Decentralized and Role-Based Authentication

A.V. Prasad Kambala, N.Ashok

Abstract— The security plays a key role to achieve authentication by making use of different ways and it can be heightened with multiple factors of security. The main aim of this work was to design and implement a pure decentralized authentication system for an Educational Academy Automation Software which uses OpenID and Role Based Authentication(RBA) System as authentication techniques to ensure that only authentic users can access the predefined roles according to their Authorization level. It admits different computing software that can be used to digitally create, manipulate, collect, store, and relay Academy information needed for attaining tasks like staff hiring, people profiles, creating departments and branches along with sections, staff scheduling, student marks and profiles, notifications, repots and various activities. Further the work mainly includes decentralization feature which facilitates the user to choose one of their trusted providers. Transport Layer Security (TLS) protocol has been implemented to provide security and data integrity for communications over networks. TLS encrypts the segments of network connections at the Transport.

Keywords: OpenID Provider: OP. An OpenID Authentication server on which a Relying Party relies for an assertion that the end user controls an Identifier. OP Endpoint: The URL which accepts OpenID Authentication protocol messages, obtained by performing discovery on the User-Supplied Identifier. This value MUST be an absolute HTTP or HTTPS URL.

I. INTRODUCTION

The World Wide Web (WWW) is a critical enabling technology for electronic commerce on the Internet. Its underlying protocol, HTTP (Hypertext Transfer Protocol [Fielding et al. 1999]), has been widely used to synthesize diverse technologies and components, to great effect in Web environments. Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. However, over the years we are using centralized authentication techniques for control accessing over many websites in World Wide Web. When the client visits an application desiring to authenticate to it, the application redirects it to CAS. CAS validates the client's authenticity, usually by checking a username and password against a database (such as Kerberos or Active Directory)[11].

If the authentication succeeds, CAS returns the client to the application, passing along a security ticket. The application then validates the ticket by contacting CAS over a secure connection and providing its own service identifier and the ticket. CAS then gives the application trusted information about whether a particular user has successfully authenticated.

The authentication technique which overcomes disadvantages in centralized authentication [10] is OpenID Authentication [1]. OpenID is a open standard through which users can be authenticated in a decentralized manner. OpenID allows you to use an existing account to sign in to multiple websites, without needing to create new accounts. You may choose to associate information with your OpenID that can be shared with the websites you visit, such as a name or email address. With OpenID, you control how much of that information is shared with the websites you visit. With OpenID, your password is only given to your identity provider, and that provider then confirms your identity to the websites you visit. Other than your provider, no website ever sees your password, so you don’t need to worry about an unscrupulous or insecure website compromising your identity.

The OpenID standard is passionate about identity, privacy, and security on the web. On successful authentication, the Authentication and authorization modules which have been developed for this purpose is called and the role for the user is retrieved. Privileges are granted to roles which in turn roles are granted to users as described below, briefly.

OpenID is an open and decentralized authentication and identity system. OpenID relying parties do not manage end user credentials such as passwords or any other sensitive information which makes authentication and identity management much simpler and secure. OpenID specifications do not discuss how users are authenticated to the OpenID providers. However, this specification defines an authentication mechanism that can be used by web services clients to authenticate themselves to OpenID providers.

OpenID protocol [13] involves into two web browser redirections at authentication: redirection from the relying party to the OpenID provider with an Authentication Request message and a redirection from the OpenID Provider to the relying party with an Authentication Response message. The authentication mechanism defined in this specification, eliminates two redirections mentioned above so that web service clients can directly communicate with OpenID providers and perform authentications.
This specification defines an authentication mechanism for the OpenID protocol that can be used by web service clients to authenticate users to OpenID providers by direct communication in a restful manner. This specification is a supporting specification for the OpenID Token Profile [13] for Web Services specification. OpenID Token Profile provides a mechanism that can be used by web services systems to utilize OpenID protocol decentralized authentication. Existing OpenID protocol contains two browser redirections which are used to authenticate users to and get responses from, the OpenID provider. Therefore the existing OpenID protocol heavily depends on web browser redirection (Indirect Communication) in the process of authentication. Nevertheless in most web services systems getting web browser involvement for communications is inappropriate. Therefore web service systems require a browser free mechanism for OpenID authentication. The authentication mechanism discussed in this specification addresses this issue. So the OpenID has several advantages [5] due to this Decentralization.

