broadband networks.

There are two main parts to the API input data:

1. **PID Block** – This is the core data structure of the input containing resident's identity attributes that are matched with the data on the server. PID block may contain demographic data, biometric data, and/or OTP. Formats, validation rules, etc are defined in detail in API specification. PID block is encrypted at source and is never decrypted until it reached UIDAI authentication server. PID supports data in XML [26] or ProtoBuf [27] formats. PID block contains all sensitive data and is encrypted and to stop tampering (post encryption) encrypted HMAC value is also sent to UIDAI servers. PID block itself is versioned, independent of API version, to support backward compatibility as future versions evolve. PID block is formed on the capture device in memory before transmitting to AUA server.
2. **Authentication XML** – Once PID block is formed, it is subsequently wrapped in API envelop. This container XML has necessary meta information such as API version, AUA code, API usage license key, device data, and most importantly the Aadhaar number of the resident. Authentication XML is typically formed on the AUA server using the PID block that came from front-end device. It is then digitally signed and sent via ASA network to UIDAI server.

API supports exact and partial matching strategies for demographic data, structured and unstructured address matching, single or multiple biometric records matching including multi-modality (fingerprint and iris), and matching of

OTP providing multi-factor capabilities. API also supports strong security features and size optimization features (ProtoBuf [27] and Synchronized Session Keys).

4.3.2 Biometric Authentication

UIDAI offers both fingerprint and iris (one or multiple) authentication schemes. As per UIDAI's ecosystem strategy, authentication module also allows certified device vendors to supply fingerprint/iris capture devices to AUAs for their use on the field to conduct Aadhaar authentication. Already there are several vendors providing single finger, multi finger, single iris, and dual iris capture devices complying and certified as per UIDAI specification. API allows any of these certified devices to be used. UIDAI has published several papers with specific details on the devices and overall use of biometric in authentication scheme [28] [29] [30].

4.3.2.1 Aadhaar Biometric SDK API

UIDAI created a standardized API specification for biometric SDKs [31] in Java and asked the all the Biometric Service Providers providing software to UIDAI to implement this standard API. These SDKs, in turn, provide modality specific matching and other capabilities that can be embedded within various applications including authentication server. Following are the key reasons to take such an approach:

- **Vendor neutrality** – Aadhaar system is implemented using open standards and standard APIs to ensure that all components across the system are neutral to proprietary and vendor specific features.
- **Interoperability** – To allow various systems to interoperate in a seamless fashion it is critical that standard interfaces are used. This allows common data format definitions, protocols across the components that expose similar functionality.
- **Use of best-of-breed algorithms** – An open API allows best of breed algorithms to be used for special purposes. For example, if one fingerprint algorithm works well for old age people, and another one for younger people, a common API is required to dynamically choose and use one algorithm based on the input.

- **Plug-n-play capability** – When multiple modalities and algorithms are used, for true plug-n-play capability, common API and discovery mechanism is required.

Using this API, SDK developers may expose support for one or many modalities. For example, an SDK developer specializing in fingerprint algorithms may choose to implement only fingerprint modality support while some other SDK may provide support for fingerprint, face and iris modalities.

There are two major components that need to be exposed via this API from within the SDK:

- *Quality Check and Segmentation (QSS) Engine*: This interface is meant to expose quality check, segmentation, and sequencing functionality.
- *Extraction and Matching Engine*: This interface is meant to expose extraction and matching functionality.

All SDKs used within UIDAI complying with Aadhaar Biometric SDK API [31], following architectural and design aspects are taken care of to ensure scalability, interoperability, and manageability:

- **Thread Safe** – SDK implementation are thread safe to ensure multi-threaded applications can embed the SDK without functional or technical issues and should continue to run correctly and reliably on large scale. Aadhaar application modules are built in a multi-threaded fashion for handling scalability on multi-core machines.
- **Statelessness** – All SDK functionality (except for insert/identify operation) within interface are stateless in the sense that none of those method calls should result in any state being maintained within the SDK. This is critical to ensure that when insert/identity operations are not used, a single instance (singleton) of the engine can be used across threads to handle large scale.
- **Multi-platform Support** – Aadhaar system is built on Java and supports multiple platforms such as Linux and Windows. Currently, SDK supports Linux 32-bit and 64-bit, Windows 32-bit and 64-bit on x86 architecture.
- **No Data store** – SDK should not mandate any persistent data store for storing data. It is expected that data is stored and managed externally by

the application using the SDK. For SDK configuration, it may use an embedded data store or configuration files.

- **No External Dependencies** – Since Aadhaar applications run on a production network isolated from Internet and is secure, it is essential that SDK does not have any dependency on any external resources outside the machine or network in which it is running.

4.3.2.2 Single/Multi biometrics and Labeled/Unlabeled Matching

Unlabeled matching: The frontend authentication device does not provide the position/label of the finger presented (e.g. right thumb) to the authentication server backend, hence the backend matches it with all the 10 fingers of the given resident. This method is referred to as unlabeled matching or 1:10 matching.

Labeled matching: The frontend authentication device provides the position/label of the finger presented (e.g. right index) to the authentication server backend, hence the backend matches it with exactly that finger of the given resident. This method is referred to as labelled matching or 1:1 matching.

Since unlabeled matching involves multiple matching operations, it results in more false matches at the same threshold. Hence, such a system must be operated at a higher threshold to maintain the target FAR, which results in a higher FRR as compared to a system based on labelled matching. However, a labelled matching system depends on correct labelling of the finger by an operator, which may introduce human error, and increase the rejection rate as well as require additional training.

During the BFD process the best finger of the resident is determined. Since the best finger is determined at the backend it is stored on the server side against the resident record. Knowledge of the best finger in the backend can be used to implement an adaptive threshold scheme in the following method.

- Resident presents a finger for authentication with providing a label (once the resident has gone through BFD process the resident is likely to present the best finger, however the resident is not restricted to the best finger).
- At the backend, ten matches are performed to return up to 10 different scores (against his/her all fingerprints):

- A lower threshold is applied for matching the best finger. This enables minimizing FRR.
- A higher threshold is applied for matching other fingers. This enables the resident to match at a higher threshold even if they do not present best finger. Higher threshold allows system to prevent increase in FAR.

If resident uses best finger, then matching would be at threshold lower than "unlabelled" threshold else it would be at threshold slightly higher than "unlabelled" threshold. Assuming most residents would use their best finger as they get familiar with usage and hence Aadhaar authentication achieves a much lower overall FRR without compromising FAR.

4.3.2.3 Biometric Matching Server

At a high level, biometric matching servers which are internal services within overall authentication, are highly configurable and rules driven. These servers use specific SDKs in embedded form to do various biometric matching and related functions.

During data load and as part of everyday Aadhaar enrolment, both finger and iris templates are extracted for specified SDKs and stored in HBase which is the core resident data store for authentication and related services. During actual authentication, incoming templates are matched against the stored ones and based on the modality and threshold, a pass/fail response is generated. If there are multiple fingers or dual iris as part of input, a fusion algorithm is used to arrive at the eventual match score. Matching logic is as follows:

- Server looks up the configuration and selects appropriate SDKs (specially tuned SDKs for each of the modality can be used in a plug-n-play fashion).
- Then the match is made for all biometric input records and match score is obtained.
- Once all match scores are available, adaptive fusion scheme is used in determining the fusion (combined) score.

- Fusion score is compared against appropriate fusion pass/fail score for that SDK (note that pass scores that are calibrated for each SDK using millions of test authentications). While doing this, labelled (finger position known) and unlabeled (finger position unknown), with their different match thresholds are also taken into consideration.

SDK Server is implemented as a Java server with embedded SDK and any number of SDK servers can be added to cluster as the overall system scales. Authentication servers connect over the local network to SDK servers whenever required to do biometric matching. Each of the SDK servers itself is multi-threaded and stateless so that same pool of threads can be used for many matches.

4.3.3 One-Time-Pin (OTP) Authentication

OTP request can be initiated by the resident by calling IVR or sending SMS or the request can be initiated by the application on behalf of the resident using OTP Request API [15]. OTP is always delivered on the resident's mobile and/or email and application is expected to capture that during authentication so that OTP can also be validated along with authentication.

