MDFBA: A Mobile Agent and Device Fingerprint-based Authentication Scheme for Enhanced Android Security

Umesh Kumar, PhD
Assistant Professor, Computer Engineering Department,
JC Bose University of Science and Technology,
YMCA, Faridabad, Haryana, India

ABSTRACT
Modern digital ecosystems, from enterprise infrastructures to Internet of Things networks, depend heavily on authentication mechanisms. Based on important performance and security criterias, such as authentication accuracy, latency, scalability, security resilience, computational overhead, and network traffic generation, this study assesses three authentication models: OAuth, Zero Trust, and a Mobile Agent and Device Fingerprint-Based Authentication Scheme (MDFBA). Using mobile agents and signature-based authentication, the implemented algorithm ensures low latency (1.23 ms) and minimum traffic overhead (~5,000 bytes for 100 nodes), making it ideal for resource-constrained situations. OAuth provides excellent scalability and interoperability and is commonly used for web authentication and Single Sign-On (SSO). However, it has middling authentication accuracy (~85%) due to security flaws including token theft and phishing attempts. In high-performance networks, its latency (72.8 ms) and traffic generation (~15,000 bytes per 100 nodes) pose scalability issues. The most secure of the three models, Zero Trust, uses dynamic policy enforcement and continuous verification, which makes it extremely resistant to replay, credential theft, and man-in-the-middle attacks. This security feature is limited in low-resource situations due to its high latency (~166.2 ms), high processing needs, and substantial network traffic (~40,000 bytes per 100 nodes). Performance and security are traded off, according to a quantitative examination conducted across different node scales. Zero Trust adds significant processing and network overheads, but it guarantees better security. On the other hand, the Implemented Algorithm balances security and efficiency, making it appropriate for Internet of Things applications, whereas OAuth offers scalability but is still susceptible to attack vectors. This study emphasizes the urgent need for hybrid authentication strategies that combine the scalability of OAuth, the security robustness of Zero Trust, and optimizations based on mobile agents. Future research will examine blockchain-based decentralized identity verification, AI-powered adaptive authentication, and quantum-resistant cryptographic improvements.

Keywords
Device Fingerprint, Android security, malware, data breach, privacy, vulnerabilities, mobile security, security measures, device identification, cybersecurity

1. INTRODUCTION
Authentication and Intrusion detection is the most challenging task in the modern era, as number of internet connected devices has increased to approximately 20 billions in 2025 and these devices are expected to rise upto 39.6 billions in 2033[2]. With the rising number of internet devices, there is a vast increase in the number of cybersecurity attacks like Malware, Denial of Service (DoS) attacks, Phishing, Spoofing, Identity based attacks, Code injection attacks, Supply chain attacks, Insider threats, DNS tunneling, IoT based attacks and many more. So, in the modern world apart from quick and fast internet access, we require a safe and efficient internet access.

This article at hand studies the recent methods of device fingerprinting, its different approaches and different scenarios where it is used. This paper, also explored a new method to authenticate a device on the network which can also be used to detect the intrusion on the network. Proposed method uses mobile agent which is known for its intelligence and decision making capability. Another less known and better feature of mobile agent is its ability to generate less traffic as compared to client server model. I have deeply analyzed the proposed algorithm onmultiple devices and different access points to generate the device signature. Device signature uses some static and dynamic parameters for generation of the signature. Static parameters can be used to authenticate the device or access point on the network whereas dynamic parameters can be very useful for intrusion detection. Finally, it provides android sdk based implementation to collect device parameters to generate device fingerprint. So contribution of this work can be as follows:

- Explore the basics and requirement of device fingerprinting based approach for authentication of device.
- Propose a new kind of device fingerprint collection method using mobile agent.
- Propose a new architecture for authentication of device using device fingerprint.
- Develop an android sdk based solution for collection of device fingerprint.

The rest of this paper is organized as follows: Section 2 presents preliminaries regarding Device Fingerprint and its use for authentication while section 3 offers the related work regarding the paper subject. Section 4 presents the mobile agent based software platform for building the proposed system which illustrates the proposed framework. Section 5 shows the working of the proposed model. Section 5.1 shows the implementation and analysis of the proposed work. The results analysis is described in section 6 and the paper's conclusion and future scope are provided in section 7 and 8.This work is an extended version of my previous publication at IJFGCN [3]. In this extended version, a more detailed analysis, additional experimental results and a novel enhancement to the methodology is proposed.

2. PRELIMINARIES
This section presents the concept of the mobile agent along with the device signature for authentication as preliminaries for the proposed work.

Mobile agent is an autonomous code which can roam inside the network by following multiple nodes which is called itinerary of the mobile agent [4]. This itinerary can be dynamic or static [5]. Mobile agent technology has seen a good amount of growth over last few years. In some scenarios, this technology can be used instead of client server architecture for reduction in traffic around the network[6].

2.1 Mobile Agent Life Cycle

Mobile agent follows a typical life cycle. A mobile agent can have multiple states during its lifecycle. Complete lifecycle is explained in Fig. 1.

Initiate: This is the first state when mobile agent is being initiated.
Active: After initiation, mobile agent is in active state and becomes an autonomous code, which can take decision on its own.
Waiting: Mobile agent can go in waiting state due to some internal or external factors.
Suspend: This is the state which represent that mobile agent is not executing currently.

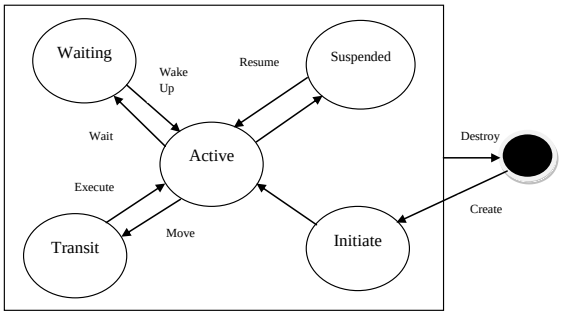

Fig 1:Mobile Agent Lifecycle

Transit: As mobile agent can move from one node to another during its execution, it enters in transit state when it is in move from one node to another.
Deleted: Mobile agent, once finishes its job, can be destroyed.

2.2 Advantages of mobile agent

There are number of advantages of mobile agent as compared to client server model. Some of these are:

Efficient bandwidth utilization: Mobile agent requires less data to be transmitted as compared to original data. Mobile agent can also pre-process data and can transmit compressed information.
Asynchronous: Mobile agents are asynchronous in nature. Once mobile agents are downloaded, they can perform their task asynchronously without any interaction from the parent machine.
Heterogeneous network support: Mobile agent depends only upon the framework in which it is designed with no limitation on underlying architecture. So, mobile agents are separate from the environment in which they are running by the framework in which they are developed.
Increased availability of resources: Mobile agent helps in reduction of traffic so, there will be increase in availability of resources to clients.

3. RELATED WORK

The risk of cybersecurity threats has increased in tandem with the proliferation of internet-connected gadgets. In manynetwork environments, device fingerprinting has become an essential method for intrusion detection and authentication. The methods, difficulties, and uses of recent developments in device fingerprinting are examined in this overview of the literature.

Device Fingerprinting Techniques and Approaches

Device fingerprinting is the process of identifying distinctive features of devices in order to facilitate intrusion detection and authentication. Numerous studies have presented innovative strategies to raise the precision and resilience of device fingerprinting methods. A scalable and cross-domain RF device fingerprinting technique that improves device identification accuracy across domains was proposed by Zhao et al. (2024). In a similar vein, Sánchez et al. (2024) investigated how resistant hardware-based and machine learning-based fingerprinting methods are to hostile attacks and suggested countermeasures. DeviceRadar, an online IoT device fingerprinting system that uses programmable switches to increase fingerprinting efficiency in ISP contexts, was introduced by Li et al. in 2024. A different method was used by Heid and Heider (2024), who examined Android apps for device fingerprinting activity and found that tracking methods are common in mobile apps.

Applications of Device Fingerprinting

Device fingerprinting is used in a variety of fields, including lifecycle management and Internet of Things security. Using device fingerprinting, Lin et al. (2024) suggested a lifecycle management architecture for power equipment that would improve critical infrastructure tracking and maintenance. A thorough analysis of radio frequency fingerprinting methods was carried out by Abbas et al. (2024), emphasising their use in secure device identification.

