**Security in Ad-Hoc Network**

Since there is no administrative facility and central entity in a mobile ad hoc network, the nodes need to collaboratively support all the network activities. To provide reliable communication service under adversarial environments, securing the basic network operation becomes one of the primary concerns in MANETs. This section describes the main types of misbehavior that can be formed in an ad hoc network.

**Communication Phase:**

Communication in ad hoc networks consists of two phases, route discovery and data transmission. Under adversarial environments, misbehaving nodes could disrupt the route discovery by obstructing the propagation of legitimate route control traffic and adversely influencing the topological knowledge of benign nodes. Misbehaving nodes could also disrupt the data transmission phase. First, they could abide the route discovery to place themselves on utilized paths. Then, they could tamper with the in-transit data in an arbitrary manner and degrade the network operation. The upper layer mechanisms and the current ad hoc network routing protocols cannot deal with malicious disruptions of data transmission.

**Misbehaving Node:**

In the MANETs, different misbehaving nodes form misbehavior with different purposes. There are generally two types of misbehaving nodes in an ad hoc network. They form misbehavior with different capabilities and out of different purposes. Malicious nodes, also called attackers, are capable of discarding or altering control and data packets, preventing route discovery between two nodes, make data packets unable to arrive at their destinations while consuming energy and available bandwidth of the network.

These nodes are controlled by adversaries. They have the potential to cause harm to the entire network and all types of network operation. Selfish nodes, which are part of an ad hoc network, use this network to establish their own communication. But they refuse to spend their power for operations that do not directly benefit them. Selfish nodes can drop data packets or refuse to forward routing control packets for other nodes. Although they are not intended to damage the network, their behavior disturbs the performance and influences the operation of the whole network.

**Security Mechanism:**

Due to its absence of infrastructure, the consequent absence of authorization facilities, and the distributed environment, the traditional security mechanisms (which mostly are depend on encryption and authorization) may not be fully applicable to a MANET. It is difficult to make the mobile nodes establish long-lasting trust relationship with each and every peer they are transiently associated with. This impedes providing cryptographic protection and authentication to all control and data traffic in the network. Moreover, even if this type of services were possible, the associated overhead and delay would pose a challenge in such a dynamic environment. These services cannot be effective against the denial of service (DoS) attack, in which the misbehaving nodes just drop its received data packets. Thus, security schemes specific to the ad hoc environment need to be investigated.

It is a great challenge to secure communication and maintain connectivity in the presence of misbehaving nodes across an unknown and frequently changing multihop wireless network topology as in MANETs. The literature contains a large number of studies on guarding the routing mechanisms against a range of attacks under different assumptions and system requirements in ad hoc networks. Secure routing procedure alone cannot guarantee secure and undisrupted data delivery across the network. Many studies on detecting the misbehavior on data and mitigating the adverse effects to provide secure data transmission in ad hoc networks have been conducted.

**Abstract:**

* An Ad hoc network is a type of temporary computer-to-computer connection.
* In ad hoc mode, you can set up a wireless connection directly to another computer.
* The opponents are trying to manipulate the data.
* The hostile use the situation of multi cast traffic.
* Authentication of source and message integrity is the basic requirement.
* Tiered Authentication is used in multicast protocol.
* Multicast traffic employs the MAC to authenticate the message source.
* Advanced Encryption Standard (AES) is used to cipher the file for the security purpose by the sender.
* Central Authority (CA) sends the ciphered file to the users in the Ad-Hoc network.
* And the users retrieves the file by the AES algorithm and confirms the integrity of the file by MAC.

**Existing :**

* Nodes communicated through the Ad-Hoc network.
* It is possible by the multi cast protocol.
* No trusted certificate is provided because keys are not provided to the nodes.
* Tired authentication is not possible.
* Data is transferred from source to destination with out key.

**Disadvantages:**

* Traffic in multi cast is used by the hostile.
* Security is not guaranteed.
* There is no authentication to trust the data.
* Whether the data may be changed by the opponent.
* Key without data increases the vulnerability.

**Proposed:**

* Multicast protocol is used in the proposed system.
* Tired authentication is possible because of multicast protocol.
* MAC is used for the authentication purpose of data and source integrity.
* key is shared by the nodes for getting the Message Authentication code.