The application initiated OTP request should work according to following flow:

1. Application (an application on an assisted device or self-service kiosks, or applications on the Internet), wanting to use Aadhaar OTP as a factor within Aadhaar authentication, initiates the transaction flow.
2. Application captures Aadhaar number.
3. Application, through AUA server, invokes the OTP Request API by forming digitally signed API Input XML.
4. UIDAI server processes the input, validates it, generates OTP, and sends it to resident's registered mobile and/or email.
5. UIDAI server then responds to the OTP request API caller with an XML with success or indicating any error.
6. AUA application then requests resident to enter OTP that was received on his/her mobile and/or email so that application can package all that data and invoke Aadhaar authentication.

For each OTP request an audit trailed needs to be maintained at data centres. Audit Trail should be replicated from regional data centres to the central audit repository. These audit trails capture incoming data and responses in a digitally signed audit XML with additional data such as matching scores and internal configurations used for future references.

Aadhaar OTP Request service (or simply OTP service) is exposed as stateless service over HTTPS. These servers, quite like authentication servers, are load-balanced within the data centre and is available in active-active mode across data centres. OTP service uses Voldemort as the in-memory distributed cache. Every OTP is configured to expire (if not used) within a specified time interval (e.g. 20 minutes). OTP is delivered via internal outbound SMS API which uses a standard SMS API (internal HTTP service) to connect via SMS server to deliver SMS to resident. If email is registered by the resident, OTP is also sent via email. All these services are stateless, load-balanced services and can scale horizontally.

If resident needs to manually request OTP (instead of AUA application initiated via API), he/she can do so using one of following means:

1. Send an SMS with text "OTP <*Aadhaar-number*>" to UIDAI's OTP mobile number. Aadhaar OTP server ensures origin mobile is same as that Aadhaar holder's registered mobile before generating and sending an OTP.
2. Visit UIDAI Resident portal page, enter Aadhaar number along with other mandatory details.
3. Aadhaar mobile application which will also allow offline HOTP.

4.3.4 Authentication Server

Authentication server implements all the logic necessary to match and authenticate Aadhaar identity claims. It is built using a highly scalable architecture supporting distributed active-active deployments across geographically separated data centres. It is highly secure and configurable to meet the stringent requirements. The Authentication server primarily has 2 core sub-modules:

1. Resident data extraction into Authentication data store
2. Core Authentication service

4.3.4.1 Resident Data Extraction

Enrolment server processes enrolment packets and stores the Aadhaar's demographic data in UID Master which contains demographic and photo data. Aadhaar data that is needed for authentication has to be stored in database that offers faster read/writes and promises higher throughput. Since RDBMS and XFS based archival system cannot be used for reading resident data during authentication (100+ million reads a day) as they do not offer desired performance, it is imperative that data is extracted from enrolment data stores and stored in a high performance, distributed, read data store such as HBase.

In addition, the biometric authentication requires usage of biometric template gallery, which is biometric feature set extracted from raw biometric images. Hence, the processing of biometric images also has to be performed as part of preparing data for authentication purposes.

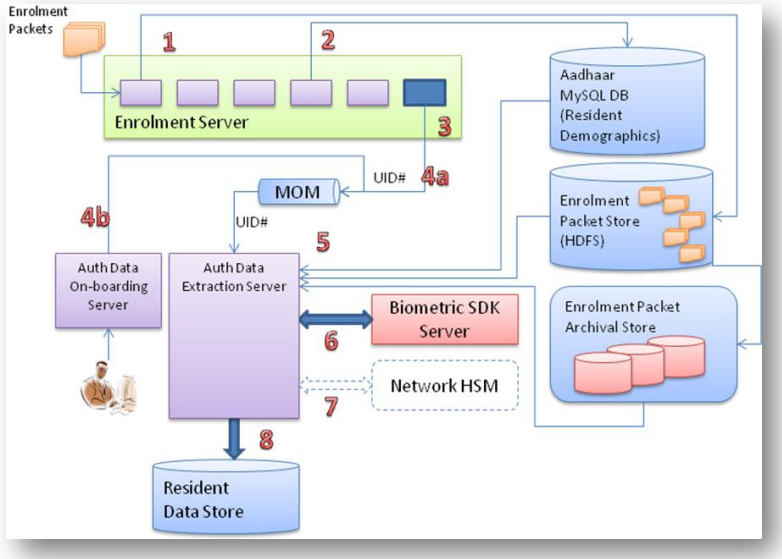


Figure 16: Authentication Resident Data Load

1. As enrolment packets are processed by Enrolment Server, they are copied to Enrolment Packet Store.
2. If enrolment packet passes all the checks defined by Enrolment server, an Aadhaar number is generated for that enrolment and is persisted in Aadhaar master database.
3. Enrolment server is built on SEDA architecture wherein the enrolment packets processed in various stages. A dedicated SEDA stage is added within enrolment/update flow, whose responsibility is to publish the Aadhaar number (UID) to a queue for further processing for authentication purposes.
4. "Authentication Data Extraction Server" is the component that is responsible for creating resident data for authentication purposes. This component listens on a queue for the Aadhaar numbers whose data has to be created/updated and stored in Authentication database. There are two components that can request for Authentication data extraction.
    a. Enrolment server: Every time a new Aadhaar number is generated or existing one is updated, enrolment server publishes the Aadhaar number to the queue which is consumed by data extraction server.
    b. Authentication data on-boarding server: This component is responsible for trigger Authentication data extraction for a list of Aadhaar numbers on-demand and is used for synching, on-demand extraction, and specific administrative tasks of on-boarding specific list for explicit sync.
5. For each of the Aadhaar number for which Authentication data extraction is requested, "Authentication Data Extraction Server" collates data from two sources: Demographics data from "UID Master" in RDBMS and raw biometrics data from the encrypted Enrolment packet (either in HDFS or archive XFS).
6. Since the biometrics images from the Enrolment packets are not directly usable for authentication purposes, the "Authentication Data Extraction Server" extracts the templates for these by using one or more Biometric SDKs.
7. Network hardware security module (HSM) is used by "Authentication Data Extraction Server" for encryption and HMAC computation of the demographics and biometrics data before storing it in the data store (HBase).
8. The demographics and biometrics data extracted in above steps are persisted in the "Central Resident Data Store" (HBase) as XML Documents.

4.3.4.2 Core Authentication

Core authentication setup consists of applications and infrastructure components needed for support OTP API and Authentication API.

Following steps are required during the processing of OTP requests:

1. One time pin (OTP) is one of the factors for authentication. An OTP request is initiated by one of the following:
    a. A resident can request the OTP by sending an SMS to UIDAI's short code. The SMS message will be converted into a HTTP GET call by the Telco's SMSC and the result of HTTP GET call is typically sent back to caller's mobile phone as SMS. Authentication setup hosts the "SMS/IVR Gateway" that handles the HTTP GET calls from SMSC. In this particular case, the web interface will be responsible for handling SMS messages requesting for new OTPs.
    b. Alternatively, residents can request for OTPs using resident portal or in future using Aadhaar mobile application.
    c. AUA application may invoke "Aadhaar OTP API" [15] which, in turn, triggers internal OTP service to send OTP to resident mobile/email.
2. In all the above cases, the OTP requests will be converted into a service call on the "OTP Server". "OTP Server" hosts a REST style web service that is available only within the UIDAI network and is capable of generating new OTPs and creating notification events so that OTP is delivered to resident via email and SMS. "OTP Server" generates a random 6-digit OTP, and stores it in "OTP Store", wherein the OTPs are indexed by Aadhaar number. Both OTP and Aadhaar number are SHA-1 hashed before getting stored in "OTP Store".
3. "OTP Server" generates an audit trail capturing request and response details. The audit trail is stored in "Audit Trail Store".
4. "OTP Server" generates a notification event that is passed on to "Authentication Event Sink" application through a queue. It also generates a BI event with the details about request and an ID for the generated OTP so that OTP generation request and corresponding Authentication request in that OTP was used can be correlated, if needed, for reporting purpose.
5. "Authentication Event Sink" converts the notification event into an Email message and into an SMS message (sent via SMS Gateway).

6. "Authentication Event Sink" converts the BI Event into a comma separated value string and persist it into Hadoop Hive based atomic data warehouse for all analytics and reporting needs.