In their study on distributed PV network terminal identification, Lv et al. (2024) showed how fingerprinting approaches improve energy distribution networks' cybersecurity. Similar to this, Xi et al. (2024) improved device identity detection by introducing an adaptable environment for intrinsic security fingerprints.

Enhancements through Machine Learning and Deep Learning

The incorporation of machine learning to improve device fingerprinting has been the subject of numerous studies. DevPF, a passive fingerprinting method for Internet of Things devices, was created by Zhang et al. (2024) and has shown increased accuracy in identifying devices without the need for active user intervention. An intelligent fingerprinting method designed for low-power embedded Internet of Things devices was presented by Kohli et al. (2024), who demonstrated its efficacy in limited settings.

A channel-resilient deep learning-driven fingerprinting technique was presented by Basha et al. (2023), who used numerous data streams to boost robustness. To ensure data integrity and dependability, Cui et al. (2023) used device fingerprinting for trust assessment in mobile crowdsensing. Sang et al. (2023) showed how key-blocks aware techniques enhance security by advancing IoT fingerprinting using proprietary protocol traffic analysis.

Device fingerprinting still faces a number of difficulties despite developments, such as flexibility to changing surroundings, scalability, and durability against hostile attacks. A comparative analysis of fingerprinting techniques for resource-constrained IoT devices was presented by Chowdhury et al. (2022), who also identified important research gaps and problems. Deep convolutional neural networks were used by Aneja et al. (2022) for device fingerprinting, demonstrating the promise of AI-driven techniques to improve security.

Monaco (2022) introduced a new temporal-based identifying technique by investigating fingerprinting using peripheral timestamps. The sensitivity of deep learning-based fingerprinting in LoRa-IoT networks was evaluated by Hamdaoui and Elmaghbub (2022), indicating its potential for protecting extensive IoT deployments. Wan et al. (2022) made it easier to conduct more research in this area by introducing DevTag, a benchmark dataset for fingerprinting IoT devices. Lastly, a semi-supervised deep RF device fingerprinting technique that uses meta-learning to increase classification accuracy was presented by Ren et al. (2022).Table 1 summarizes all the related work.

Table 1: Various Device fingerprinting techniques comparison

Author(s)	Techniques Used	Advantages	Disadvantages
Zhao, Tianya, Xuyu Wang, and Shiwen Mao	Cross-domain RF fingerprinting using deep learning	Scalable and interpretable across different domains	May require large datasets for training
Sánchez, Pedro Miguel et al.	Adversarial attack and defense mechanisms for ML-based and hardware-based fingerprinting	Enhances security against adversarial threats	Increased computational overhead
Li, Ruoyu et al.	IoT device fingerprinting using programmable switches in ISPs	Enables real-time device identification	Dependence on ISP infrastructure
Heid, Kris and Jens Heider	Detection of fingerprinting activity in Android apps	Identifies privacy-invasive fingerprinting techniques	Focused only on Android ecosystem
Lin, Ziqing et al.	Lifecycle management for power equipment via device fingerprinting	Enhances equipment monitoring and reliability	Specific to power equipment applications
Abbas, Sohail et al.	Survey on RF fingerprinting techniques for device identification	Comprehensive analysis of existing methods	Lacks implementation and experimental results
Lv, Zhuo et al.	Distributed PV network terminal identification using fingerprint generation	Improves security in distributed PV networks	Domain-specific application
Xi, Zesheng et al.	Adaptive environment-based intrinsic security fingerprinting	Enhances adaptability to changing environments	Possible complexity in implementation
Zhang, Dahua et al.	Passive fingerprinting for IoT device identification	Energy-efficient and non-intrusive	Limited accuracy in noisy environments
Kohli, Varun, Muhammad Naveed Aman, and Biplab Sikdar	Intelligent fingerprinting for low-power embedded IoT devices	Optimized for resource-constrained environments	Potential trade-offs in accuracy and security
Kumar & Paul	Survey on device fingerprinting for cyber-physical systems	Provides taxonomy and categorization of fingerprinting methods	No practical implementation or evaluation
Basha et al.	Multiple data streams for channel-resilient fingerprinting	Improves robustness against channel variations	High complexity in data processing
Cui et al.	Trust assessment via device fingerprinting	Enhances security in mobile crowdsensing	Depends on the reliability of collected data
Sang et al.	Key-blocks aware fingerprinting for proprietary protocol traffic	Works with proprietary IoT protocols	May not generalize well across different protocols
Chowdhury & Abas	Survey on device fingerprinting for IoT	Comparative study of techniques; highlights research challenges	Lacks experimental validation
Aneja et al.	Deep convolutional neural networks (CNN)	High accuracy in fingerprinting devices	Computationally expensive
Monaco	Peripheral timestamp-based fingerprinting	Low overhead; works with existing hardware	Sensitive to timing variations and environmental factors
Hamdaoui & Elmaghbub	Deep learning-based fingerprinting for LoRa-IoT	Enhances security in LoRa networks	Performance degrades with network topology changes
Wan et al.	DevTag benchmark for IoT fingerprinting	Provides a standard benchmark for evaluation	Limited scope in real-world deployment

4. PROPOSED FRAMEWORK

This section will describe the overview for device fingerprint based authentication model. The Model describes the working of various components like Access Point (AP), Authenticator (A) and Authentication Server (AS). Various mobile nodes are connected to the access point with the help of 802.11 technologies. Architecture allows heterogeneous communication devices to contact each other in a secure manner. Here, the authenticator acts as an interface between the AP and the AS. The authentication process goes through the access point to authenticator and then to the authentication server. Fig 2 shows the architecture and working of Device Signature based Authentication Model. Here, multiple AP's are connected to the Authenticator (A). Authenticator in turn is connected to the Authentication Server (AS).

1. Access Point (AP): Access point provides the network access to multiple devices within the wireless range. Multiple access points can be within the range of a single device. Device is connected to an access point with the strongest signal after registration process (if not registered) and authentication. An access point supports multiple devices.

2. Authenticator (A): All access points are connected to the authenticator for authentication purposes. Authenticator contains three modules:

- **Device Fingerprint Scanner:** It extracts the device fingerprint parameters from the mobile agent and creates fingerprint with the help of those parameters. It stores it in cache memory and forwards it to the Authentication Server for registration purpose. If the device requires authentication, then device fingerprint is forwarded to the cache memory.
- **Cache Memory:** Cache memory stores the fingerprint of most recent devices which are connected to the AP or the devices which are recently registered on the network. Cache memory forwards device fingerprint to the threshold comparator.
- **Threshold Comparator:** It compares two fingerprints and gives true or false depending upon the value of the threshold matched

3. Authentication Server (AS): Authenticator is connected to the Authentication Server. AS provides the necessary authentication information for the devices and access points within the network. It registers all the device signatures in the database and provides the signature value when required by the authenticator.

5. WORKING OF THE PROPOSED MODEL

Proposed model supports both single device authentication and multi device authentication. Step 1 in Figure 2 shows the single device authentication involving only the single device. Multi device authentication will take more time as compared to single device authentication, as in former case mobile agent will have to traverse through multiple devices for device signature extraction. Step 2 in Figure 2 shows the description of multi device authentication involving multiple devices.

Single Device Authentication: Single device authentication is required when a new device comes into the range of an access point and wants to connect to the network. This single device can be either a user mobile device or a fixed access point.

Multi Device Authentication: Proposed model also supports the authentication of multiple devices in one go. Mobile agent from authenticator follows the itinerary for multiple devices and carries the device signatures of these devices. If the signature matches to the calculated one, then it gives success, otherwise failure. This loaded mobile agent then comes to the authenticator along with necessary information.Device fingerprint is a unique signature of a device. This device fingerprint is generated depending upon the various parameters. These parameters will be extracted from the device using mobile agent which is being sent by the Authenticator and executed on the target machine.Various parameters that are used to calculate a device signature value are listed below. It may be the case that device have only some parameters value. Using these available parameters of the device, fingerprint will be calculated.