**Advantages:**

* Prevent the transmission of data reading by the adversary.
* Manipulation of data by hostile is prevented.
* Trusted authentication is provided because of MAC .
* Original data from source is received by the destination.

**Data flow:**

Sender

File selection

MAC key generation

Appling MAC

Append

AES key

Encryption

File transfer

CA

File Transfer

Receiver 2

Receiver 1

AES key

AES key

Decryption

Decryption

Getting MAC key

Getting MAC key

Applying MAC

Applying MAC

Integrity Checking

Integrity Checking

**Software Requirements:**

* Os : Windows xp
* Coding language : Java
* IDE : Netbeans
* Database : mysql server

**Hardware Requirements:**

* Hard Disk  : 250 GB.
* Monitor   :   15 VGA Color.
* Mouse    :   Logitech.
* Ram    : 512 MB.

**No of modules:**

* MAC yielding.
* File ciphering using AES and transfer.
* CA deployment and data distribution.
* File retrieve using AES.
* Data integrity checking.

**Modules description:**

1. **MAC Yielding:**
   * Sender selects a file for transmission and creates a Message Authentication Code (MAC) for data integrity.
   * Fetch the Message Authentication Code with its appropriate key to the transmission file.
2. **File ciphering using AES and Transfer:**
   * File is ciphered for the purpose of security using AES algorithm by the user defined key.
   * And the file which is encrypted is transferred to the (CA) Central Authority.
3. **CA Deployment and Data Distribution:**

* The file from sender is received by the Central Authority.
* Central Authority sends the appropriate file to the destinations with out doing any change.

1. **File Retrieve using AES:**

* The encrypted file from the Central Authority is received by the receiver.
* Receiver retrieves the file by using the AES algorithm.
* Receiver must use the correct key for the decryption of the file.

1. **Data Integrity Checking:**
   * Receiver gets the key from the file which is used to generating the MAC.
   * And receiver creates the MAC and compared it to the MAC which is received from the file.
   * If there is no change in the two MAC means then the file is not changed and the integrity is proved.

**Literature Survey:**

1. A survey of key management in ad hoc networks:

Abstract:

The wireless and dynamic nature of mobile ad hoc networks (MANETs) leaves them more vulnerable to security attacks than their wired counterparts. The nodes act both as routers and as communication end points. This makes the network layer more prone to security attacks. A main challenge is to judge whether or not a routing message originates from a trustworthy node. The solution thus far is cryptographically signed messages. The general assumption is that nodes in possession of a valid secret key can be trusted. Consequently, a secure and efficient key-management scheme is crucial. Keys are also required for protection of application data. However, the focus here is on network-layer management information. Whereas key management schemes for the upper layers can assume an already running network service, schemes for the protection of the network layer cannot. Keys are a prerequisite to bootstrap a protected network service. This article surveys the state of the art within key management for ad hoc networks, and analyzes their applicability for network-layer security. The analysis puts some emphasis on their applicability in scenarios such as emergency and rescue operations, as this work was initiated by a study of security in MANETs for emergency and rescue operations.

1. Multi-receiver authentication codes: models, bounds, constructions, and extensions:

Abstract:

Multireceiver authentication codes allow one sender to construct an authenticated message for a group of receivers such that each receiver can verify authenticity of the received message. In this paper, we give a formal definition of multireceiver authentication codes, derive information theoretic and combinatorial lower bounds on their performance and give new efficient and flexible constructions for such codes. Finally we extend the basic model to the case that multiple messages are sent and the case that the sender can be any member of the group.

1. A survey of clustering schemes for mobile ad hoc networks:

Abstract:

Clustering has been found to be an effective means of resource management for MANETs regarding network performance, routing protocol design, Quality of Service (QoS) and network modeling though it has yet to be refined to satisfy all the issues that might be faced by choosing this approach. Scalability is of particular interest to ad hoc network designers and users and is an issue with critical influence on capability and capacity. Where topologies include large numbers of nodes, routing packets will demand a large percentage of the limited wireless bandwidth and this is exaggerated and exacerbated by the mobility feature often resulting in a high frequency of failure regarding wireless links. In this paper we present a comprehensive survey and classification of recently published clustering algorithm, which we classify based on their objectives. We survey different clustering algorithm for MANET's; highlighting the defining clustering, the design goals of clustering algorithms, advantages of clustering for ad hoc networks, challenges facing clustering including cost issues and classifying clustering algorithms as well as discussion on the objectives and features of various clustering schemes presented in a comprehensive survey of the related literature.

