Chapter 1

INTRODUCTION

1.1. Description

Mobile phones store regularly private, sensitive and classified information and hence it is very important that we can confidently identify their users. To gain secure and efficient access either user must change his password frequently or the user should use strong password i.e. combination of alphabets, numeric and special symbols. In order to solve the above problems Keystroke Dynamics is applied. Keystroke Dynamics is a behavioral biometric approach to enhance the computer access rights. Keystroke biometric is based on the assumption that the typing pattern of each user is unique. It is a class of behavioral biometrics that captures the typing style of a user. Typing style includes factors such as amount of time it takes to type the password, dwell time - how long we press a key, flight time - how long we take to type successive keys, finger pressure and amount of space occupied on finger touch. By collecting all these data we can develop a model of how the person types these credentials. In addition to this static information, one can investigate how a person's typing style evolves with continued practice. Neural networks can be used to extract user characteristics as well as validate the user. Neural networks are very efficient as they quickly adapt to changing environments and learn easily. Neural networks can work with large number of features without much complexity. Because of these properties accurate results can be obtained with neural networks.

1.2. Problem Formulation

The traditional method used for securing the private, sensitive and classified information is password authentication systems. However, this password can be easily remembered or hacked. If a password is compromised, it is no longer adequate for authenticating its rightful

owner. The deficiencies of traditional password-based access systems have become more acute as these systems have grown in size and scope. Increase in number of software and devices for hacking and cracking causes gains in unauthorized access which results in exploitation of important data. Methods like authentication using password or lock patterns which is mostly used as security is now not reliable and secure due to rapidly increase in hacking of these passwords. In order to provide a second level of security, password authentication using keystroke dynamics can be applied. This verifies the user based on its typing patterns and does not allow intruders to access the system.

1.3. Motivation

Common models for automating the process of keystroke dynamics are usually done by using statistical based methods. These models include nearest neighbor, Manhattan distance method, Support Vector Machines, Mahalanobis distance method, Euclidean distance method etc. Although these algorithms are commonly used to evaluate the parameters like key hold time, diagraph and latency between two or more keys, its efficiency is limited since its equal error rate is quite high.

Another approach for user authentication using keystroke dynamics would be the use of machine learning techniques. These techniques are widely used in pattern recognition. The core idea is to identify and classify pattern and make correct decision based on data provided. However, it takes a lot of memory to store all the test cases and may get stuck at local minima, leading to high variance in results.

Neural networks help to solve all these problems. Some widely used neural networks are back propagation, radial basis function network, learning vector quantization, multilayer perceptron and self-organizing map. Neural network is claimed to be capable of producing better result than the statistical methods. The classifiers require not only genuine keystroke patterns but also intruders to train the network. It has low average error rate after the detection mode and the error rate of rejecting legitimate users is dropped to zero after the PIN verification mode, thus proving most efficient and reliable of all.

1.4. Proposed Solution

Neural networks have long been able to solve problems which are not solvable by traditional methods. The advantage of using neural network is that they are flexible throughout different types of datasets. Also, large number of datasets can be considered simultaneously. This data-driven approach is also relatively faster as compared to the statistical approaches. In other words, neural network can be trained specifically for a valid user.

Existing statistical models for keystroke dynamics are difficult to be converted into comprehensible forms especially when the number of features increases. Neural networks on the other hand can support multiple features which in the end can still preserve the comprehensibility.