Parameters to calculate device fingerprint:

Geo location lat/long: Itis the geographic location of a device. The location can be calculated by the network or Global Positioning System (GPS) (if supported).
IP addresses: Itis the unique 32/128 bit address which is allocated to the device. The version of the IP address can be IPv4 or IPv6.
MAC addresses: Itis the Media Access Control (MAC) address which is assigned to the network interface card of the device.
Network ID: Every existing network has a unique id value associated with it. This is the id of the network to which the device is connected.
Browser name and version: Itis the name and version of the browser currently installed. Device may have more than one browser, the details of all are included in the fingerprint.
OS name and version: This is the name and current version of the operating system of the device.
Electronic Serial Number (ESN): This number is generated by the manufacturer on the microchip on mobile devices.
International Mobile Equipment Identity or Mobile Identification Number (IMEI / MIN): It is the unique identification number that all mobile phones have.
Received Signal Strength Indicator (RSSI): With this parameter, the received signal power from access point to mobile device can be measured. RSSI is usually measured in decibels relative to a milliwat (dBm). The stronger the signal is, closer it is to zero.
Basic Service Set Identifier (BSSID): Itis the MAC address of the access point, which is the combination of the organization unique identifier and identifier for the radio chipset.
Service Set Identifier (SSID): Itis the name which is assigned to the wireless local area network. Mobile devices use SSID to identify the network and to join the network also.
CenterFreq: It is the measure of the frequency between upper and lower frequency cutoffs. Arithmetic mean or geometric mean of the lower and upper cutoff frequency is used to define this.
Frequency: Itis the frequency value in MHz. This is the value over which the communication will take place.
Level: Itis the strength of the GSM/CDMA signal received. Lower the level, lower will be the strength of the signal.
Timestamp: Itcontains the time of a particular communication between device and the access point. This value is generally mentioned in microseconds.
VenueName: Itis the name of the location that is distributed by the access point.

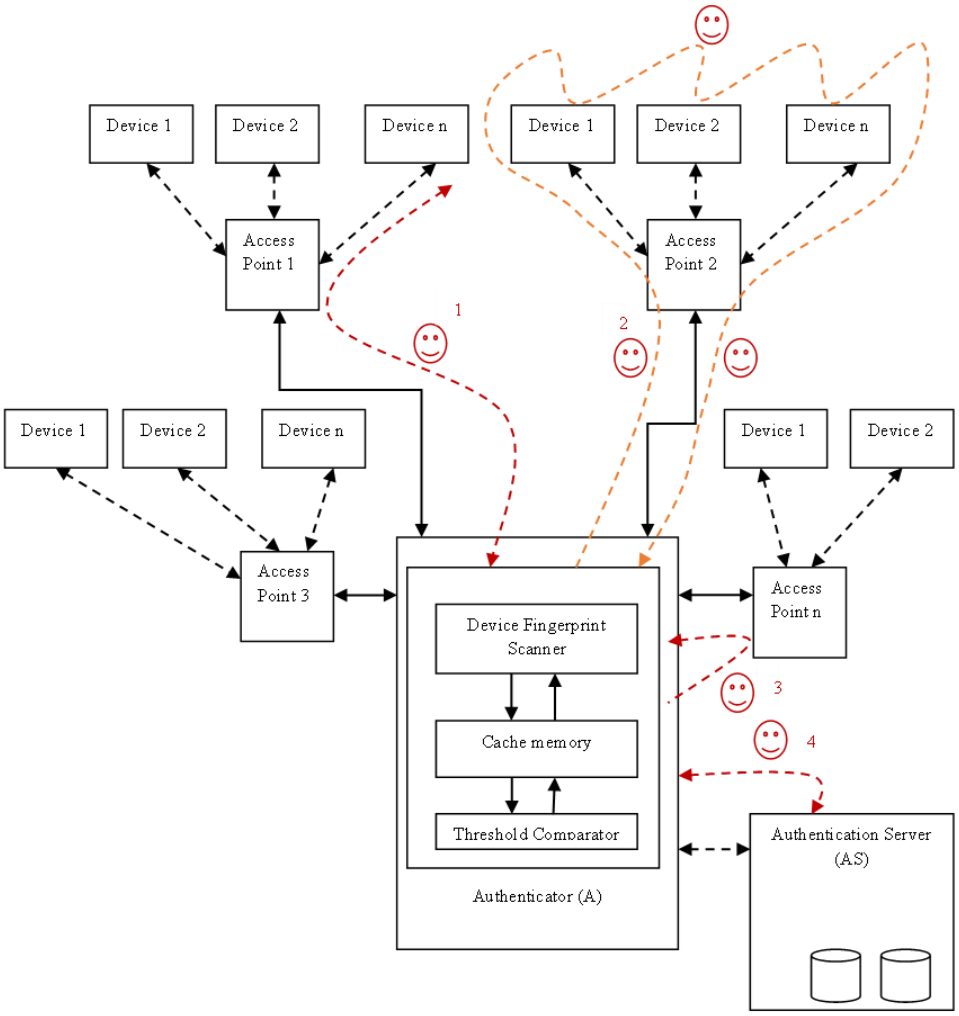

Fig 2:Architecture and working of the proposed model

Device to AP RTT Supported: Itis the value of the inbuilt function supported by the device. Using this function, mobile device can calculate the distance between the access point and device.

isTdlsSupported: It is the value of the tunneled direct link setup. IEEE 802.11z supports this. Its value will be true if supported.

WifiConfiguration.GroupCipher: Itis the cipher mechanism supported by device. Its value can be CCMP which is AES in Counter mode with CBC-MAC or TKIP which is Temporal Key Integrity Protocol.

LinkSpeed: Itis the current speed in Mbps of the channel between device and the access point.

Capabilities: It describes the authentication, key management, and encryption schemes supported by the access point.

Camera characteristics: If a device has camera features, then camera characteristics can also be included in the signature.

Gateway address: Itis the address of the router, which is maintained by Internet Service Provider (ISP).

These parameters are entered in to the device fingerprint generator, which generates the device fingerprint. Some of the parameters mentioned above will give the real time dynamic values like location, IP address, Link Speed and some values will remain unchanged like IMEI number, MAC address and camera characteristics etc.

Every time the mobile agent fetches the device fingerprint parameters, it is compared with the earlier stored one. If the percentage of signature matching is more than threshold, then access can be granted.

The threshold value can be decided by the administrator depending upon the criticality of the application. In the proposed model, a device needs to be registered to the Authentication Server for having access of the network resources. The step by step registration algorithm is shown in Algorithm 1.

Algorithm 1: Registration
Input:
1. Number of Nodes
2. Client addresses i.e. client(i)
3. Authenticator address i.e. authenticator_address
4. Authentication Server address
5. itinerary[ ]
Output: Success or failure of registration.
1.create itinerary[ ] of i number of nodes for MA
2.create MA sign_Collector( )
3.while itinerary[ ] is not empty
    3.1 if node itinerary[i] is active then
    3.1.1 dispatch sign_Collector( ) to the itinerary[i] node.
    3.1.2 compute device signature and append it to the results
    3.2 else

3.2.1 continue
3.3 end if
4. end while
5. authenticator stores all signatures in the cache memory and forwards it to the Authentication Server.

Algorithm 1

After the registration, device can use the network services. If a device comes later on in the network for the use of network services, then the device needs to be authenticated. Mobile agent used for the authentication purpose, collects all the authentication information from either single or multiple devices. The itinerary created in this approach can use the location fingerprinting technique to create the three dimensional graph of various mobile stations. Later on, Local Closest First (LCF) or Global Closest First (GCF) technique is applied to create the itinerary for the mobile agent. Authentication steps are shown in Algorithm 2.

Algorithm 2: Authenticator
Input:
1. Necessary information for authentication process e.g. username, password or device fingerprint
2. Client addresses i.e. client(i)
3. Authenticator address i.e. authenticator_address
4. Authentication server address
5. itinerary[ ]
Output: Success or failure of authentication.