1. Trust management in mobile ad hoc networks using a scalable maturity- based model:

Abstract:

In this paper, we propose a human-based model which builds a trust relationship between nodes in an ad hoc network. The trust is based on previous individual experiences and on the recommendations of others. We present the Recommendation Exchange Protocol (REP) which allows nodes to exchange recommendations about their neighbors. Our proposal does not require disseminating the trust information over the entire network. Instead, nodes only need to keep and exchange trust information about nodes within the radio range. Without the need for a global trust knowledge, our proposal scales well for large networks while still reducing the number of exchanged messages and therefore the energy consumption. In addition, we mitigate the effect of colluding attacks composed of liars in the network. A key concept we introduce is the relationship maturity, which allows nodes to improve the efficiency of the proposed model for mobile scenarios. We show the correctness of our model in a single-hop network through simulations. We also extend the analysis to mobile multihop networks, showing the benefits of the maturity relationship concept. We evaluate the impact of malicious nodes that send false recommendations to degrade the efficiency of the trust model. At last, we analyze the performance of the REP protocol and show its scalability. We show that our implementation of REP can significantly reduce the number messages.

1. A survey of multicast routing protocols for mobile ad-hoc networks:

Abstract:

Design of a suitable routing protocol is difficult for mobile ad hoc networks due to its inherent dynamism and frequent topology change. Multicasting is even more complex because it requires transmission of an information to various destinations at approximately same time, if possible. Active research work in this field has resulted in a variety of proposals based on tree or mesh structures. This paper presents a state-of-the-art overview of multicast routing protocols for ad hoc networks. We believe that this survey will be a great source of information for researchers in ad hoc networks.

**SOFTWARE DESCRIPTION**

**Java**

Java is a programming language originally developed by James Gosling at Sun Microsystems (now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere." Java is currently one of the most popular programming languages in use, particularly for client-server web applications.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1995. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Class path.

**Java Platform:**

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets. Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

A major benefit of using byte code is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would. Just-in-Time compilers were introduced from an early stage that compiles byte codes to machine code during runtime.

Just as application servers such as Glass Fish provide lifecycle services to web applications, the Net Beans runtime container provides them to Swing applications. Application servers understand how to compose web modules, EJB modules, and so on, into a single web application, just as the Net Beans runtime container understands how to compose Net Beans modules into a single Swing application.

Modularity offers a solution to "JAR hell" by letting developers organize their code into strictly separated and versioned modules. Only those that have explicitly declared dependencies on each other are able to use code from each other's exposed packages. This strict organization is of particular relevance to large applications developed by engineers in distributed environments, during the development as well as the maintenance of their shared codebase.

End users of the application benefit too because they are able to install modules into their running applications, since modularity makes them pluggable. In short, the Net Beans runtime container is an execution environment that understands what a module is, handles its lifecycle, and enables it to interact with other modules in the same application.

Registration of various objects, files and hints into layer is pretty central to the way Net Beans based applications handle communication between modules. This page summarizes the list of such extension points defined by modules with API. Context menu actions are read from the layer folder Loaders/text/x-ant+ xml/Actions.

Key maps folder contains subfolders for individual key maps ( Emacs, JBuilder, and Net Beans). The name of key map can be localized. Use "System File System. Localizing Bundle" attribute of your folder for this purpose. Individual key map folder contains shadows to actions. Shortcut is mapped to the name of file. Emacs shortcut format is used, multi keys are separated by space chars ("C-X P" means Ctrl+X followed by P). "Current Key map" property of "Key maps" folder contains original (not localized) name of current key map.

This folder contains registration of shortcuts. It’s supported for backward compatibility purpose only. All new shortcuts should be registered in "Key maps/Net Beans" folder. Shortcuts installed INS Shortcuts folder will be added to all key maps, if there is no conflict. It means that if the same shortcut is mapped to different actions in Shortcut folder and current key map folder (like Key map/Net Beans), the Shortcuts folder mapping will be ignored.