Following steps are required during the processing of authentication requests:

1. Authentication API is routed by AUA hosted application servers through the ASA private secure network to UIDAI CIDR.
2. Additionally, Authentication API may also be invoked by internal wrapper applications (e-KYC, update service, mobile/IVR/portal applications, etc).
3. Requests received by both the above mediums are processed by the Authentication server cluster behind a hardware load balancer. This application is the entry point for processing of incoming XML requests. It interacts with the Network HSM (Hardware Security Module) for decryption of encrypted parts of the authentication requests.
4. Authentication server validates whether HMAC present in the Authentication request has already been used by checking whether it is present in HMAC store. If yes, then, the Authentication request can be considered to be replayed. Based on the configuration of Authentication server, such requests may be processed or rejected. If HMAC is not used earlier, Authentication server will record the usage of HMAC by storing its value in the "HMAC store".
5. Authentication server validates "AUA" and "ASA" codes, digital signature, and license keys. Then server reads the resident's data from the "Resident Data Store". If input contains demographic data, demographic authentication is performed first. The "Language SDK" is used to match the values if name or address is specified in Indian Language. Master data and recent resident data are cached in-memory to ensure that repeated reads of same data can be read faster instead of going to the database always.
6. If OTP is used in the authentication request, then, the OTP value for that Aadhaar number is read from "OTP Store" for matching.
7. If authentication request contains biometrics, then, biometric template verification using "Biometric SDK Server" is performed against stored biometric templates for that resident.
8. After a successful/failed authentication, Authentication server creates an audit record, and stores it in the "Audit Trail Store".

9. Authentication server generates two sets of events after the authentication. Both these events are published to a queue to be consumed and processed by "Authentication Event Sink" application.
    a. Authentication notification events – These events represent notification to resident about the result of authentication.
    b. BI Event – These events represent BI events that capture those details about the authentication which are needed for BI analytics.
10. Event Sink module processes the incoming "Notification Events" by transforming them into Email and/or SMS to residents. Event Sink module also processes the "BI Events" by converting it into a comma separated value string and by storing it in Hadoop Hive atomic data warehouse.

4.3.4.3 Scalability and High Availability

Scalability and availability are the key NFR cross-cutting concerns for all the components of this architecture. This section details how scalability and availability concerns can be addressed for each of the components within authentication.

Authentication, OTP and SMS/IVR Gateways

Authentication Architecture has following components that are web-based in nature and uses Apache Tomcat as runtime:

1. Authentication Server
2. OTP Server
3. SMS/IVR gateway

These applications are made highly available by deploying multiple instances of the application in a cluster and then load-balancing them using hardware load balancers. Entire deployment is repeated in geographically separate data centres.

All these applications are stateless in nature. None of these applications need session-like semantics, which means that a request can be processed by any node in the cluster across both data centres. As the load on the server increases, application can be scaled horizontally by adding application instances. If any node

goes down in the cluster, the other members of the cluster continue to provide services and thus ensuring service availability in case of node failures. If all the nodes in a cluster go down, Intra-DC load balancer routes the requests to other DC.

Biometric SDK Server

Biometric SDK Server is also a web application similar to Authentication and OTP servers. Hence, it can also be scaled either using hardware load balancer or a software load balancer such as Apache httpd server.

Authentication Data and Event Sync Servers

Authentication data extraction server can be horizontally scaled to extract data for more number of Aadhaar numbers by adding more instances. Currently this is scaled to handle 1.5 million a day to meet Aadhaar generation of enrolment module. Due to inherent nature of Rabbit MQ messaging wherein one message is processed by only one consumer, by adding more RabbitMQ consumer instances, extraction throughput can be increased as needed. Template extraction can also be done across data centres.

4.3.5 Information Privacy & Security

Aadhaar authentication uses open, standard based security mechanism to secure data and service and is designed to address transaction privacy.

API Data Security

Both Authentication and OTP API detail mechanisms to secure the data in transit by proposing usage of encryptions, HMAC, and digital signatures, and ensure that data cannot be stolen or modified in the transit, and its authenticity and origin can also be verified.

Data should be encrypted with a dynamic session key using AES-256 symmetric algorithm (AES/ECB/PKCS7Padding). Session key, in turn, is encrypted with 2048-bit UIDAI public key using asymmetric algorithm (RSA/ECB/PKCS1Padding). Session key must not be stored anywhere except in memory and should not be reused across transactions. Only re-use of session key that is allowed is its use as seed key when using synchronized session key scheme.

The encryption/decryption flow is as defined below:

1. Aadhaar number, demographic, and biometric details as required by the application are entered into the device along with other factors such as OTP if it is used. If OTP is used, the request for OTP is sent to Aadhaar server along with Aadhaar number (see "Aadhaar OTP Request API 1.5" specification). Aadhaar Authentication server sends the OTP back to the resident's registered mobile phone as an SMS and to the registered Email address.
2. AUA/Sub-AUA application generates a one-time session key.
3. The authentication "Data" XML block is encrypted using the one-time session key and then encoded (base 64).
4. The session key is then encrypted with the UIDAI public key.
5. AUA application on the device sends the encrypted block along with HMAC data to AUA server.
6. AUA server forms the final authentication XML input for API including license key, transaction reference ("txn" attribute), digitally signs it, and sends the data to Aadhaar authentication server through an ASA network.
7. Aadhaar authentication server decrypts session key with the UIDAI private key. The data block is then decrypted using the session key.
8. The resident's decrypted biometric, demographic information, and optional OTP is taken into account during match based on the input.
9. Aadhaar authentication server responds with a "yes/no" as part of the digitally signed response XML.

Aadhaar authentication records all the authentication requests and their responses for audit purposes. Notification inbox is maintained for each resident using which last *n* months (based on UIDAI's audit retention policy) of notifications can be viewed by resident.

All authentication responses are digitally signed by UIDAI which helps AUAs to maintain electronic audits. In addition, attributes "ts", "info" within the API

response can be used to verify if it the request was indeed for a particular Aadhaar number, if the request indeed had a biometric factor, when was the authentication done, etc. Such self verifiability of the authentication response allows 3rd party applications to trust and electronically verify the response quite similar to that of an offline trust establishment against a signed paper.

Input Tamper Protection

As per authentication API specification, it is essential that authentication client application sends an HMAC for the PID block so that server can verify the PID block's integrity. The value of this element has to be computed by authentication client as SHA-256 Hash of unencrypted PID block.

Authentication server performs the following processing on every request:

1. Decode and decrypt the PID XML from <Data> element.
2. Re-compute the SHA-256 Hash of PID XML.
3. Decode and decrypt the value of <HMAC> element.
4. Compare the re-computed SHA-256 hash with HMAC value received in authentication request.
    a. If both values match, then, integrity of authentication request is preserved and server will proceed with further processing of the request.
    b. If values do not match, reject the authentication request with error code representing "HMAC Validation failed".

CIDR Data Storage

Within the UIDAI backend systems, Aadhaar number is not stored in any of the authentication database. Instead, a SHA-1 hash of Aadhaar number is stored. In addition, record level encryption and tamper detection features ensure resident data within HBase is neither available to internal administrators nor it can be modified by unauthorized people or applications.

Audit trail stores the request and response XMLs along with unique response code as the audit key. When publishing events to BI system, RefID is published instead of Aadhaar number, and entire BI system is stripped off all PII data. Similarly, PINs

are never stored in plain text form. Their SHA-1 hash value is store to avoid anyone from being able to view the raw values and misusing them.

Network and Server Security

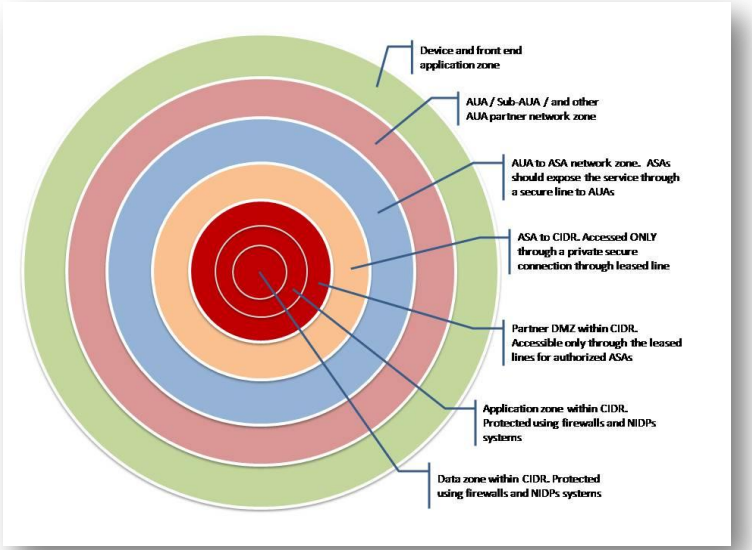


Figure 17: Authentication Network Security Layers

Securing network at multiple levels between the front end authentication points to CIDR is necessary to ensure protection against network attacks which result in "Denial of Service" (DoS). It is also important to ensure high availability and redundancy even if some parts of network are compromised or unavailable.