1. create and load mobile agent signature_Verifier( ) with device signature of all the devices.
1.1 For all the devices whose signature are not present at the Authenticator's cache memory
1.1.1 creates a mobile agent signature_Collector(devices [ ]), loads it with the device id's of all devices.
1.1.2 Server loads the signatures from the database.
1.1.3 dispatch signature_Collector( ) to the Authentication Server.
1.1.4 append the signatures to signature_Verifier( ) agent.
1.2 end for
2. create itinerary[ ] of MA using itinerary algorithm like LCF or GCF
3. while itinerary[ ] is not empty
3.1 dispatch mobile agent signature_Verifier( ) from Authenticator to node itinerary [i].
3.2 agent collects and checks the device fingerprint against the stored signature.
3.3 agent gives the success/ failure message depending upon the threshold value of the signature verification result.
4. end while.
5. Last client dispatches the signature_Verifier( ) to authenticator.
6. Authenticator collects success or failure message from mobile agent signature_Verifier( ) based on the device signature verification and provides the access.

Algorithm 2

Authentication Server stores information of all devices that are registered and authorized to use network resources. Algorithm for the same is presented in Algorithm 3.

Bulk data transmission facility which is not provided in traditional EAP based approach has been incorporated into the proposed mobile agent based approach. Agents can be loaded with bulk data and can be dispatched to the destination. Steps for the same are presented in Algorithm 4.

Algorithm 3: Authentication Server
Input:
1. Mobile agent
Output:
Successful retrieval or storage of signature.
Repeat for every request from Authenticator
1. if signature_Collector( ) arrives for registration
1.1 retrieve signatures from mobile agent
1.2 store it into the database.
2. if signature_Collector( ) arrives for device signature
2.1retrieve signatures from database of Authentication Server
2.2 load and dispatch the mobile agent signature_Collector( ) to Authenticator.

Algorithm 3

Algorithm 4: Data_Transmission
Input:
1. Data to be transmitted
2. Destination address (server or client)
Output: Success or failure of data transmission
1. Repeat while client or server has data to send
1.1 create agent Load_Data( ).
1.2 load agent with the requisite data &session_key.
2. dispatch agent to the destination

Algorithm 4

5.1 Implementaion and Analysis

Proposed device signature based authentication mechanism is implemented on Android platform. Mobile application extracts parameters of a device, which helps in generation of signature of a device. This signature can be used in one form or another to authenticate the device on to the network. Figure 3,4 shows the mobile application screenshots used to capture the parameters needed to generate device signature.Using this application, device signatures of multiple devices are collected as shown in Figure 5.These device signatures comprising of multiple device characteristics can be used to identify the device on the network in one form or another.

5.2 Authentication using Device Signature

Captured device signature parameters can be used in many forms for authentication in addition to the currently existing authentication protocols like EAP.

Some of the parameters usages are:

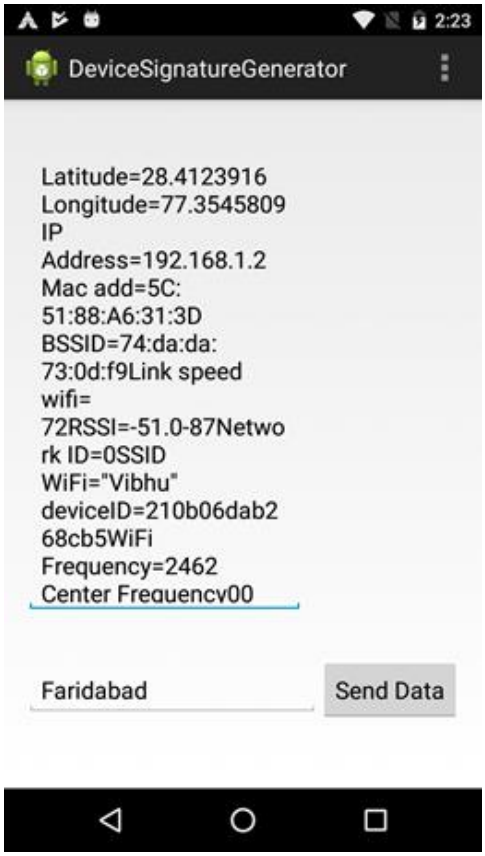

Fig 3:Application showing device signature parameters on Motorola device

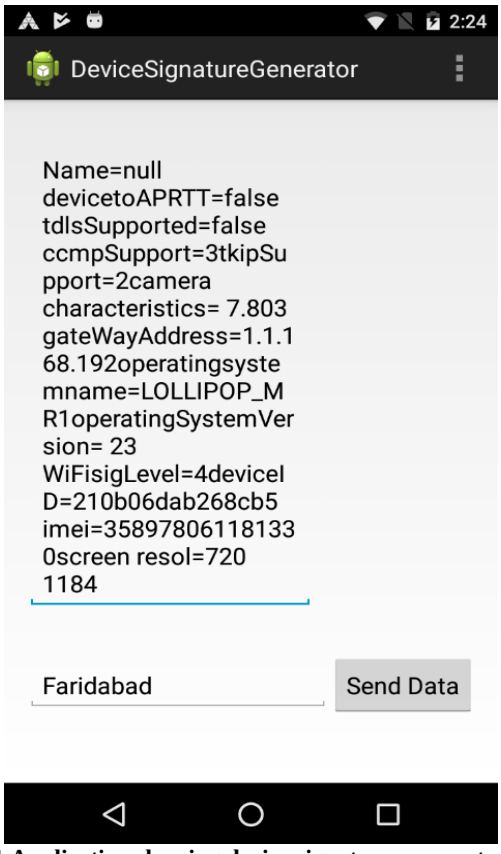

Fig 4:Application showing device signature parameters on Motorola device

Location: Figure 6 shows the MyLocationListener class, which is used to capture the current location of the mobile device. Location captured using this class can be used to check whether a device is within the range of the Access Point or not. Figure 7 shows the snapshot of MyLocationChecker class which can be used to compare the location of the device and already stored location of the Access Point. This class compares the current device location if it is within the 500 meter radius of the access point or not. This check can be very useful for checking an unauthorized person or device pretending to be a valid one.

```
class MyLocationListener implements LocationListener {
        Context mycontext = null;
        public MyLocationListener(Context context) {
                // TODO Auto-generated constructor stub
                mycontext = context;
        }
        @Override
        public void onLocationChanged(Location loc) {
                Log.d("Location changed",
                                loc.getLatitude() + ", "
+ loc.getLongitude());
                String longitude = "Longitude: " +
loc.getLongitude();
    Log.v("long=", longitude);
    String latitude = "Latitude: " + loc.getLatitude();
    Log.v("lat=", latitude);
                String cityName = null;
                Geocoder gcd = new Geocoder(mycontext,
Locale.getDefault());
                List<Address> addresses;
    try {
      addresses = gcd.getFromLocation(loc.getLatitude(),
        loc.getLongitude(), 1);
      if (addresses.size() > 0)
        System.out.println(addresses.get(0).getLocality());
      cityName = addresses.get(0).getLocality();
    } catch (IOException e) {
      e.printStackTrace();
    }
    String s = longitude + "\n" + latitude + "\n\nMy Current
City is: "
        + cityName;
                Log.d("Location=", s);
            }
}
```

Fig6: Location Listener

```
Public class MyLocationChecker {
        public boolean checkLocationWithinRadius(Location
deviceLocation,
                        Location accessPointLocation) {
                float[] dist = new float[1];
                if (deviceLocation != null &&
accessPointLocation != null) {
        Location.distanceBetween(deviceLocation.getLatitud
e(),
        deviceLocation.getLongitude(),
                accessPointLocation.getLatitude(),
                accessPointLocation.getLongitude(), dist);
                        if (dist[0] / 500 > 1) {
        // If device Location is outside 500m radius area
                                return false;
                        } else
                                return true;
                }
```

```
                return false;
        }
}
```