\* Database Explorer Layer API in Database Explorer

\* Loaders-text-db schema-Actions in Database Explorer

\* Loaders-text-sql-Actions in Database Explorer

\* Plug-in Registration in Java EE Server Registry

XML layer contract for registration of server plug-in and instances that implement optional capabilities of server plug-in. Plug-in with server-specific deployment descriptor files should declare the full list in XML layer as specified in the document plugin-layer-file.html from the above link.

Source files must be named after the public class they contain, appending the suffix .java, for example, HelloWorldApp.java. It must first be compiled into byte code, using a Java compiler, producing a file named Hello World App. class. Only then can it be executed, or 'launched'. The Java source file may only contain one public class but can contain multiple classes with less than public access and any number of public inner classes.

A class that is not declared public may be stored in any .java file. The compiler will generate a class file for each class defined in the source file. The name of the class file is the name of the class, with .class appended. For class file generation, anonymous classes are treated as if their name were the concatenation of the name of their enclosing class, a $, and an integer.

The keyword public denotes that a method can be called from code in other classes, or that a class may be used by classes outside the class hierarchy. The class hierarchy is related to the name of the directory in which the .java file is located.

The keyword static in front of a method indicates a static method, which is associated only with the class and not with any specific instance of that class. Only static methods can be invoked without a reference to an object. Static methods cannot access any class members that are not also static. The keyword void indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call System. Exit () explicitly.

The method name "main" is not a keyword in the Java language. It is simply the name of the method the Java launcher calls to pass control to the program. Java classes that run in managed environments such as applets and Enterprise JavaBeans do not use or need a main () method. A Java program may contain multiple classes that have main methods, which means that the VM needs to be explicitly told which class to launch from.

The main method must accept an array of String objects. By convention, it is referenced as args although any other legal identifier name can be used. Since Java 5, the main method can also use variable arguments, in the form of public static void main (String... args), allowing the main method to be invoked with an arbitrary number of String arguments. The effect of this alternate declaration is semantically identical (the args parameter is still an array of String objects), but allows an alternative syntax for creating and passing the array.

The Java launcher launches Java by loading a given class (specified on the command line or as an attribute in a JAR) and starting its public static void main(String[]) method. Stand-alone programs must declare this method explicitly. The String [] args parameter is an array of String objects containing any arguments passed to the class. The parameters to main are often passed by means of a command line.

Printing is part of a Java standard library: The System class defines a public static field called out. The out object is an instance of the Print Stream class and provides many methods for printing data to standard out, including println (String) which also appends a new line to the passed string.

**Java a High-level Language:**

A high-level programming language developed by Sun Microsystems. Java was originally called OAK, and was designed for handheld devices and set-top boxes. Oak was unsuccessful so in 1995 Sun changed the name to Java and modified the language to take advantage of the burgeoning World Wide Web.

Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called byte code (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Byte code can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.

Object-oriented software development matured significantly during the past several years. The convergence of object-oriented modeling techniques and notations, the development of object-oriented frameworks and design patterns, and the evolution of object-oriented programming languages have been essential in the progression of this technology.

Object-Oriented Software Development using Java: Principles, Patterns, and Frameworks contain a much applied focus that develops skills in designing software-particularly in writing well-designed, medium-sized object-oriented programs. It provides a broad and coherent coverage of object-oriented technology, including object-oriented modeling using the Unified Modeling Language (UML) object-oriented design using Design Patterns, and object-oriented programming using Java.

**Net Beans**

The **Net Beans Platform** is a reusable framework for simplifying the development of Java Swing desktop applications. The Net Beans IDE bundle for Java SE contains what is needed to start developing Net Beans plug-in and Net Beans Platform based applications; no additional SDK is required.