1.1 Protocol Overview

1. Web Service client initialize the authentication with OpenID identifier. It performs discovery on the normalized OpenID and establish the OP endpoint URL that client uses for authentication.

2. Web service client sends an Authentication Request [2] request to the end point URL with the password.

3. The OpenID Provider extracts the password from the request and performs user authentication.

4. The OP send back a positive response with required attributes if the authentication was successful, else OP send back negative response.

5. Web service client receives the positive response and creates the request for access web service.

6. Web service receives the request and processes the request and verifies the request with the OP, if verification success web server grants the request to access web service.

Normalization and Discovery:
Normalization and discovery are done according to the OpenID Authentication 2.0 [5] specification. Yadis discovery is used as the discovery protocol.

Requesting Authentication - Direct Communication:
Direct communication is initiated by a web service client to an OP endpoint URL. This specification extends the Authentication Request message such that it carries users password that is used to authenticate to the OpenID provider. This communication MUST be done over HTTPS [9],[3].

1.2 Logging In With OpenID Provision

1. First open the project login page it will displays a login page which consists of a text box to enter OpenID along with a list of openid providers as shown in Figure 1.

![OpenID Login form](image1.png)

Figure 1: OpenID Login form

The above Fig. 1 is a list of OpenID providers. We provide a list of OpenID providers and we must choose our providers (trusted parties) based on our previous existence or interest.

2. After choosing OpenID provider then the server redirects the page to OpenID Server shows Login page which is like as below Fig.2.

![OpenID Provider Login Page](image2.png)

Figure 2: OpenID Provider Login Page

3. Here the OpenID provider checks to see if you are who you say you are. The login page has to full fill with user id and password. Other wise if you already logged in then the page automatically bypasses to next step.

4. Now you’ve proven to your provider that you really are who you say you are. Next, your provider wants to make sure that you want to log into the requesting web site and that you are willing to share information with it.

Most of the web sites, usually just want your provider to authenticate that you own a particular OpenID, but some web sites want to know other things, like your e-mail address, so that they don’t have to bother you to get it. Your provider asks you which of this information you’re willing to give out and which not.
You usually also have the option of giving it out just this once or giving it out automatically whenever the website asks for it. That will look something like in Fig.3:

All you have to do is choose how much information to give and whether to give it just once or whenever the website asks. Now, your provider sends you back to the web site you were visiting and gives it the information you allowed. You are now logged in!

**Figure 3 : Allowing OpenID – Conformation**

### 1.2 Role Based Authentication System:

The Academy Automation software includes Role based Authentication in addition to OpenID authentication. In this software predefined roles are assigned to users of Educational Academy. The roles are created by various authorizations to each task according to TaskIDs. That means the role consists of set of authorizations to various tasks. When the OpenID provider redirects the user to requested website also called relying party (i.e. Educational academy website) as authentic it checks to see the presence of user login information (i.e. OpenID URL) in its database if the user existed it fetches role for particular user and provide access based on user role, if the user not existed in database it return result as invalid user.

The above Fig.4 depicts the Authentication process between Relaying party (Service provider) and OpenID provider. The arrow reveals the direction of communication and the messages are description of the activity occurring over there.

### II. Observation And Problem Description

The Educational Academy Structure consists of some predefined roles. The roles consist of some activities and the activity consists of some tasks. Thus these roles are assigned to some users like principal, head of the department, teaching staff, non teaching staff and students.

The tasks are differentiated between users through assigning roles to users, so that they can be authorized to do only the assigned or authorized tasks for roles. The roles are created like below.

- **Principal**: Staff hiring, Operations on profiles and Operations on Departments can view all the details.
- **Head of the Department**: Syllabus creation, Student attendance, Student Status modifying, Student Promotions, Making Schedule and can view all the details.
- **Teaching staff**: Student marks updating, can view all the details.
- **Non Teaching staff**: Fee details, exam schedules and other related activities.
- **Students**: Students can view his profile and needed information. The user login into this system through the authentication verified and returned by third party. If the verification succeeded then the application simply fetches the corresponding role and allows him to do only the actions that he authorized to do. For example the student cannot allow entering marks.