Fig 7: Location Checker

IP Address: IP address of the device can also be one of the parameter for authentication. Whenever a network is created, a range of IP addresses are assigned to it. With the help of device IP address, algorithm can check whether the device IP address is within the valid range or not. If it is not within the valid range, then mobile device can be blocked. Class is shown in Fig 8.

```
public class IPRangeChecker {
        public static long ipToLong(InetAddress ip) {
                byte[] octets = ip.getAddress();
                long result = 0;
                for (byte octet : octets) {
                        result <<= 8;
                        result |= octet & 0xff;
                }
                return result;
        }
        public static boolean isValidRange(String ipStart,
String ipEnd, String ipToCheck) {
                try {
                        long ipLo =
ipToLong(InetAddress.getByName(ipStart));
                        long ipHi =
ipToLong(InetAddress.getByName(ipEnd));
                        long ipToTest =
ipToLong(InetAddress.getByName(ipToCheck));
                        return (ipToTest >= ipLo &&
ipToTest <= ipHi);
                } catch (UnknownHostException e) {
                        e.printStackTrace();
                        return false;
                }
        }
        public static void main(String[] args) {
        System.out.println(isValidRange("122.170.122.0",
"122.170.122.255","122.170.122.215"));
        }
}
```

Fig 8: Valid IP Range Checker

Sr. No.	Latitude	Longitude	IP address	Mac Address	Device ID	IMEI	RSSI (dB)	Network ID	gateway address	camera characteristics	Screen_Resolution	Operating System (OS)	OS Version	WiFi Signal Level
1	28.3673571	77.3164947	10.0.124.68	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-36 -74	1	200.0.0.10	7.803	720*1184	LOLLIPOP_MR1	23	1
2	28.3673867	77.3164891	10.0.102.81	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-36.0-79	1	200.0.0.10	7.803	720*1184	LOLLIPOP_MR1	23	4
3	28.36661523	77.31622427	10.0.102.83	C4:0B:CB:6E:74:4D	92ad48e64838558f	863194034228707	-127 -83	1	200.0.0.10	0.1	1080*1798	M	24	4
4	28.36786543	77.31509557	192.168.0.106	D8:32:E3:6E:F4:9D	707d9a7021b2f189	869048031585622	-69 -92	9	1.0.168.192	0.1	1080*1798	N	25	4
5	28.4125847	77.3547662	192.168.1.4	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-55.0-89	1	1.1.168.192	7.803	720*1184	LOLLIPOP_MR1	23	4
6	28.36936204	77.31673351	192.168.0.102	00:EC:0A:9A:5A:A9	36636b40942cd957	863675038302563	-50-105	70	1.0.168.192	0.1	1080*1798	N	25	4
7	28.412209	77.3545501	192.168.1.4	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-44.0-87	1	1.1.168.192	7.803	720*1184	LOLLIPOP_MR1	23	4
8	28.3673979	77.3164271	10.0.103.204	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-127-67	1	200.0.0.10	7.803	720*1184	LOLLIPOP_MR1	23	4
9	28.3674017	77.3164239	10.0.103.204	5C:51:88:A6:31:3D	210b06dab268cb5	358978061181330	-50-73	2	200.0.0.10	7.803	720*1184	LOLLIPOP_MR1	23	4
10	33.7885096	151.075712	192.168.0.5	80:58:F8:1E:F5:D7	dc898264b08c544f	351892081906096	-63-92	17	1.0.168.192	11.907	1080*1798	O	27	3

Fig 5: Multiple Android Device Signatures (Access Point)

MD5 Fingerprint: From the device signature static and dynamic values are extracted as shown in Table 2 and Table 3. Static values are the values which are not going to change throughout the life of the device. These static values of a device are taken to produce MD5 Fingerprint of the device. Similarly, static signature values for AP can be used as shown in Table 2 to produce the MD5 Fingerprint of the access point. The fingerprint for Access Point shown in Table 4 can be used to capture the fake access point into the network.Advantage of creating MD5 Fingerprint is to create the MD5 Fingerprint from static device or access point parameters but reverse is not possible i.e. device or access point parameters cannot be extracted from the MD5 Fingerprint. Apart from the MD5 Fingerprint, Hamming distance can be used to match and compare two mobile device signatures.

Table 2: Static parameters of device

Sr. No.	Static Parameters	
1	Mac Address	**Device**
2	Device ID	
3	IMEI	
4	Camera Characteristics	
5	Screen Resolution	
6	Operating System	
7	Operating System Version	

Table 3: : Dynamic parameters of device

Sr. No.	Dynamic Parameters	
1	Latitude	**Device**
2	Longitude	
3	IP address	

Table 4: Static parameters of access point

Sr. No.	Static Parameters	
1	BSSID	**AP**
2	SSID	
3	Frequency	
4	tdls supported	
5	ccmp support	
6	tkip support	

Hamming Distance: Hamming distance of two device signatures can also be one of the mechanism for identification of mobile device. Distance between two signatures is measured using the hamming distance formula.

Hamming distance gives minimum number which is required for a string to be similar with other. For device signature matching, hamming distance based formula is used as shown in equation 1.

$$d(x,y) = \frac{\sum_{i=1}^{F}(x_i! = y_i)}{F}, \qquad 0 \le d(x,y) \le 1 \quad \ldots(1)$$

Where, F is the number of features and x_i and y_i represents i[th]

feature of the device signature stored and extracted respectively. If x_i equals y_i, then it contributes 1 to the summation.
If the device signature parameters are different, only then they will contribute to distance.
Let's compare two signatures of same device using the equation 1.

$$x = \{x_1, x_2, x_3, \ldots . x_n\}$$
$$y = \{y_1, y_2, y_3, \ldots . y_n\}$$

Where x represents device signature stored in the database during registration and y represents the signature supplied by the mobile agent when it tries to connect to the network.

x = {28.4125847, 77.3547662, 192.168.1.4, 5C:51:88:A6:31:3D, 210b06dab268cb5, 358978061181330, 74:da:da:73:0d:f9, 72, -55.0-89, 1,Vibhu, 2452, FALSE, 3, 2, 200.0.0.10, 0.192, 720*1184, LOLLIPOP_MR1, 23, 1}

y={28.412209, 77.3545501, 192.168.1.4, 5C:51:88:A6:31:3D, 210b06dab268cb5, 358978061181330, 74:da:da:73:0d:f9, 72, -44.0-87, 1, Vibhu, 2452, FALSE, 3, 2, 200.0.0.10, 0.192, 720*1184, LOLLIPOP_MR1, 23, 1}