AUAs and their partners (sub-AUAs, application providers, etc.) are required to put appropriate network security in place to ensure their systems are protected from attack. It is hence mandated that standard network practices such as usage of encrypted channel, usage of digital certificates, IP filtering, authentication of

applications/users/ devices, network protection through firewalls and NIPS, auditing, etc. are put in place.

Within CIDR, UIDAI ensures multiple levels of network security through creation of DMZ, application zone, and data zones and protecting all the zones using multiple firewalls, network intrusion prevention systems, and strong access control and audit schemes.

Since many applications and services across the country will heavily depend on Aadhaar authentication, it is strategically important to not to expose Aadhaar authentication over Internet (or any public network) and not create "*single point*" of attack that can potentially affect many services. It is hence critical to expand the secure zone beyond CIDR and allow authentication service to be exposed through multiple network end points. Creation of ASA as a network service provider and exposing authentication service ONLY through secure private connections using leased lines is strategic to ensure multiple end points always exist to provide authentication service in a secure and always available fashion. As per UIDAI policy, authentication and other online services such as e-KYC are never exposed over Internet or any public network [25].

Replay Attack Protection

Request replay attack refers to a form of network attack in which an attacker can acquire an authentication request XML, and re-send it to Aadhaar Authentication server to perpetrate a fraud.

Each authentication request has a unique session key and HMAC. Aadhaar Authentication server keeps track of all the session keys and HMACs used for a period of *n* hours, after which an authentication request is considered expired.

On receiving an authentication request, server ensures that session key and HMAC contained in the request are not being re-used by checking it against the list of session keys that were seen by it. If they are reused, it would mean a possible replay, and such requests are rejected.

4.3.5.1 Registered Devices

Aadhaar authentication supports two kinds of sensor devices to be used to capture biometrics – Public Devices and Registered Devices.

Term "Registered Devices" refers to devices which are registered with Aadhaar system for encryption key management. Aadhaar authentication server can individually identify and validate these devices and manage encryption keys on each registered device. Term "Public Devices" refers to devices which are not registered with Aadhaar system and use their own encryption key generation scheme. Aadhaar authentication server does not individually identify public devices and uses an alternate encryption strategy for them.

In order to protect against reuse of stored biometric, several measures are already in place such as deploying signed applications, local encryption from firmware to host, operator/device authentication by AUA, usage of multi-factor authentication, resident SMS/Email alerts on every biometric authentication, host device ID/location for analytics and fraud management, and so on.

In the case of public devices, although above security measures are in place, since the encryption between capture device and host is not managed, there is still a technical possibility of having stored biometrics being used on a compromised device/application to conduct some transactions although it can be potential detected and subsequent usage stopped on the server using above security monitoring and protection scheme already in place.

Aadhaar Registered Devices Specification [32] pushes the security all the way to capture hardware to address the above need.

4.3.5.2 Resident Notifications

Resident is notified by SMS and/or Email about the outcome of authentication requests. Notification will not, by default, be sent for demographic Authentication unless resident has explicitly opted for it whereas mandatory notification is sent on biometric/OTP authentication. Notifications will also be available via portal/mobile applications.

4.3.6 Data Model and Technology Stack

Following diagram represents logical data model of authentication databases:

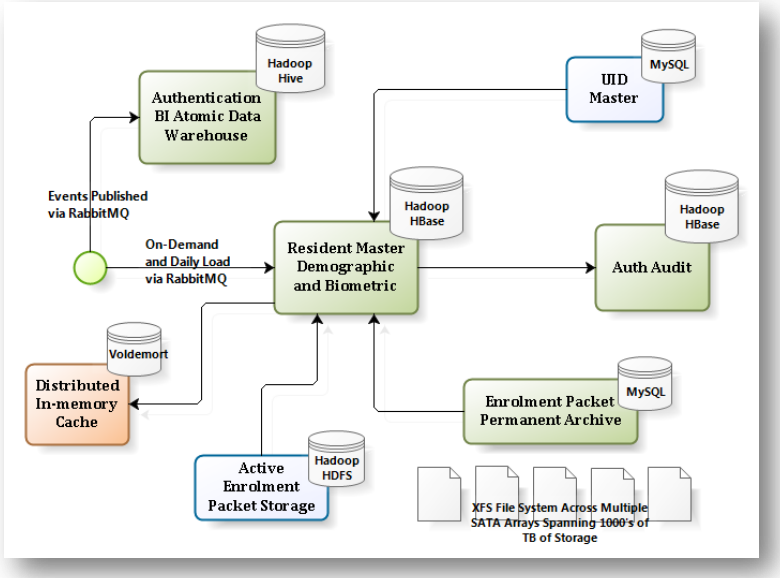


Figure 18: Authentication Data Model

Resident and Audit Data Store

Resident and audit trail data stores are based on Hadoop HBase which allows horizontal scaling of the underlying data nodes and high availability.

OTP and HMAC Store

OTP and HMAC stores both need following characteristics:

1. Fast read/write response times and high throughput
2. Storage of data for short period of time (max of 1 day) and expiry
3. Data need to be visible to all the systems and across DCs in near real time.

To meet these needs, Aadhaar authentication server used Project Voldemort, a distributed in-memory key-value store. This is also used for data caching of master data and recently used resident data.

Technology mappings to architecture elements	
Architecture element	**Technology mapping**
Operating System	Any Enterprise Linux (RHEL is used as of now)
Application language	Java 1.6 or above
Messaging Platform – Publish/Subscribe, Queues	RabbitMQ – provides high throughput messaging, distributed deployment across data centres and high availability, reliability options like message persistence and transaction support
Application Container	Spring Framework – provides lightweight container for Java objects, Security via Acegi, AOP, Remoting, and implementation of Dependency Injection for better maintainability
RDBMS Persistence – storing relational data	MySQL – As being used by already existing applications.
Large-scale random-access data store for long term storage.	Hadoop HBase – Cloudera and MapR distributions used in production.
Small-scale random-access distributed in-memory key-value store	Project Voldemort – is a distributed key value storage system. It is used for storing OTPs and Skeys in addition to master data and recently used resident data.
Batch Processing – execution of scheduled and repeating jobs.	Spring Batch – elaborate framework managing jobs, steps within jobs and tasks within. Provides support for file system read-write jobs
Web Application Container for API services	Tomcat
Various UIs (portal, NoC, etc)	Liferay, HTML, Javascript, CSS, Apache Tomcat
BI, Analytics, Reporting	Apache Hive (Hadoop family) for atomic warehouse and various metric computations, Pentaho and MySQL for derived metric storage, reporting, etc
Encryption/Decryption, PKI key storage,	HSM (network appliance)
Transliteration and Indian language data matching	Cognirel's and CDAC Transliteration and Matching Library complying to Aadhaar Transliteration API

4.4 E-KYC Module

The Aadhaar e-KYC API can be used by an approved agency (KUA) to provide paperless KYC experience for Aadhaar holders. Whenever availing a service, agency that provides the service may request resident to either share a paper copy of the Aadhaar letter or if the agency is an e-KYC user agency, request resident to authorize via biometric/OTP authentication, to retrieve demographic and photo information in digitally signed, encrypted, XML format.

Aadhaar e-KYC service provides a convenient mechanism for agencies to offer an electronic, paper-less KYC experience to Aadhaar holders. The e-KYC service provides simplicity to the resident, while providing cost-savings from processing paper documents and eliminating the risk of forged documents to the service agencies.

4.4.1 E-KYC API

Aadhaar e-KYC API [16] allows agencies to build paperless, electronic PoI and PoA verification on a computer or on a tablet or on mobile/handheld making it easy to provide services such as bank account opening, insurance issuance, etc at the customer. API is designed to support applications to be built using any programming language, to run on any OS, and access via any network. It is built as a wrapper to core authentication API ensuring resident is integral part and is explicitly authorizing every e-KYC transaction.