### 2.1 Problem issues and challenges

There are following problems:

1. The information line must be secured i.e. the line between Relaying party and Provider must be in a secured way.

2. The assignment of roles must be careful, so that there may have no chance of vulnerability.

3. The users credentials must be store in an encrypted[2] form

4. The creation of roles must be careful.

**Figure 4 : The OpenID Authentication Process**
5. After work completion, the server will take responsibility to take care of removal of all user credentials.

6. Control the information access using roles.

7. To differentiate the tasks, each must be have an ID, so that the created roles structure becomes robust.

8. During the selection of OPs (OpenID Providers) list, the phishing attacks may occurred.

III. METHODOLOGY

In Educational Academy Automation the role-id are assigned as follows, if the role is “principal” its role-id is “1”, head of department role-id is “2”. The full description of roles and role-id are given in table I.

<table>
<thead>
<tr>
<th>Role</th>
<th>Role-Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal</td>
<td>10</td>
</tr>
<tr>
<td>Head of department</td>
<td>9</td>
</tr>
<tr>
<td>Teaching staff</td>
<td>8</td>
</tr>
<tr>
<td>Student</td>
<td>7</td>
</tr>
<tr>
<td>Administration non-teaching staff</td>
<td>20</td>
</tr>
<tr>
<td>Department non-teaching staff</td>
<td>19</td>
</tr>
</tbody>
</table>

The Table I shows two columns Role and Role-Ids. The Role is a name and to identify it uniquely we assign a Role-ID. After logging into the Organization with some particular role, the permissions that are assigned to the role are replicated in the user environment providing restrictions to unauthorized operations.

The Table I is a educational academy organization structure. The content depicts the roles as the users (members) and the Ids are the unique identifiers to roles. Here the permissions are like as explained, if the student has the permission “7” and it has a “view” authorization then the staff contains the permissions “8” so it also has the “view” authorizations along with some additional authorizations.

In this scenario each task is identified by a Task-IDs and the roles with Role-IDs respectively. And the Roles (Role-IDs) are assigned to the users and the roles are constructed with set of authorizations of tasks (Task-IDs). So a role depicts a view role as in the database.

3.1 Database Table Structure:

We will have a common login page for all the sections of The Academy Automation. The looks up table of the corresponding IDs are shown in Table I. After login their roles are mapped and corresponding activities are listed according to their authorizations.

IV. ADDING NEW AGE AUTHENTICATION TECHNIQUES AND MECHANISM

4.1 OpenID

It is an authentication system that is based on the premise that anyone can have a URL (or alternatively an Extensible Resource Identifier (XRI) [6] which is allowed in version 2.0[13]) and an OpenID Identity Provider (OP) which is willing to speak on behalf of this URL or XRI. During its short lifetime, OpenID has evolved through three versions, namely Open ID v1.0, v1.1 and v2.0. Whilst the first two versions were only concerned with authentication, v2.0 has added the capability for the exchange of identity attributes as well [7]. The first version of OpenID (v1.0) had several security weaknesses some of which were quickly patched in v1.1 (e.g. messages could be replayed), while others were fixed in v2.0. However, as described below, v2.0 still suffers from several security weaknesses, which may or may not pose a significant risk, depending upon the application that one wishes to secure with OpenID. OpenID works as follows in Figure 1. When a user contacts a Service Provider (SP) that supports OpenID, he presents his URL (or XRI), and the SP contacts the URL to see who is the OP that speaks for it. Once the identity provider has been discovered, the SP must establish a shared secret with it so that future messages can be authenticated, using the well known process of message authentication codes (MAC). The OpenID specifications use Diffie-Hellman to establish the shared secret between the OP and SP; unfortunately, Diffie-Hellman is vulnerable to man in the middle attacks. Once the OP and SP have established a shared secret, the SP redirects the user to the OP, to be authenticated by any mechanism deemed appropriate by the OP. During the authentication process the OP is supposed to check that the user wants to be authenticated to this SP, by displaying the “realm” of the SP to the user. (The realm is a pattern that represents part of the name space of the SP e.g. *.kent.ac.uk). But this realm information can easily be spoofed by an evil SP, which will lull the user into a false sense of security that they are authenticating to a known SP when in fact they will be redirected to an evil one after the authentication has completed. After successful user authentication, the OP redirects the user back to the SP along with an authentication token saying that the user has been authenticated and has control over the OpenID they specified. The SP then grants the user access to its services, as it deems appropriate.
4.2 Advantages of OpenID