Applications can install modules dynamically. Any application can include the Update Center module to allow users of the application to download digitally-signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

The platform offers reusable services common to desktop applications, allowing developers to focus on the logic specific to their application. Among the features of the platform are:

* User interface management (e.g. menus and toolbars)
* User settings management
* Storage management (saving and loading any kind of data)
* Window management
* Wizard framework (supports step-by-step dialogs)
* Net Beans Visual Library
* Integrated Development Tools

**J2EE**

A Java EE application or a Java Platform, Enterprise Edition application is any deployable unit of Java EE functionality. This can be a single Java EE module or a group of modules packaged into an EAR file along with a Java EE application deployment descriptor. Java EE applications are typically engineered to be distributed across multiple computing tiers.

Enterprise applications can consist of the following:

* EJB modules (packaged in JAR files);
* Web modules (packaged in WAR files);
* connector modules or resource adapters (packaged in RAR files);
* Session Initiation Protocol (SIP) modules (packaged in SAR files);
* application client modules;
* Additional JAR files containing dependent classes or other components required by the application;

Java Platform, Enterprise Edition or Java EE is Oracle's enterprise java computing platform. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE extends the Java Platform, Standard Edition (Java SE), providing an API for fault-tolerance, object-relational mapping, distributed and multi-tier architectures, and web services. The platform incorporates a design based largely on modular components running on an application server. Software for Java EE is primarily developed in the Java programming language and uses XML for configuration.

**References:**

* R. Safavi-Naini and H. Wang, “Multi-receiver authentication codes :models, bounds, constructions, and extensions,” *Inf.Computation, vol.*151, no. 1–2, pp. 148–172, May 1999.
* A. M. Hegland, E. Winjum, S. F. Mjolsnes, C. Rong, O. Kure, and P. Spilling, “A survey of key management in ad hoc networks,” *IEEE Commun. Surveys & Tutorials, vol. 8, no. 3, pp. 48–66, Dec. 2006.*
* J. Y. Yu and P. H. J. Chong, “A survey of clustering schemes for mobile ad hoc networks,” *IEEE Commun. Surveys &Tutorials, vol.1,no.1,*pp. 31–48, 2005.
* P. B. Velloso, *et al., “Trust management in mobile ad hoc networks* using a scalable maturity-based model,” *IEEE Trans. Network Service Management, vol. 7, no. 3, Sep. 2010.*
* L. Junhai, Y. Danxia, X. Liu, and F. Mingyu, “A survey of multicast routing protocols for mobile ad-hoc networks,” *IEEE Commun. Surveys & Tutorials, vol. 11, no. 1, pp. 78–91, first quarter 2009.*

**Sample coding:**

**1.Mac yielding.java**

package tamfinal;

import java.io.\*;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

public class MACYielding extends javax.swing.JFrame {

public static String filepath,strkey,digestB641;

public MACYielding() {

initComponents();

}

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jLabel2 = new javax.swing.JLabel();

jTextField1 = new javax.swing.JTextField();

jButton1 = new javax.swing.JButton();

jLabel3 = new javax.swing.JLabel();

jTextField2 = new javax.swing.JTextField();

jButton2 = new javax.swing.JButton();

jButton3 = new javax.swing.JButton();

jButton4 = new javax.swing.JButton();

jLabel1 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setMinimumSize(new java.awt.Dimension(600, 600));

getContentPane().setLayout(null);

jLabel2.setFont(new java.awt.Font("Times New Roman", 1, 14));

jLabel2.setForeground(new java.awt.Color(255, 255, 255));

jLabel2.setText("Select a file:");

getContentPane().add(jLabel2);

jLabel2.setBounds(55, 166, 77, 17);

getContentPane().add(jTextField1);

jTextField1.setBounds(132, 163, 244, 20);

jButton1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/browse.png"))); // NOI18N

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

getContentPane().add(jButton1);

jButton1.setBounds(210, 235, 100, 30);

jLabel3.setFont(new java.awt.Font("Times New Roman", 1, 14));

jLabel3.setForeground(new java.awt.Color(255, 255, 255));

jLabel3.setText("MACode is:");

getContentPane().add(jLabel3);

jLabel3.setBounds(55, 330, 72, 17);

getContentPane().add(jTextField2);

jTextField2.setBounds(132, 327, 244, 20);

jButton2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/createmac.png"))); jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

getContentPane().add(jButton2);

jButton2.setBounds(210, 380, 100, 30);

jButton3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/macappend.png"))); jButton3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton3ActionPerformed(evt);