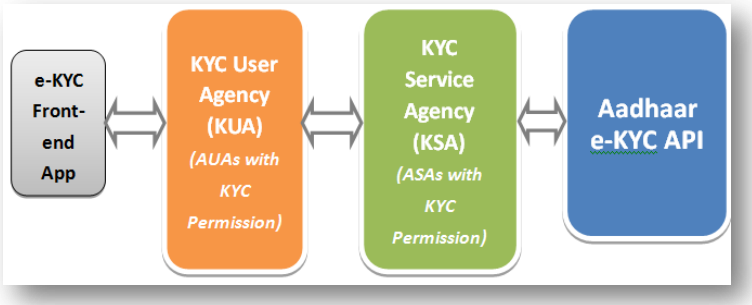


Figure 19: Aadhaar e-KYC Flow

Following the data flow of a typical KYC API call from left to right and back.

1. KYC front-end application captures Aadhaar number + biometric/OTP of resident and forms the encrypted PID block.
2. KUA forms the input XML, digitally signs it, and sends to KSA.
3. KSA forwards the KYC XML to Aadhaar KYC API.
4. After successful resident authentication, responds with digitally signed and encrypted XML containing demographic and photograph of the resident.
5. KSA sends the response back to KUA enabling paper-less electronic KYC.

4.4.2 Information Privacy & Security

The Aadhaar e-KYC service does not compromise security for convenience, instead offers a convenient solution that is very secure, resident authorized and protecting data privacy by eliminating paper trail on the field.

For details, refer to "UIDAI e-KYC Policy" [17] *and "Aadhaar e-KYC API"* [16] *documents.*

4.4.3 E-KYC Server

Aadhaar e-KYC server module is built as a simple wrapper to internal core APIs – Authentication API and Common Search API. It is developed as a web application deployed within a web container behind a load balancer. E-KYC service is stateless service and is load balanced across multiple machines within a data centre and clusters across data centres.

This web application primarily does the following 4 key steps:

1. Validate input request
2. Authenticate resident using API call to authentication cluster via HTTPS
3. Obtain resident demographics and photo using API call to Common Search API cluster via HTTPS
4. Audit the transaction before returning the digitally signed and encrypted XML back to KUA.

4.5 Platform & Common Modules

Aadhaar application is built on common technology platform based on Java and open source technologies. In addition, common modules such as portals, business intelligence & reporting, fraud detection, etc are shared across enrolment and authentication. Following sections describe architecture details of these frameworks and modules.

4.5.1 Technology Platform

Application modules are built on common technology platform that contains frameworks for persistence, security, messaging, etc. The Platform standardizes on a technology stack based on open standards and using open source where prudent. A list of extensively used open source technology stacks is given below:

- Spring Framework – application container for all components and runtime
- Spring Batch – runtime for various batch jobs
- Spring security – for all application security needs
- Mule ESB – runtime for loosely coupled SEDA stages of enrolment flow
- RabbitMQ – messaging middleware
- Hadoop stack – HDFS, Hive, HBase, Pig and Zookeeper
- Quartz – scheduling of batch jobs
- MySQL – RDBMS for storing relational data
- Apache Solr – Index for full text search
- Apache Tomcat – Web container
- Liferay – portal framework
- Several other open source libraries for random number generation, hashing, advanced data structures, HTML UI components, etc

The diagram below depicts the e-Governance platform as a set of layered technology building blocks that are used to build applications.

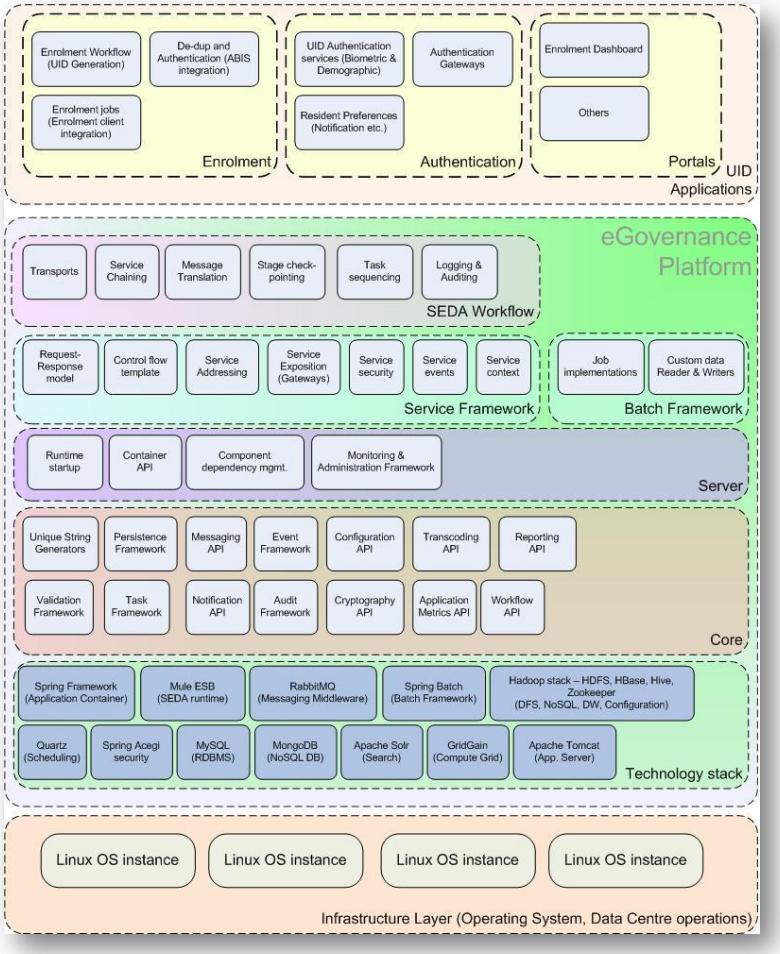


Figure 20: Application Stack & Frameworks

I

to common technical needs. The frameworks permit high degrees of reuse on implemented behaviour and are often extensible. Following sections describe these briefly.

4.5.1.1 Application Bootstrap Framework

The technology platform runtime follows a pre-defined bootstrapping sequence. This ensures order in loading of application containers and its components. The bootstrap framework is JVM agnostic and has the following capabilities:

1. Provide ability to specify configuration locations from which the application is initialized.
2. Configure application wide logging preferences.
3. Specify the type of application container to be loaded i.e. Generic, Service, Batch or SEDA container.
4. Provide hooks into the bootstrapping sequence. Applications may extend the thread safe bootstrapping sequence with bootstrap extensions. Each bootstrap extension may also influence the bootstrap outcome i.e. allow it to proceed or veto the entire bootstrapping sequence thereby ensuring that the application startup is complete where all application dependencies are initialized.
5. Provide JMX based administration interfaces to manage life-cycle of the bootstrapped application instance

4.5.1.2 Persistence Framework

Persistence framework within the technology platform provides a standardized mechanism to persist data to respective data stores. The Persistence Manager interface is used to persist data and has methods similar to the JPA framework.

The persistence framework is designed using the following entities:

- Persistence manager – Primary interface to the persistence framework
- Persistence provider – Registered with the Persistence Manager and normally maps to one data store type – RDBMS, DFS etc.
- Persistence handler – Registered with the Persistence Provider and implements the persistence calls.
- Persistent entity – Any business entity that requires persistence
- Criteria – Metadata container for the persistence call

Persistence framework provides the following value added features:

- Declarative mapping between persistent entities and their persistence stores. Ease of switching persistence strategies.
- Transaction support in persistence calls to specific data stores. Currently the framework supports single schema transaction support for RDBMS persistence provider.
- Sharding support for distributing data across data store instances. Shard hints identify target data store instances and may be specified at different levels of granularity viz. at the persistent handler level, for individual persistent entity instances and in search criteria. The framework also supports data retrieval from multiple shard sources.
- Persistence provider implementations for MySQL, HDFS, HBase, XFS, etc.
- The framework has connection pooling support for many of these data stores and also implements connection retry and recovery mechanisms.

4.5.1.3 Event Framework

Technology platform provides an Event framework for use by various platform components and applications to perform the following:

- Publish self-contained technical or business information that may be categorized as events in the system. Events are relatively small data elements that communicate occurrence of an action at a point in time, execution of a business logic or rule, indicate a notification or signal an alert of some kind.
- Enable point to point and multi-cast patterns for event publishing.
- Enable event consumption by Event type or subscribe to events from logical (URI based) or physical endpoints (message queues).
- Enable bulk or individual events publishing and consumption
- Provide a mechanism for integration between UID applications and sub-systems, enabling independent deployment and scaling of event producers and consumers.