4.2.1 Increase registration and conversion rates:

Most websites ask for an extended, repetitive amount of information in order to use their application. OpenID accelerates that process by allowing you to sign in to websites with a single click. Basic profile information (such as your name, birth date and location) can be stored through your OpenID and used to pre-populate registration forms, so you spend more time engaging with a website and less time filling out registration pages.

4.2.2 Access rich user profile data:

Accepting OpenID gives access to a rich set of user data that would otherwise require the completion of lengthy registration forms to obtain. Many OpenID providers collect and share a wide range of demographic information, including name, date of birth, location, gender and an email address. This data allows you to optimize your marketing efforts and tailor your website to better target the needs of your core audience.

4.2.3 Reduce Frustration Associated with Maintaining Multiple Usernames and Passwords:

Most web users struggle to remember the multiple username and password combinations required to sign in to each of their favorite websites, and the password recovery process can be tedious. But using the same password at each of your favorite websites poses a security risk. With OpenID, you can use a single, existing account (from providers like Google, Yahoo, AOL or your own blog) to sign in to thousands of websites without ever needing to create another username and password. OpenID is the safer and easier method to joining new sites.

4.2.4 Gain Greater Control Over Your Online Identity

OpenID is a decentralized standard, meaning it is not controlled by any one website or service provider. You control how much personal information you choose to share with websites that accept OpenIDs, and multiple OpenIDs can be used for different websites or purposes. If your email (Google, Yahoo, AOL), photo stream (Flickr) or blog (Blogger, Wordpress, LiveJournal) serves as your primary online presence, OpenID allows you to use that portable identity across the web.

4.2.5 Minimize Password Security Risks:

Many web users deploy the same password across multiple websites. And since traditional passwords are not centrally administered, if a security compromise occurs at any website you use, a hacker could gain access to your password across multiple sites.

With OpenID, passwords are never shared with any websites, and if a compromise does occur, you can simply change the password for your OpenID, thus immediately preventing a hacker from gaining access to your accounts at any websites you visit. Because the focus of most OpenID providers (such as Google, Yahoo and AOL) is in identity management, they can be more thorough about protecting your online identity. Most website operators are less likely to be as dedicated to protecting your identity as the OpenID providers, whose focus is on securely hosting user identities.

4.2.6 Reduce customer care and password recovery costs:

With OpenID, visitors to your site use an existing portable identity to sign in to your site. Because these users authenticate against an existing identity provider, there is no need to store passwords and invest valuable time and resources into expensive account and password recovery. This frees you to focus on the core functions of your web application and achieve greater customer satisfaction by eliminating frustrations associated with forgotten passwords.

4.2.7 Link your site to the social web:

OpenID is the building block for several other open standards that allow you to enrich the experience for your users and connect your site to the social web. Open source protocols such as Portable Contacts can be used with OpenID to offer your site access to a user’s address book and friends lists. Activity Streams can be implemented on top of OpenID to allow users who authenticate with an OpenID to publish information from your site to their social networks, thereby extending your reach and projecting your brand to the social web.

4.3 OpenID Implementation:

For Educational Automation Academy software, OpenID authentication was implemented[4] by using j2EE[8], .java provided OpenID authentication facility through package called “OpenId.jar”(version jopenid-2.0 )[10]. In this automation software the main home is displayed which consist of set of OpenID providers list and after that selection the login screen displays according to their selection of OpenID providers (come from OpenID servers).

4.4 The Design of OpenID Communication:

File: home.jsp
<Html>
<Body>
<Center>
<form action="/OpenIdServlet" method="post">
<br/>
<br/>
<input type="radio" name="op" value="Google" checked="true"/>
<br/>
<br/>
<input type="radio" name="op" value="Yahoo"/>
<br/>
<br/>
</form>
</Center>
</Body>
</Html>
The above is a part of code of home.jsp which consist of a set of OpenID providers list, from which we have to select one of them to get the authentication.