}

});

getContentPane().add(jButton3);

jButton3.setBounds(330, 480, 100, 30);

jButton4.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/next.png"))); // NOI18N

jButton4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton4ActionPerformed(evt);

}

});

getContentPane().add(jButton4);

jButton4.setBounds(470, 480, 100, 30);

jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/img/macyielding.png"))); getContentPane().add(jLabel1);

jLabel1.setBounds(0, 0, 600, 600);

pack();

}// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

JFileChooser fc=new JFileChooser();

int a=fc.showOpenDialog(null);

if(a==JFileChooser.APPROVE\_OPTION)

{

File fileToOpen=fc.getSelectedFile();

jTextField1.setText(filepath=fileToOpen.toString());

}

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

try{

strkey = "NItY4ophQ==";

SecretKeySpec key = new SecretKeySpec(strkey.getBytes("UTF-8"), "HmacMD5");

String s,s1;

Mac mac = Mac.getInstance(key.getAlgorithm());

mac.init(key);

FileInputStream fis = new FileInputStream(filepath);

FileOutputStream fos=new FileOutputStream("d:\\data1.txt");

PrintStream ps=new PrintStream(fos);

DataInputStream dis = new DataInputStream(fis);

BufferedReader br = new BufferedReader(new InputStreamReader(dis));

while ((s = dis.readLine()) != null){

ps.print(s);

}

FileInputStream fis1 = new FileInputStream("d:\\data1.txt");

DataInputStream dis1 = new DataInputStream(fis1);

BufferedReader br1 = new BufferedReader(new InputStreamReader(dis1));

s1 = dis1.readLine();

byte[] utf8 = s1.getBytes("UTF8");

byte[] digest = mac.doFinal(utf8);

digestB641 = new sun.misc.BASE64Encoder().encode(digest);

jTextField2.setText(digestB641);

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

try{

FileWriter fstream = new FileWriter(filepath,true);

BufferedWriter out = new BufferedWriter(fstream);

out.newLine();

out.write(strkey);

out.newLine();

out.write(digestB641);

out.close();

JOptionPane.showMessageDialog(null,"MAC is appended");

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

try{

Fileciphering fc=new Fileciphering();

fc.setVisible(true);

this.dispose();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {public void run() {new MACYielding().setVisible(true);}});

}

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JButton jButton3;

private javax.swing.JButton jButton4;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JTextField jTextField1;

private javax.swing.JTextField jTextField2;

}

**2.File ciphering.java**

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

String s="d:\\aesenc.txt";

File desFile= new File(s);

FileInputStream fis = null;

FileOutputStream fos;

CipherInputStream cis;

byte key[] = (jTextField1.getText()).getBytes();

SecretKeySpec secretKey = new SecretKeySpec(key,"AES");

Cipher encrypt = Cipher.getInstance("AES");

encrypt.init(Cipher.ENCRYPT\_MODE, secretKey);

try {

fis = new FileInputStream(filepath1);

} catch(IOException err) {

System.out.println("Cannot open file!");

System.exit(-1);

}

cis = new CipherInputStream(fis, encrypt);

fos = new FileOutputStream(desFile);

byte[] b = new byte[8];

int i = cis.read(b);

while (i != -1) {

fos.write(b, 0, i);

i = cis.read(b);

}

fos.flush();

fos.close();

cis.close();

fis.close();

JOptionPane.showMessageDialog(null,"File is Encrypted");

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

**3.CA Deployment.java**

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new centralauthority1().setVisible(true);

}

});

try{

ServerSocket welcomeSocket = new ServerSocket(6789);

Socket connectionSocket = welcomeSocket.accept();

BufferedReader inFromClient =new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));

ip1=inFromClient.readLine();

ip2=inFromClient.readLine();

String s="d:\\CA.txt";

File f=new File(s);

FileOutputStream fos=new FileOutputStream(f);

BufferedInputStream d=new BufferedInputStream(connectionSocket.getInputStream());

BufferedOutputStream outStream = new BufferedOutputStream(fos);

byte buffer[] = new byte[1024];

int read;

while((read = d.read(buffer))!=-1)