The event framework is designed around the following concepts:

- Event publisher – provides behaviour for publishing events to application contexts, URIs or queues
- Event consumers – provides behaviour for event consumption from URIs or queues and perform forwarding
- Event types – Framework or application module specific event data containers
- Event multi-casting- Act as a mediator for mapping event publishers and consumers instead of hardwiring the two. Also enables one-to-many messaging pattern.

Event publishing and consumption in the Event framework is either synchronous or asynchronous, depending on specific producer and consumer implementations. The framework interfaces therefore define method signatures devoid of any processing details.

4.5.1.4 Task Framework

Task Framework in the platform enables work to be dispatched and executed outside an application's runtime container. Tasks may be executed within the same JVM or in a distributed manner on a compute grid. The application may decide on the execution mode after considering trade-offs between moving compute across machines and moving data across machines.

The task framework is designed using the following:

- A Task that encapsulates behaviour and data required for its execution. The task can travel across machine boundaries in its serialized form.
- A TaskContext that may be used pass additional information such as previous tasks' results and a mechanism to return results of its own
- Data containers for capturing task execution results
- Task Manager interface, and implementations, to execute tasks

The simplest implementation executes the tasks in the same JVM as the application's runtime and also within the same execution context. Task distribution across multiple processes or server instances has its overheads in data transfer and runtime management.

4.5.1.5 Validation Framework

Validation Framework in the platform provides a set of API for performing field level validations and also grouping multiple of these into business entity specific validations. The validation framework implements the "Strategy" design pattern.

The Validation Framework is designed using the following:

- Validator interface and its implementations for field-level validations. Generic expression based validations are supported.
- Business entity specific validation strategies that may be reused in application services and components where identical set of validations are required on the business entity type.
- Container for collecting and returning validation outcomes from execution of validation strategies.
- Ability for validators to influence execution of subsequent validations in a strategy. A Validator implementation may veto or allow continuation of the validation sequence.

4.5.1.6 Service Framework

Service Framework is an important building block in the platform that is used by multiple Aadhaar application modules. All business functionalities such as enrolment stages and authentication services are built on this framework. It has the following features:

- Implement services as stateless POJO objects and declaratively enable their access from local and remote clients. This stateless nature of services aids horizontal scalability of business services and their distribution across a number of servers.

- Implement a service interface that is request-response based. The interface classes are compiled from XSD definitions. This ensures that interfaces are derived from well-defined and standards-based schemas thereby aiding integration.
- Locate and load service definitions from the deployed project and provide uniform means to invoke the deployed services via a single Service container instance deployed per JVM.
- Manage life cycle of services and integrate it with that of the Service container.
- Intercept service requests and permit implementations to be plugged-in that perform pre-processing steps like authorization checks and to measure service performance metrics.
- Provide a uniform mechanism for service addressing using a uniquely identifiable service key comprising of a service name and version.
- Provide means to transparently change the deployment of services from local to remote machine or set of machines.
- Ability to break up service implementation activities into Tasks that may then be executed within the same JVM or on a compute grid.

4.5.1.7 Batch Framework

Batch framework runs as a server instance and is built entirely on Spring Batch and integrates with the Quartz scheduling framework for scheduling needs in batch job execution.

The custom code implementation on top of what is available through the Spring Batch framework is quite limited in the platform. The design of the batch framework itself is therefore not elaborated in detail in this document.

This layered architecture has three major high level components: Application, Core, and Infrastructure. The application contains all batch jobs and custom code written by developers using Spring Batch. The Batch Core contains the core runtime classes necessary to launch and control a batch job. Both Application and

Core are built on top of a common infrastructure. This infrastructure contains common readers and writers, and services for retrying, etc.

The following custom features have been built on top of the Spring batch framework:

- Batch Component Container – This Container is a Component Container implementation that provides runtime management features for batch jobs within the context of Platform bootstrap. This component container locates and loads batch jobs as independent Spring application contexts. This separation of job contexts allows isolation of one job from another within same container and also provides distribution of jobs across different batch instances.
- JMX interface for Job Administration – The following information from the job registry and repository is exposed via JMX – "Invocation statistics" (provides individual job specific invocation statistics such as status of last execution, execution time and errors if any) and "Out of turn job execution" (provides ability to execute a job interactively. This is particularly useful during development and testing and in production scenarios when a certain job needs to be executed well before its cron trigger fires.)

4.5.1.8 Rule engine Framework

The Rule engine is a general purpose framework that enables applications to externalize business rules such that the application and the rules can evolve independently of one another. Rule authoring is typically done by business users while application modules are delivered by development teams. An effective rule engine framework provides an application the required flexibility to support feature enhancements and adding new features like business validations and checks.

The rule engine framework within Aadhaar application is built using the JBoss Drools platform. Rule Engine framework provides following enhanced functionality over the default Drools API.

- Defines an instance of a rule engine containing a set of "Knowledge Bases" (group of rules). "Process Flows" are defined to further classify rules within each Knowledge Base.
- The Rule Engine framework allows to cache the rules in rule engine and refresh from Drools Guvnor at given time interval for performance.
- On application startup the rule engine will be initialized and all the defined Knowledge Bases will be loaded to working memory of rule engine.
- On application destroy the rule engine is stopped by closing all open resources and disposing the Knowledge Agent associated with the Knowledge Base.
- "Drools Rule Executor" provides means to programmatically read the rule attributes.
- The input data and rule execution result structure is defined outside the framework, so that the framework users have the option to define data formats.
- The Drools Guvnor is completely configurable, so that rules can be added, removed and modified at any time and makes the changes available without restarting the rule engine.

4.5.1.9 Security Framework

The technology platform provides a comprehensive application security framework built over the Spring Security. Spring security is a powerful and mature application security framework for use by enterprise applications. Platform leverages the following features from this framework:

- Easy Configuration using Spring Dependency Injection – aligns well with the rest of the platform frameworks like Server, Service Framework, Batch Framework – all of which are built using Spring.
- Non-intrusive and Non-invasive – Keeps the application objects free of any security code.
- Pluggable architecture – Follows interface driven design and therefore allows replacement of critical aspects of the framework like Authentication and Authorization provider implementations.

- Support for comprehensive Authentication mechanisms:
  - LDAP based authentication of user principals
  - Single Sign-on support via integration with CAS
- Support for comprehensive Authorization services:
  - Role and ACL (Access Control List) based authorization definitions
  - Web-layer security using Http filters and integration with Spring MVC
  - Service layer security by allowing to secure method invocations on Spring managed POJO objects

4.5.1.10 Cryptography Components

Security Service Module provides a standardized security solution by implementing Java cryptography, public key infrastructure to encrypt/decrypt data, hashing techniques, tamper detection schemes using HMAC etc. The security component comprises a set of APIs, and implementations of commonly-used security algorithms based on Java Cryptography Architecture (JCA) and The Java Cryptography Extension (JCE) frameworks and used to implement all cryptography required operations.

The module provides an API that security and application developers use to implement security functions in Java applications. Within that context, the module also acts as an intermediary between security functions that are implemented in Java applications using the Java security service APIs and security providers configured into the Java Security Service Module.

Cryptography module uses the following cryptographic provider packages:

- An implementation of the RSA algorithm and PKCS11 HSM Provider.
- An implementation of the SHA-1 and SHA-256 message digest algorithms.
- A RSA key pair generator for generating a pair of public and private keys.
- A certificate path builder and validator for PKIX, as defined in the Internet X.509 Public Key Infrastructure Certificate and CRL Profile.

- A certificate factory for X.509 certificates and Certificate Revocation Lists.
- A keystore implementation.

4.5.1.11 SEDA Runtime

The Staged Event Driven Architecture (SEDA) runtime is used when an application has one of the following requirements:

- Need to perform parallel processing of business logic in a distributed manner across a number of machines. The processing pattern is asynchronous.
- Need to orchestrate a sequence of steps or stages where the output of one stage is passed on as input to the next, after modifications if any.
- Support easy scaling out of workloads across a set of machines and managed entirely through configuration.

The simplified view of SEDA design is shown below:

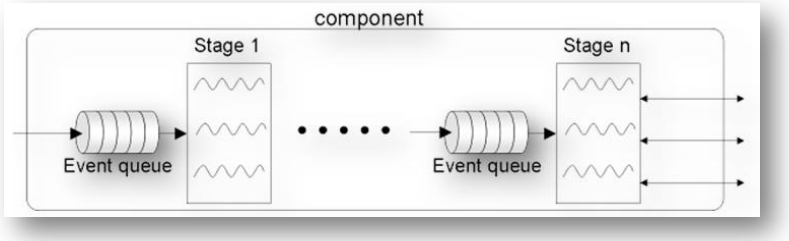


Figure 21: A Typical SEDA Flow

The Mule open source framework is used as is in the SEDA runtime. Mule is a light weight Java ESB (Enterprise Service Bus) and is highly scalable and configurable.