File: openid-providers.properties
Google = https://www.google.com/accounts/o8/id
Google.alias = ext1
Yahoo.alias = ax

The needed URLs for OpenID servers paths we place a properties file consist of set of OpenID Providers (OP) list and corresponding URLs to establish Endpoints.

4.5 The OpenID Authentication Retrieval:

4.5.1 The classes needed for communication:

Association:
Association between RP and OP, and will be cached in memory for a certain time. Association is something like a secure key shared by your web site and OpenID provider.

Authentication:
It contains Authentication information returned from OpenID Provider.

Base64:
Encodes and decodes the data to and from Base64 notation.

Endpoint:
Endpoint for OpenID Provider, and will be cached in memory for a certain time. The Endpoint URL tells us how to redirect the user to the Google sign on page.

OpenIdManager:
Open ID Manager for all open id operation.

4.5.2 Some Important points to be noted (Assumptions):

- We assume that the MainServlet above has been mapped to /openid in our web application:
- If user requests URL /openid?op=Google, then the MainServlet will redirect user to the Google's sign on page, and store the Mac key of current association into Http Session for later use.
- After sign on, the user will be redirected back to the MainServlet with URL http://your-domain/openid?parameters.....
- To prevent the replay-attack, OpenID standards use a random String called nonce which is different every time when user sign on. And it is our responsibility to check if the nonce is valid.
- The nonce String can be got by request.getParameter("openid.response_nonce") like this:
- If the nonce is valid, the Authentication object can be got from request with a pre-stored Mac key. Use getIdentity () to get the identifier of user. And email can be got by getEmail (), but OP may not returning user's email, which means you may get a null value of email.
- build the redirect URL for end user to sign on, the Endpoint is lookup by the name of OP

Endpoint endpoint = manager.lookupEndpoint("Google");

4.5.3 The methods functionality description:

1. public void init() throws ServletException

This method simply creates a OpenID manager to manages OpenID operations by setting setRealm () and setReturnTo () methods to system on which the domain existed and to the point where the returned Authentication was catches respectively

2. void checkNonce(String nonce)

If the nonce is valid, the Authentication object can be got from request with a pre-stored Mac key. Use getIdentity () to get the identifier of user. And email can be got by getEmail (), but OP may not returning user's email, which means you may get a null value of email.

3. void showAuthentication(PrintWriter pw, Authentication auth)

Displays the obtained Authenticated information from Authentication object which stores the Authenticated information returned by OpenID provider.
4. Protected void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

It first check if “op” equals to null. If then the authentication was obtained otherwise if it is not equals to null then we must obtain the Authentication from a global third party.

4.5.4 Sample code:

```java
import java.io.IOException;
import java.io.PrintWriter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.HashSet;
import java.util.Set;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.expressme.openid.Association;
import org.expressme.openid.Authentication;
import org.expressme.openid.Endpoint;
import org.expressme.openid.OpenIdException;
import org.expressme.openid.OpenIdManager;

/**
 * Sample servlet using JOpenID.
 * @author Michael Liao (askxuefeng@gmail.com)
 */

public class OpenIdServlet extends HttpServlet {

    static final long ONE_HOUR = 3600000L;
    static final long TWO_HOUR = ONE_HOUR * 2L;
    static final String ATTR_MAC = "openid_mac";
    static final String ATTR_ALIAS = "openid_alias";

    private OpenIdManager manager;

    @Override
    // Overriding the init method in super class
    public void init() throws ServletException {
        super.init();
        manager = new OpenIdManager();
        manager.setRealm("http://localhost");
        manager.setReturnTo("http://localhost/openid/OpenIdServlet");
    }

    @Override
    // Overriding the doGet method in superclass
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String op = request.getParameter("op");
        if (op == null) {
            // check sign on result from Google or Yahoo:
            checkNonce(request.getParameter("openid.response_nonce"));
            // get authentication:
            byte[] mac_key = (byte[]) request.getSession().getAttribute(ATTR_MAC);
            String alias = (String) request.getSession().getAttribute(ATTR_ALIAS);
            Authentication authentication = manager.getAuthentication(request, mac_key, alias);
            response.setContentType("text/html; charset=UTF-8");
            showAuthentication(response.getWriter(), authentication);
            return;
        } else if (op.equals("Google") || op.equals("Yahoo")) {
            // redirect to Google or Yahoo sign on page:
            Endpoint endpoint = manager.lookupEndpoint(op);
            Association association = manager.lookupAssociation(endpoint);
            request.getSession().setAttribute(ATTR_MAC, association.getRawMacKey());
            request.getSession().setAttribute(ATTR_ALIAS, endpoint.getAlias());
            String url = manager.getAuthenticationUrl(endpoint, association);
            response.sendRedirect(url);
        } else {
            throw new ServletException("Unsupported OP: "+ op);
        }
    }

    void showAuthentication(PrintWriter pw, Authentication auth) {
        pw.print("<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" /></head><title>Test JOpenID</title></head><body><h1>You have successfully signed on!</h1><p>Identity: "+ auth.getIdentity () + "</p>
        pw.print ("Email: "+ auth.getEmail () + "</p>"
        pw.print("Full name: "+ auth.getFullName() + "</p>"
```
pw.print("<p>First name: " + auth.getFirstname() + "</p>);
pw.print("<p>Last name: " + auth.getLastname() + "</p>);
pw.print("<p>Gender: " + auth.getGender() + "</p>);
pw.print("<p>Language: " + auth.getLanguage() + "</p>);
pw.print("</body></html>");
pw.flush();