Using the equation 1 hamming distance of the signatures x and y is calculated. The Java code for the same is shown in Figure 10. Output of the same i.e. the hamming distance is 0.1428571492433548. This hamming distance tells about the mismatch of the two signatures. Larger the hamming distance value, larger the mismatch is. This hamming distance can be used to identify the signature of mobile device.Figure 7 shows the MainActivity of the Android based mobile application.

```
public class MainActivity extends ActionBarActivity {
        LocationManager locationManager;
        WifiManager wifiManager;
        WifiInfo wifiInfo;
        TelephonyManager telephonyManager;
        CellInfoGsm cellinfogsm;
        CellInfo cellInfo;
        CellSignalStrengthGsm cellSignalStrengthGsm;
        CameraManager cameraManager;
        DhcpInfo dhcpInfo;
        deviceSignature mySignature;
        LocationManager mLocationManager;
        Location myLocationl;
        Geocoder geocoder;
        List<Address> addresses;
        String cityName;
        EditText mEdtTxtSgn;
        EditText mEdtTxtLctn;
        Button btnSndDta;
        String data;
        float megaPixel = -1;

        @Override
        protected void onCreate(Bundle savedInstanceState)
{
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                if (android.os.Build.VERSION.SDK_INT
> 9) {
                        StrictMode.ThreadPolicy policy
= new StrictMode.ThreadPolicy.Builder().permitAll().build();
                StrictMode.setThreadPolicy(policy);
                }

                mySignature = new deviceSignature();
                mySignature.screenDisplay = new
ScreenDisplay();
                wifiManager = (WifiManager)
getSystemService(WIFI_SERVICE);
                locationManager = (LocationManager)
getSystemService(LOCATION_SERVICE);
                telephonyManager = (TelephonyManager)
this

        .getSystemService(Context.TELEPHONY_SERVIC
E);
                wifiInfo =
wifiManager.getConnectionInfo();
                cameraManager = (CameraManager)
getSystemService(CAMERA_SERVICE);
                dhcpInfo = wifiManager.getDhcpInfo();
                getDeviceLocation();
                getIPAddress();
                getRSSI_WiFi();
                getMacAddress();
                getBSSID();
                getWiFiLinkSpeed();
                getNetworkID();
                getSSID();
                getIMEI();
                getWiFiFrequency(); // in MHz
                getdevicetoAPRTT();
                getTDLSValue();
                getGroupCipherValue();
                getCameraInfo();
                getGatewayAddress();
                getOperatingSystem();
                getAllCellInfo();
                getWiFiSignalLevel();
                getDeviceID();
                getScreenDisplay();
                writeToFile();
                mEdtTxtSgn = (EditText)
findViewById(R.id.edtTxtDvcSgntr);
                mEdtTxtSgn.setText(data);
                mEdtTxtLctn = (EditText)
findViewById(R.id.edtTxtLocation);
                mEdtTxtLctn.setText(cityName);
                btnSndDta = (Button)
findViewById(R.id.sndData);
                btnSndDta.setOnClickListener(new
OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                final Intent emailIntent
= new Intent(
        android.content.Intent.ACTION_SEND);
        emailIntent.setType("text/plain");
        emailIntent.putExtra(android.content.Intent.EXTRA_
EMAIL, new String[] { "umesh554@gmail.com" });
        emailIntent.putExtra(android.content.Intent.EXTRA_
SUBJECT, "My Device Signature");
        emailIntent.putExtra(android.content.Intent.EXTRA_
TEXT, data);
        emailIntent.setType("message/rfc822");
                                        try {
        startActivity(Intent.createChooser(emailIntent,
        "Send email using..."));
                                        }
catch (android.content.ActivityNotFoundException ex) {
                                        }
                                }
                        });
```

```
	}
	private void getScreenDisplay() {
		Display display =
getWindowManager().getDefaultDisplay();
		Point size = new Point();
		display.getSize(size);
		mySignature.screenDisplay.setData(size.x,
size.y);
	}
	private void getDeviceID() {
		mySignature.deviceID =
Secure.getString(getApplicationContext()
	.getContentResolver(), Secure.ANDROID_ID);
	}
	private void getWiFiSignalLevel() {
		int x = (int) mySignature.rssi_WiFi;
		// Integer.valueOf(mySignature.rssi_WiFi);
		mySignature.wifiSignalLevel =
WifiManager.calculateSignalLevel(x, 5);
	}

	private void getAllCellInfo() {
		List<CellInfo> cellInfos =
telephonyManager.getAllCellInfo();
		if (cellInfos != null) {
			for (int i = 0; i < cellInfos.size();
i++) {
				if
(cellInfos.get(i).isRegistered()) {if (cellInfos.get(i) instanceof
CellInfoWcdma) {	CellInfoWcdma
cellInfoWcdma = (CellInfoWcdma) telephonyManager

	.getAllCellInfo().get(0);
	CellSignalStrengthWcdma cellSignalStrengthWcdma
= cellInfoWcdma
	.getCellSignalStrength();
	mySignature.rssi_Cell =
String.valueOf(cellSignalStrengthWcdma.getDbm());
				} else if
(cellInfos.get(i) instanceof CellInfoGsm) {
			CellInfoGsm cellInfogsm =
(CellInfoGsm) telephonyManager.getAllCellInfo().get(0);
CellSignalStrengthGsm cellSignalStrengthGsm = cellInfogsm

	.getCellSignalStrength();
	mySignature.rssi_Cell = String
	.valueOf(cellSignalStrengthGsm.getDbm());
				} else if
(cellInfos.get(i) instanceof CellInfoLte) {
		CellInfoLte cellInfoLte = (CellInfoLte)
telephonyManager
	.getAllCellInfo().get(0);
	CellSignalStrengthLte cellSignalStrengthLte =
cellInfoLte
	.getCellSignalStrength();
	mySignature.rssi_Cell = String
		.valueOf(cellSignalStrengthLte.getDbm());
				}
			}
		}
	}
	}

	private void writeToFile() {
		try {
			OutputStreamWriter
outputStreamWriter = new OutputStreamWriter(
```

```
		openFileOutput("config.txt",
Context.MODE_APPEND));
			data = "Latitude=" +
Double.toString(mySignature.latitude)
			+ "
Longitude=" + Double.toString(mySignature.longitude)
			+ " IP
Address=" + mySignature.ipAddress + " Mac add="
			+
mySignature.macAddress + " BSSID=" + mySignature.BSSID
			+ "Link
speed wifi= "
			+
Integer.toString(mySignature.linkSpeed_WiFi) + "RSSI="
			+
Double.toString(mySignature.rssi_WiFi)
			+
mySignature.rssi_Cell + "Network ID="
			+
Integer.toString(mySignature.networkID_WiFi)
			+ "SSID
WiFi=" + mySignature.SSID_WiFi + " deviceID="
			+
mySignature.deviceID + "WiFi Frequency="
			+
Integer.toString(mySignature.wifiFrequency)
			+ " Center
Frequency0"
			+
Integer.toString(mySignature.centerFrequency0)
			+ " Center
Frequency1="
			+
Integer.toString(mySignature.centerFrequency1)
			+ "
Capabilities=" + mySignature.capabilities
			+ "Venue
Name=" + mySignature.venueName + " devicetoAPRTT="
			+
Boolean.toString(mySignature.devicetoAPRTT)
			+ "
tdlsSupported="
			+
Boolean.toString(mySignature.tdlsSupported)
			+ "
ccmpSupport="
			+
Integer.toString(mySignature.ccmpSupport)
			+
"tkipSupport="
			+
Integer.toString(mySignature.tkipSupport)
			+ "camera
characteristics= " + mySignature.maxResolution
			+ "
gateWayAddress=" + mySignature.gatewayAddress
			+
"operatingsystemname=" + mySignature.operatingSystemName
			+
"operatingSystemVersion= "
			+
Integer.toString(mySignature.operatingSystemVersion)
			+ "
WiFisigLevel=" + mySignature.wifiSignalLevel
			+
"deviceID=" + mySignature.deviceID + " imei="
```

```
                                                +
mySignature.imei + "screen resol=" + ""
                                                +
mySignature.screenDisplay.getData();
                                Log.i("mySign", data);
                                Log.d("mySign", data);
                                Log.i("mySign", data);
                                //
outputStreamWriter.write(data);
                                outputStreamWriter.close();

                        } catch (IOException e) {
                                Log.e("Exception", "File write
failed: " + e.toString());
                        }
                }

                private void getOperatingSystem() {
                        mySignature.operatingSystemName =
android.os.Build.VERSION_CODES.class

                .getFields()[android.os.Build.VERSION.SDK_INT].
getName();
                        mySignature.operatingSystemVersion =
android.os.Build.VERSION.SDK_INT;

                }

                private void getGatewayAddress() {
                        mySignature.gatewayAddress =
mySignature.intToIp(dhcpInfo.gateway);
                }

                private void getGroupCipherValue() {
                        mySignature.ccmpSupport =
WifiConfiguration.GroupCipher.CCMP;
                        mySignature.tkipSupport =