{

outStream.write(buffer, 0, read);

outStream.flush();

}

welcomeSocket.close();

d.close();

outStream.close();

fos.close();

inFromClient.close();

System.out.println("Files are received and stored ");

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

**4.File retrieve**

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

File desFile=new File(fp2);

FileInputStream fis=null;

FileOutputStream fos;

CipherInputStream cis;

byte key[]=(jTextField2.getText()).getBytes();

SecretKeySpec secretKey=new SecretKeySpec(key,"AES");

Cipher encrypt=Cipher.getInstance("AES");

encrypt.init(Cipher.DECRYPT\_MODE, secretKey);

try{

fis=new FileInputStream(filepath1);

}

catch(IOException e){

System.out.println("Cannot open file");

System.exit(-1);

}

cis=new CipherInputStream(fis, encrypt);

fos=new FileOutputStream(desFile);

byte[] b=new byte[8];

int i=cis.read(b);

while(i!=-1){

fos.write(b, 0, i);

i=cis.read(b);

}

fos.flush();

fos.close();

cis.close();

fis.close();

JOptionPane.showMessageDialog(null, "File is Decrypted");

r1intchek r1ic=new r1intchek();

r1ic.setVisible(true);

this.dispose();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new receiver1().setVisible(true);

}

});

try{

Socket clientSocket=new Socket("192.168.0.19",6790);

byte buffer[]=new byte[1048576];

File f=new File("C:\\file.txt");

FileOutputStream fos=new FileOutputStream(f);

BufferedInputStream bis=new BufferedInputStream(clientSocket .getInputStream());

BufferedOutputStream bos=new BufferedOutputStream(new FileOutputStream(f));

int read;

while((read=bis.read(buffer))!=-1){

bos.write(buffer, 0, read);

}

bos.close();

bis.close();

fos.close();

clientSocket.close();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

**5.Data Integrity Cheking**

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

String fp1,str;

FileReader f1=new FileReader(fp1=fp);

BufferedReader br=new BufferedReader(f1);

while((str=br.readLine())!=null){

macinfile=str;

}

RandomAccessFile raf=new RandomAccessFile(fp1, "rw");

jTextField1.setText(macinfile);

int a=macinfile.length();

long l=Long.parseLong(String.valueOf(a));

long l1=raf.length()-l-2;

raf.setLength(l1);

f1.close();

br.close();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

String fp1,str;

FileReader f1=new FileReader(fp1=fp);

BufferedReader br=new BufferedReader(f1);

while((str=br.readLine())!=null){

mackey=str;

}

RandomAccessFile raf=new RandomAccessFile(fp1, "rw");

jTextField2.setText(mackey);

int a=mackey.length();

long l=Long.parseLong(String.valueOf(a));

long l1=raf.length()-l-2;

raf.setLength(l1);

f1.close();

br.close();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

try{

SecretKeySpec key=new SecretKeySpec(mackey.getBytes("UTF-8"), "HmacMD5");

String s,s1;

Mac mac = Mac.getInstance(key.getAlgorithm());

mac.init(key);

FileInputStream fis = new FileInputStream(fp);

FileOutputStream fos = new FileOutputStream("c:\\data1.txt");

PrintStream ps = new PrintStream(fos);

DataInputStream dis = new DataInputStream(fis);

BufferedReader br = new BufferedReader(new InputStreamReader(dis));

while((s = dis.readLine())!=null){

ps.print(s);

}

FileInputStream fis1 = new FileInputStream("c:\\data1.txt");

DataInputStream dis1 = new DataInputStream(fis1);

BufferedReader br1 = new BufferedReader(new InputStreamReader(dis1));

s1 = dis1.readLine();

byte[] utf8 = s1.getBytes("UTF8");

byte[] digest = mac.doFinal(utf8);

String digestB641 = new sun.misc.BASE64Encoder().encode(digest);

if(digestB641.equals(macinfile)){

jTextField3.setText("Not changed");

JOptionPane.showMessageDialog(null, "File is not changed");

this.dispose();

}

else{

jTextField3.setText("Changed");

JOptionPane.showMessageDialog(null, "File is changed");

this.dispose();

}

}

catch(Exception e){

System.out.println(e.getMessage());

}

}