4.5.1.12 Build and Deployment packaging

All application modules within the technology platform follow a standardized build and deployment process which produces language runtime and operating system specific deployment bundles. The sub-sections below describe the process and tools used in build and deployment packaging.

Building the Platform and its applications

All application instances run on one of the J2SE editions of Java runtime. The output binaries for the application modules vary in content. They are however built using a common tool.

Application source code is maintained in SVN (source control) and is broken up into a number of projects. Multiple projects may be combined to create an application binary. SVN projects are created to abstract related set of modules or features that can be independently included in another application. As a general practice, any functionality that may be reused is a candidate project and multiple such related features eventually become one.

Apache Maven is used to build all SVN projects. The .pom (Project Object Model) build file is used to specify the output binary format for a project, its dependencies and the build order to be followed. It is a powerful build system with multiple plug-ins and extensions available to suit project needs. It comes with pre-defined targets for performing certain well defined tasks such as compilation of code, binary packaging and running test harnesses.

Linux RPM deployment packaging

All application instances are packaged and deployed as Linux RPMs. The term RPM is used to refer to both the installation package and the package management tool used to create it. The RPM Package Manager is a program used for installing, uninstalling and managing software packages in Linux.

The use of RPM provides lots of advantages like easy program installation, un-installation and in deploying updates. The output of the build process is a ".rpm" file (per each deployment unit) which typically includes the compiled programs and dependent libraries.

4.5.2 Business Intelligence & Reporting

Platform standardizes on analytics and reporting technology that integrates well with the rest of the stack and at the same time meets most of the feature needs of applications built on the Platform. BI architecture consists of the three broad sections of data acquisition, Data storage and Data distribution platform; all of which can be considered to be part of an over-arching data warehouse strategy.

- Data acquisition includes all source systems (enrolment, authentication, and other application modules) that feed data into the data warehouse. It is necessary to have the BI system integrated with all the process to ensure consistency of data across multiple operational data sources.
- Atomic Data Warehouse (ADW) is the repository that contains all data in its granular details. It is important to note that data storage for reporting and analytics should be separated from the core transaction data (data that is part of the live production systems). The advantage of the highly de-normalized analytics data being completely separated is to ensure scalability and make least impact to production systems. Within Aadhaar ADW, no information that can be personally identifiable (PII) is stored.
- Data distribution platform provides access to derived data and knowledge to end users (UIDAI officials, partners, and public). This data is presented to the end users and general public, in a timely manner, while still protecting privacy, confidentiality, and security.

Aadhaar BI system employs highly scalable, distributed, reliable and open source technology components to meet the requirements. Aadhaar BI platform consists of an Atomic Data Warehouse (ADW) consisting of granular level analytics data and tools to load data from source systems into ADW (both event publishing via messaging layer as well as offline data extraction via ETL tools). In addition to ADW, BI platform consists of a "data distribution platform" that enables provisioning of data through various datasets and an "analytical delivery platform" that delivers relevant metrics, dashboards, and portals. ADW is built using Hadoop Hive to handle "big data" storage requirements and Pentaho platform to manage all the data in the back-end (ETLs, Access, Reporting, etc).

BI platform caters to long term data analytics and ADW stores data that is not changing. Operational data, on the other hand, refers to in-process and volatile information. Aadhaar system maintains an Operational Data Store (ODS) that provides this information on an "end of day" basis for such volatile operational data. This separation of the Operational and BI data helps UIDAI meet the short and long term reporting and analytics requirements.

The logical view of the BI platform is given below.

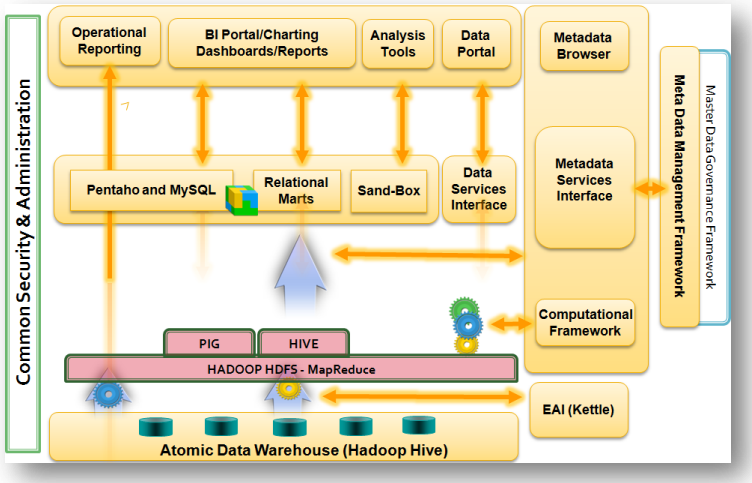


Figure 22: BI Technology Platform

4.5.3 System & Application Monitoring (NoC)

The following key principles are adhered to in the solution architecture and technology selection:

- **Non-intrusive monitoring**: The monitoring solution must be least intrusive with respect to its interface with the components being monitored. Integration with monitoring agents and server uses standard and ubiquitous protocols like HTTP and JMX. This also obliterates vendor lock-in with the chosen third party monitoring tool.

- **Minimal overheads due to monitoring**: The monitoring agents and interfaces must have almost negligible overheads on the components being monitored. This guideline also applies to use of associated infrastructure like network bandwidth and storage.
- **Data correctness**: The information collected by the monitoring system must be correct. It may however not be accurate when compared to point in time information stored in the system's database. This trade off reduces the monitoring overheads while not compromising the premise of monitoring.
- **Aiding system trouble shooting**: The primary purpose of the monitoring solution is to identify, proactively where possible, potential issues in system processing of business transactions i.e. enrolments, authentications etc. This ensures that the information gathered and analyzed by the monitoring agents and the server is kept to a minimum.

Figure 23: Application Monitoring in NoC

System monitoring differentiates the infrastructure and application monitoring:

- **Application monitoring**: Includes metrics exported by custom built application components like enrolment, authentication, etc.
- **Infrastructure monitoring**: Includes metrics available from Operating system processes (CPU, Memory utilization etc) and infrastructure services.

5 Key Lessons and Conclusion

Aadhaar system is built purely as an identity platform that other applications, Government and private, can take advantage of. A sound strategy and a strong technology backbone enabled the program to be launched ahead of plan in Sept 2010 and reach the kind of scale that was never achieved in any biometric identity systems across the world. In less than 4 years since launch, Aadhaar system has grown in capability and more than 600 million Aadhaar numbers have been issued so far using the system.

Entire technology architecture behind Aadhaar is based on principles of openness, linear scalability, strong security, and most importantly vendor neutrality. Aadhaar software currently runs across two of the data centres within India managed by UIDAI and handles 1 million enrolments a day and at the peak doing about 600 trillion biometric matches a day to meet its peek needs. In coming years Aadhaar system will cover rest of the country proving identity to more than 1.2 billion residents in India and its electronic usage (Authentication, e-KYC, etc) is expected to grow exponentially.

Architecting Aadhaar system keeping the scale, security, ecosystem design, and most importantly the constraints of an e-Governance system, has been a learning experience. Following are the architectural and design lessons learned from Aadhaar project. Some of them may be obvious to many, but, nevertheless they are still important to be listed down for completeness purpose. These points ought to be kept in mind by architects especially when working on other large scale e-Governance projects.