WifiConfiguration.GroupCipher.TKIP;
                }

                private void getTDLSValue() {
                        mySignature.tdlsSupported =
wifiManager.isTdlsSupported();
                }

                private void getdevicetoAPRTT() {
                        mySignature.devicetoAPRTT =
wifiManager.isDeviceToApRttSupported();
                }

                private void getWiFiFrequency() {
                        mySignature.wifiFrequency =
wifiInfo.getFrequency();
                        int channel_number = 0;
                        if (mySignature.wifiFrequency == 2484) {
                                channel_number = 14;
                                Log.i("channel_Number=", "" +
channel_number);
                                return;
                        }
                        if (mySignature.wifiFrequency < 2484) {
                                channel_number =
(mySignature.wifiFrequency - 2407) / 5;
                                Log.i("channel_Number=", "" +
channel_number);
                                return;
                        }
                        channel_number =
mySignature.wifiFrequency / 5 - 1000;
                        Log.i("channel_Number=", "" +
channel_number);
                }
                private void getIMEI() {
                        telephonyManager.getDeviceId();
                        mySignature.imei =
telephonyManager.getDeviceId(0);
                         String imeiSIM1 =
telephonyInfo.getImsiSIM1();
                }
                private void getSSID() {
                        mySignature.SSID_WiFi =
wifiInfo.getSSID();
                }
                private void getNetworkID() {
                        mySignature.networkID_WiFi =
wifiInfo.getNetworkId();
                }
                private void getWiFiLinkSpeed() {
                        mySignature.linkSpeed_WiFi =
wifiInfo.getLinkSpeed();
                }
                private void getBSSID() {
                        mySignature.BSSID =
wifiInfo.getBSSID();
                }
                private void getMacAddress() {
                        mySignature.macAddress =
wifiInfo.getMacAddress();
                        try {
                                List<NetworkInterface> all =
Collections.list(NetworkInterface
                                .getNetworkInterfaces());
                                for (NetworkInterface nif : all) {
                                        if
(!nif.getName().equalsIgnoreCase("wlan0"))
                                                continue;
                                        byte[] macBytes =
nif.getHardwareAddress();
                                        if (macBytes == null)
{
                                                // return "";
                                        }
                                        StringBuilder res1 =
new StringBuilder();
                                        for (byte b : macBytes)
{

                res1.append(String.format("%02X:", b));
                                        }
                                        if (res1.length() > 0) {
                        res1.deleteCharAt(res1.length() - 1);
                                        }
                mySignature.macAddress = res1.toString();
                                }
                        } catch (Exception ex) {
                                ex.printStackTrace();
                                Log.i("Exception in mac address
", "");
                                mySignature.macAddress =
"02:00:00:00:00:00";
                        }
                }
                private void getRSSI_WiFi() {
                        mySignature.rssi_WiFi =
```

```
wifiManager.getConnectionInfo().getRssi();
        }

        private void getIPAddress() {
                int ipAddress = wifiInfo.getIpAddress();
                String ip = String.format("%d.%d.%d.%d",
(ipAddress & 0xff), (ipAddress >> 8 & 0xff), (ipAddress >> 16
& 0xff), (ipAddress >> 24 & 0xff));
                mySignature.ipAddress = ip;
        }
        static class CompareSizesByArea implements
Comparator<Size> {
                @Override
                public int compare(Size lhs, Size rhs) {
                        // We cast here to ensure the
multiplications won't overflow
                        return Long.signum((long)
lhs.getWidth() * lhs.getHeight()
                        - (long) rhs.getWidth() * rhs.getHeight());
                        }
        }
        private void getCameraInfo() {
                CameraManager cameraManager =
(CameraManager)
getSystemService(Context.CAMERA_SERVICE);
                long pixelCount = -1;
                long tempPixelCount = -1;
                Size largest = new Size(0, 0);
                try {
                        for (String cameraId :
cameraManager.getCameraIdList()) {
                                CameraCharacteristics
chars = cameraManager
        .getCameraCharacteristics(cameraId);
        StreamConfigurationMap map = chars
        .get(CameraCharacteristics.SCALER_STREAM_CO
NFIGURATION_MAP);
                        if (map == null) {
                                continue;
                        }
                        // For still image
captures, we use the largest available size.
                        largest = Collections.max(
        Arrays.asList(map.getOutputSizes(ImageFormat.JPE
G)), new CompareSizesByArea());
        tempPixelCount = largest.getHeight() *
largest.getWidth();
                if (tempPixelCount > pixelCount) {
                                pixelCount
= tempPixelCount;
                        }
                }
                } catch (CameraAccessException e) {
                        e.printStackTrace();
                }
                Log.e("pixelCount=",
Long.toString(pixelCount));
                Log.e("pixelCount=",
Long.toString(pixelCount));
                Log.e("pixelCount=",
Long.toString(pixelCount));
                Log.e("pixelCount=Largest",
Long.toString(pixelCount));
                megaPixel = (float) pixelCount /
(1024000.0f);
                mySignature.maxResolution =
Float.toString(megaPixel);
                Log.e("megaPixel=",
Float.toString(megaPixel));
        }
        private void getDeviceLocation() {
                // TODO Auto-generated method stub
                mLocationManager = (LocationManager)
getApplicationContext()
        .getSystemService(LOCATION_SERVICE);
                List<String> providers =
mLocationManager.getProviders(true);
                Location bestLocation = null;
                for (String provider : providers) {
                        Location l =
mLocationManager.getLastKnownLocation(provider);
                        if (l == null) {
                                continue;
                        }
                        if (bestLocation == null
                                        ||
l.getAccuracy() < bestLocation.getAccuracy()) {
                                // Found best last
known location: %s", l);
                                bestLocation = l;
                        }
                }
                mySignature.latitude=
bestLocation.getLatitude();
                mySignature.longitude =
bestLocation.getLongitude();
                Geocoder geocoder;
                List<Address> addresses;
                geocoder = new
Geocoder(MainActivity.this, Locale.getDefault());
                try {
                        addresses =
geocoder.getFromLocation(mySignature.latitude,
                mySignature.longitude, 1);
                        if (!addresses.isEmpty()) {
        Address returnedAddress = addresses.get(0);
                                StringBuilder
strReturnedAddress = new StringBuilder("");
for (int i = 0; i < returnedAddress.getMaxAddressLineIndex();
i++) {
        strReturnedAddress
.append(returnedAddress.getAddressLine(i)).append(
                                        " ");
                                }
        Log.e("MyCurrentLoctionAddress",
                "" + strReturnedAddress.toString());
                                cityName =
addresses.get(0).getLocality();
Log.e("City Name=", "" + strReturnedAddress.toString());
                        } else {
Log.e("MyCurrentLoctionAddress", "No Address returned!");
                        }
                        MyLocationChecker
myLocationChecker = new MyLocationChecker();
                        mySignature.isLocationInRadius
= myLocationChecker
.checkLocationWithinRadius(bestLocation, bestLocation);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Fig. 7:Main Acitivity Class

6. COMPARISON RESULTS