/*It is for checking the response nonce for any replay-attack*/
void checkNonce(String nonce) {
    // check response_nonce to prevent replay-attack:
    if (nonce==null || nonce.length()<20)
        throw new OpenIdException("Verify failed.");
    // make sure the time of server is correct:
    long nonceTime = getNonceTime(nonce);
    long diff = Math.abs(System.currentTimeMillis() - nonceTime);
    if (diff > ONE_HOUR)
        throw new OpenIdException("Bad nonce time.");
    if (isNonceExist(nonce))
        throw new OpenIdException("Verify nonce failed.");
    storeNonce(nonce, nonceTime + TWO_HOUR);
}

// simulate a database that store all nonce:
private Set<String> nonceDb = new HashSet<String>();

// check if nonce is exist in database:
boolean isNonceExist(String nonce) {
    return nonceDb.contains(nonce);
}

// store nonce in database:
void storeNonce(String nonce, long expires) {
    nonceDb.add(nonce);
}

/*This method for getting nonce time used by the method checkNonce*/
long getNonceTime(String nonce) {
    try {
        return new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssZ")
            .parse(nonce.substring(0, 19) + "+0000")
            .getTime();
    } catch(ParseException e) {
        throw new OpenIdException("Bad nonce time.");
    }
}

V. CONCLUSION

The research problem and goal of the Academy Automations to design a highly secure and efficient framework based on Service Oriented Architecture. Keeping in mind the policies of minimum data redundancy and efficient security, the work revolved around designing a plug in for secure role based authentication. Presently the authentication is based on the traditional userid-password based approach, but as is suggested in this report, work can be done to incorporate newage technologies such as OpenID. OpenID provides increased flexibility for application deployment by enabling applications to leverage and third-party authentication providers for handling authentication. Providers such as OpenID have become very common as more users want a single user profile across multiple sites for blogs, wikis, and other social networking activities. Additionally, many Web sites do not want to maintain, or require users to continually provide, the same profile-related information just to ensure that the user credentials are valid.

REFERENCES

AUTHORS PROFILE

A V Prasad Kambala holds his B.Tech in Computer Science Engineering & Technology. He is currently pursuing M.Tech in Computer Science from Devineni Venkata Ramana & Dr. Himasekhar MIC College of Technology, Kanchikacherla, Krishna DIST, Andhra Pradesh, INDIA. His research interest includes network security client-server computing and IT based education.

Ashok Nutalapati holds M.Tech in Computer Science from Jawaharlal Nehru Technological University Hyderabad, Andhra Pradesh-INDIA. He is currently working as Assistant Professor in Computer Science and Engineering Department in Devineni Venkata Ramana & Dr. Himasekhar MIC College of Technology, Kanchikacherla, Krishna DIST, Andhra Pradesh, INDIA. His research includes Computer Networks, Analysis of Algorithms, Client-Server Computing and IT based education.