5.1 Key Lessons

1. **Define and use APIs for all components** – Insist on API based, loosely coupled modules from day one. All functional components that you write must be encapsulated in a reusable API (preferably REST style for inter system usage) driven and must have an automated test suite. You will be proven wrong if you assume components will not be re-factored or re-written. If system needs to be adaptable and flexible, code will need to be rapidly re-factored. Hence having an API driven approach with strong regression test suite on top of these APIs are critical to such flexibility.
2. **Embrace open source, eliminate vendor lock-in** – As proven in many very large scale Internet applications and in Aadhaar project, open source and free does not mean it is not enterprise grade, it works well and it scales well. So embrace it fully! In addition, especially for e-Governance projects, using open source means you have control over code that was seen and used by potentially 1000's of developers across the world. This means higher security compared to proprietary & closed codebase. If you really need to procure special purpose components for which no open source alternates are available, ensure you define a standard, open interface for all vendors proving that component to adhere. Eliminating vendor lock-in in e-Governance projects is utmost important.
3. **Create single source of truth** – When managing data sources (master data or transactional data), it is critical to wrap the core functionality (read, update) into APIs and ensure all modules requiring that data always access it through those APIs. Also, ensure business logic is centralized into APIs to avoid inconsistent information access and interpretation across modules. People always expect consistent truth from the application rather than absolute truth at that instance of time. Data islands, duplicated data, and data inconsistency can cause huge issues with business logic, data visibility, and decision making.
4. **Believe in data** – For any large scale e-Governance project, collection of granular usage and process metrics are very critical to ensure all decision making is done using intelligence derived from data rather than gut feel and anecdotes. Architects should ensure application collect all kinds of process

data (who, when, where, app screen usage, clicks, exceptions, issues, for that matter anything measureable!) using application instrumentation, store in big data analytics systems, derive metrics and intelligence, and make it available to decision makers.

5. **Build failure resilient system** – When running large scale software system, many components will fail including hardware in your daily operations. Expecting highly skilled manpower in operations team to handle such failures is not realistic. Experts are far and few and it is critical that system is built to be able to recover from such failures automatically. Unless system recovers, retries transactions, and sort of self-heals, there is no way a large system can successfully function.
6. **Security and privacy should be by design** – For any national system, especially applications containing resident personal information, transaction information, it is critical that data security and privacy are designed as part of strategy and architecture rather than as an afterthought. Encryption, hashing, anonymization, auditing, access control, etc are essential for all applications that are built today even if data is managed by people that you trust.
7. **Scalability can't be bought, architect for it** – One thing that is fundamental in building a large scale application is that "scalability" comes from the design patterns and architecture of each and every component within the system. It is silly to assume that usage of a large scale COTS product can somehow scale the application. No single product or vendor can promise scalability rather, it is in the architecture!
8. **Create an ecosystem** – All e-Governance applications should be built with open APIs and multi-vendor ecosystem strategy. As an architect, if you do not find at least two vendors offering same capability/technology, do not use it unless it is completely open source. Whenever you need to use a proprietary module, build an open API and ensure you wrap the vendor product specific APIs behind it to ensure plug-n-play approach and ability to use an alternate product in future.
9. **Open scale-out is the only way** – No large scale system should be built assuming homogenous computing, storage architecture. Applications must assume availability of completely heterogeneous, multi-vendor,

virtualized/non-virtualized systems and ensure applications can scale across such environment. Usage of commodity computing, built to grow in linear fashion, ability to procure compute/storage as and when required, etc are critical to ensure you take advantage of rapid advancements and price reduction in technology. Nothing else other than an "open scale-out" strategy you should agree to!

10. **Keep it simple** – Finally, keep it simple, minimal, avoid scope creep, and be ready to rewrite/re-factor components as they evolve. Build a strong architecture foundation, build strong generic components, build open APIs and interfaces, and let individual features evolve over time on top of these.

References

[1] UIDAI, "UIDAI Strategy Overview," http://uidai.gov.in/UID_PDF/Front_Page_Articles/Documents/Strategy_Overveiw-001.pdf, 2011.

[2] UIDAI, "Aadhaar Web Portal," https://portal.uidai.gov.in/, 2009-2014.

[3] UIDAI, "Role of Biometric Technology in Aadhaar," http://uidai.gov.in/images/FrontPageUpdates/role_of_biometric_technology_in_aadhaar_jan21_2012.pdf, 2012.

[4] UIDAI, "Role of Biometric Technology in Aadhaar Authentication," http://uidai.gov.in/images/role_of_biometric_technology_in_aadhaar_authentication_020412.pdf, 2012.

[5] UIDAI, "Aadhaar Technology Strategy," http://uidai.gov.in/images/AadhaarTechnologyStrategy_mar2014.pdf, 2014.

[6] UIDAI, "Aadhaar Product Document," http://uidai.gov.in/images/AadhaarProductDoc_mar2014.pdf, 2014.

[7] UIDAI, "UIDAI Background," http://uidai.gov.in/all-about-uidai/uidai-background.html, 2009.

[8] UIDAI, "The Demographic Data Standards and verification procedure Committee Report," http://uidai.gov.in/UID_PDF/Committees/UID_DDSVP_Committee_Report_v1.0.pdf, 2009.

[9] UIDAI, "Aadhaar Enabled Service Delivery," http://uidai.gov.in/images/authDoc/whitepaper_aadhaarenabledservice_delivery.pdf, 2012.

[10] UIDAI, "UID Numbering Scheme," http://uidai.gov.in/UID_PDF/Working_Papers/A_UID_Numbering_Scheme.pdf, 2010.

[11] UIDAI, "The Biometrics Standards Committee Report," http://uidai.gov.in/UID_PDF/Committees/Biometrics_Standards_Committee_report.pdf, 2010.

[12] UIDAI, "Aadhaar Automated Biometric Identification System (ABIS) API," http://uidai.gov.in/UID_PDF/Working_Papers/Aadhaar_ABIS_API.pdf, 2010.

[13] UIDAI, "Aadhaar Authentication API 1.6," http://uidai.gov.in/images/FrontPageUpdates/aadhaar_authentication_api_1_6.pdf, 2012.

[14] UIDAI, "Aadhaar Best Finger Detection (BFD) API 1.6," http://uidai.gov.in/images/FrontPageUpdates/aadhaar_bfd_api_1_6.pdf, 2012.

[15] UIDAI, "Aadhaar One-Time-Request (OTP) API 1.5," http://uidai.gov.in/images/FrontPageUpdates/aadhaar_otp_request_api_1_5.pdf, 2012.

[16] UIDAI, "Aadhaar E-KYC API 1.0," http://uidai.gov.in/images/aadhaar_kyc_api_1_0_170912.pdf, 2012.

[17] UIDAI, "Aadhaar E-KYC Service - Policy," http://uidai.gov.in/images/ekyc_policy_note_18122012.pdf, 2012.

[18] Data.Gov, "Data Portal for Government of India," http://data.gov.in/, 2012.

[19] UIDAI, "Aadhaar Biometric Capture Device API," http://uidai.gov.in/UID_PDF/Working_Papers/UID_Biometrics_Capture_API_draft.pdf, 2010.

[20] Wikipedia, "Universally Unique Identifier," http://en.wikipedia.org/wiki/Universally_unique_identifier, 2012-2013.

[21] Oracle, "Java 6 UUID Class," http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html, 2011-2013.

[22] UIDAI, "UIDAI Data Update Policy ver 2.1," http://uidai.gov.in/images/update_policy_version_2_1.zip, 2012.

[23] UIDAI, "Aadhaar Authentication Operating Model," http://uidai.gov.in/images/authDoc/d3_1_operating_model_v1.pdf, 2012.

[24] UIDAI, "Aadhaar Authentication Framework," http://uidai.gov.in/images/authDoc/d2_authentication_framework_v1.pdf, 2012.

[25] UIDAI, "Aadhaar Authentication Security Model," http://uidai.gov.in/images/authDoc/d3_4_security_policy_framework_v1.pdf, 2012.

[26] W3C, "Extensible Markup Language (XML)," http://www.w3.org/XML/, 2012.

[27] Google, "Protocol Buffers," https://code.google.com/p/protobuf/, 2013.

[28] UIDAI, "Role of Biometric technology in Aadhaar Authentication," http://uidai.gov.in/images/role_of_biometric_technology_in_aadhaar_authentication_020412.pdf, 2012.

[29] UIDAI, "Iris Authentication Accuracy - PoC Report," http://uidai.gov.in/images/iris_poc_report_14092012.pdf, 2013.

[30] STQC, "Bio-metric Devices Testing and Certification," http://stqc.gov.in/content/bio-metric-devices-testing-and-certification, 2011-2013.

[31] UIDAI, "Aadhaar Biometric SDK API Version 2," http://uidai.gov.in/images/aadhaar_biometric_sdk_api_2_0.pdf, 2012.

[32] UIDAI, "Aadhaar Registered Devices Specification," http://uidai.gov.in/images/aadhaar_registered_devices_1_0.pdf, 2014.

Published by UIDAI Technology Center
Bengaluru, Karnataka, India.