Proposed algorithm is also compared with existing OAuth 2.0 and Zero Trust on various metrics like authentication time, accuracy, scalability, security, energy consumption, computational overhead, memory usage, key management complexity, resistance to replay attacks, interoperability and privacy protection. As shown in Table 5 below the proposed algorithm performs well in various metrics.

Table 5: Comparison table using multiple metrics

Metric	Implemented Algorithm	OAuth 2.0	Zero Trust
Average Authentication Time (s)	0.0000123	0.0728	0.1662
Accuracy (%)	100%	80%	100%
Scalability	High (Fast response)	Medium	Medium
Security	Very High (Signature-based)	High (Token-based, vulnerable to token theft)	Very High (Multi-factor, continuous verification)
Energy Consumption (Joules)	Low (~0.5 J)	Medium (~5 J)	High (~15 J)
Computational Overhead	Minimal (Lightweight signature processing)	Moderate (Token encryption & decryption)	High (Multiple encryption & verification steps)
Memory Usage (MB)	Low (~2 MB)	Moderate (~50 MB)	High (~150 MB)
Key Management Complexity	Simple (Predefined signatures)	Moderate (Token expiration, refresh mechanisms)	Complex (Continuous authentication, MFA keys)
Resistance to Replay Attacks	High	High (Tokens are time-bound but vulnerable to theft)	Very High (Continuous verification prevents stale attacks)
Interoperability	High (Custom implementation required due to mobile agents)	High (Widely supported by web & mobile apps)	Medium (Requires strict policy enforcement)
Privacy Protection	High (Device signature stored centrally)	Low (Tokens can be leaked or stolen)	High (Zero trust minimizes access to sensitive data)

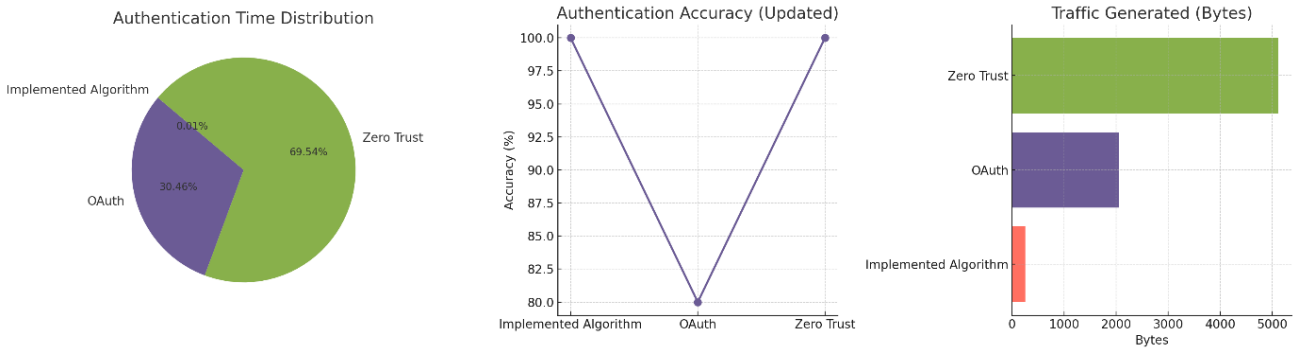

Fig 8: Charts showing Authentication time, accuracy and traffic generated

Fig 8 shows that proportions remain the same, showing that Zero Trust has the highest delay, while Implemented Algorithm is the fastest. OAuth with 80% accuracy, making it more competitive but is still lower than Implemented Algorithm & Zero Trust (100%). No change in traffic, as it is determined by the authentication process structure. Further, one of the advantage of using mobile agent is that it produces less traffic as compared to traditional client server model.Fig 9 shows that implemented algorithm produces very less traffic.

7. CONCLUSION

In this paper device fingerprint and agent based device authentication mechanism has been proposed. Mechanism uses various parameters for device signature calculation. Proposed device signature based authentication mechanism includes device fingerprint generation and comparison with the stored signature. Signature can be used in many ways including location checking of the device with the stored Access Point location. IP address of the device can be checked with the range of IP addresses available in to the network. Static parameters of the signature can be used to produce MD5 Fingerprint of the device which can be matched with the stored MD5 Fingerprint for authentication. Benefit of using MD5 approach is to produce the signature from the static parameters but reverse is not possible i.e. device parameters cannot be generated from the MD5 Fingerprint. Proposed algorithm is also compared with existing popular algorithms and performs well on various parameters

8. FUTURE WORK

AI-Powered Anomaly Detection: Implementing machine learning-based authentication using device signature to detect behavioral anomalies in real time. This can also help specially in intrusion detection. **Blockchain for Decentralized Identity Management:** We can use blockchain to eliminate central authorities, reducing risks of credential theft. Blockchain can also help in creating and maintaining the trust score using consensus algorithm.

9. REFERENCES

[1] U. Kumar and S. Gambhir, "Device fingerprint and

mobile agent-based authentication technique in wireless networks," Int. J. Fut. Gen. Comm. Netw., vol. 11, no. 3, pp. 33–48, 2018.

[2] Statista, "Number of IoT connected devices worldwide," 2024.[Online].Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[3] U. Kumar and S. Gambhir, "Device Fingerprint and Mobile Agent based Authentication Technique in Wireless Networks," *Int. J. Future Gener. Commun. Netw.*, vol. 11, no. 3, pp. 33–48, 2018.

[4] M. El Fissaoui, A. Beni-hssane, and M. Saadi, "Multi-mobile agent itinerary planning-based energy and fault aware data aggregation in wireless sensor networks," EURASIP J. Wireless Commun. Netw., 2018.

[5] X. Wang, M. Chen, T. Kwon, and H. C. Chao, "Multiple mobile agents' itinerary planning in wireless sensor networks: Survey and evaluation," IET Commun., vol. 5, pp. 1769–1776, 2011.

[6] H. Q. Qadori, Z. A. Zulkarnain, Z. M. Hanapi, and S. Subramaniam, "Multi-mobile agent itinerary planning algorithms for data gathering in wireless sensor networks: A review paper," Int. J. Distrib. Sensor Netw., vol. 13, no. 2, pp. 1–13, 2017.

[7] T. Zhao, X. Wang, and S. Mao, "Cross-domain, scalable, and interpretable RF device fingerprinting," in Proc. IEEE INFOCOM, 2024.

[8] P. M. Sánchez et al., "Adversarial attacks and defenses on ML-and hardware-based IoT device fingerprinting and identification," Future Gener. Comput. Syst., vol. 152, pp. 30–42, 2024.

[9] R. Li et al., "DeviceRadar: Online IoT device fingerprinting in ISPs using programmable switches," IEEE/ACM Trans. Netw., 2024.

[10] K. Heid and J. Heider, "Haven't we met before?-Detecting device fingerprinting activity on Android apps," in Proc. Eur. Interdiscip. Cybersecurity Conf., 2024.

[11] Z. Lin et al., "A life cycle management architecture for power equipment based on device fingerprinting," in Proc. IEEE Int. Symp. Auton. Syst. (ISAS), 2024.

[12] S. Abbas et al., "Radio frequency fingerprinting techniques for device identification: A survey," Int. J. Inf. Secur., vol. 23, no. 2, pp. 1389–1427, 2024.

[13] Z. Lv et al., "Research on distributed PV network terminal identification technology based on device fingerprint generation," in Proc. 3rd Int. Conf. Comput. Technol., Inf. Eng., Electron. Mater. (CTIEEM), vol. 12987, SPIE, 2024.

[14] Z. Xi et al., "Device identity recognition based on an adaptive environment for intrinsic security fingerprints," Electronics, vol. 13, no. 3, p. 656, 2024.

[15] D. Zhang et al., "DevPF: Device identification through passive fingerprints in IoT," in Proc. IEEE Int. Symp. Auton. Syst. (ISAS), 2024.

[16] V. Kohli, M. N. Aman, and B. Sikdar, "An intelligent fingerprinting technique for low-power embedded IoT devices," IEEE Trans. Artif. Intell., 2024.

[17] V. Kumar and K. Paul, "Device fingerprinting for cyber-physical systems: A survey," ACM Comput. Surv., vol. 55, no. 14s, pp. 1–41, 2023.

[18] N. Basha et al., "Channel-resilient deep-learning-driven device fingerprinting through multiple data streams," IEEE Open J. Commun. Soc., vol. 4, pp. 118–133, 2023.

[19] H. Cui et al., "Trust assessment for mobile crowdsensing via device fingerprinting," ISA Trans., vol. 141, pp. 93–102, 2023.

[20] Y. Sang et al., "Toward IoT device fingerprinting from proprietary protocol traffic via key-blocks aware approach," Comput. Secur., vol. 131, p. 103145, 2023.

[21] R. R. Chowdhury and P. E. Abas, "A survey on device fingerprinting approach for resource-constraint IoT devices: Comparative study and research challenges," Internet Things, vol. 20, p. 100632, 2022.

[22] S. Aneja et al., "Device fingerprinting using deep convolutional neural networks," Int. J. Commun. Netw. Distrib. Syst., vol. 28, no. 2, pp. 171–198, 2022.

[23] J. V. Monaco, "Device fingerprinting with peripheral timestamps," in Proc. IEEE Symp. Secur. Privacy (SP), 2022.

[24] B. Hamdaoui and A. Elmaghbub, "Deep-learning-based device fingerprinting for increased LoRa-IoT security: Sensitivity to network deployment changes," IEEE Netw., vol. 36, no. 3, pp. 204–210, 2022.

[25] S. Wan et al., "DevTag: A benchmark for fingerprinting IoT devices," IEEE Internet Things J., vol. 10, no. 7, pp. 6388–6399, 2022.

[26] Z. Ren, P. Ren, and T. Zhang, "Deep RF device fingerprinting by semi-supervised learning with meta pseudo time-frequency labels," in Proc. IEEE Wireless Commun. Netw. Conf. (WCNC), 2022.

10. AUTHOR'S PROFILE

Umesh Kumar received his B.Tech. degree in Computer Engineering from DCRUST University, Sonepat, Haryana, India, in 2007. He completed his M.Tech. and Ph.D. degrees in Wireless Security from JC Bose University of Science and Technology, YMCA, Faridabad, Haryana, India, in 2010 and 2020, respectively. From 2010 to 2012, he worked as an Android Application Developer at Tata Consultancy Services. Since 2012, he has been serving as an Assistant Professor in the Department of Computer Engineering at JC Bose University of Science and Technology, YMCA. His research interests include Wireless Security, Blockchain, and Artificial Intelligence