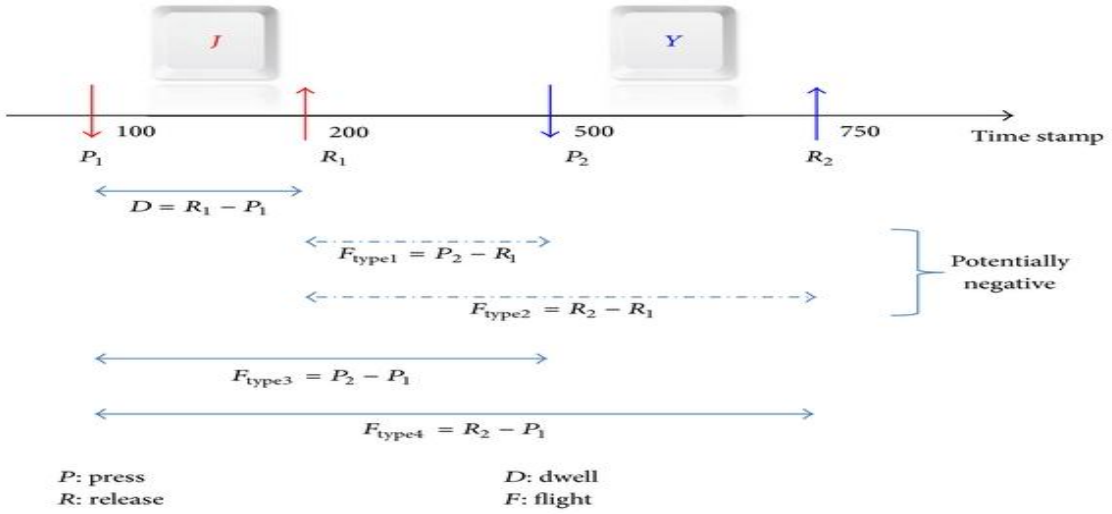


Figure 1.4 1 Keystroke Dynamics

The proposed solution measures keystroke in following ways:

Dwell time (DT): how long a key is pressed

Flight time (FT): how long it takes to move from one key to another

Digraph: Time elapsed between releasing the key and pressing the next key

Finger touch: Amount of space occupied by finger touch.

Pressure: Finger pressure applied while typing

The proposed system operates in 2 phases:

Training phase: In this phase, the extracted features are fed into the neural network and are

trained on this profile. The sample data is iteratively fed into the network based on the current state of its initial pre determined weights. This output is compared with true output and an error value is computed. This value is then propagated backwards through the network so that the weights can be re calculated to reduce the error value and is reiterated until the overall error value falls below a predefined threshold.

Testing phase: In this phase, the features are extracted when the user types the password and the neural network differentiates between legitimate users and imposter. If the detector raises an alarm during this mode, the system does not allow access to that user.

1.5. Scope of the project

The scope of the system includes developing an application that can detect the user not only based on the password but also based on the typing biometrics by identifying the typing patterns. The scope includes the following implementations.

To develop data collector module which stores and accesses the data from DBMS .

To develop password check module which can check the password entered

To develop key stroke recognition module which calculate time between keystrokes

To develop pressure analyzer module that analyses the applied finger pressure

To develop pattern matching module which checks the patterns which matches the existing patterns with the current one for authentication.

To develop decision module that grants permission.

Chapter 2

REVIEW OF LITERATURE

2.1. Statistical Algorithm:

This method adapts the model by retraining the statistical classification algorithm which generates a user model by extracting a set of statistics from the training examples [1]. This algorithm computes some statistics from the training examples (mean, median and standard deviation) for dwell and flight values for specified phrase in whole database. These statistical values represent the user model. Afterwards, in the matching phase, this algorithm verifies each attribute of a new example to check if it meets required conditions. For each attribute, which satisfies required conditions, the algorithm updates the score. The classification of a new example is defined by comparing score to a threshold value. If score is larger than the threshold, the example is classified as positive (legitimate user), otherwise, as negative (intruder) [1].

Some methods like Double Parallel algorithm is used to retrain classification algorithms to adapt the user model [1]. Any example recognized as positive (legitimate user) that reached a score value above an update threshold is added to a set of examples stored in the window. The classification algorithm is then retrained using the window as an updated training set. After training, the system is tested against trained set. If the accuracy of the obtained results is not satisfying then the network may be trained again using the training data.

2.2. Fuzzy Logic:

Fuzzy classifiers can provide acceptable accuracies on diffused datasets because they assign a given data point a degree of membership to all available classes. The primary task of fuzzy classification is to determine the boundaries of the decision regions based on the training data points. Once the class - labeled decision regions in the feature space are determined, classification of an unknown point is achieved by simply identifying the region in which the unknown point resides. Since fuzzy logic assigns each data point a degree of membership to

different decision regions instead of a single association to a decision region, an accurate and efficient learning mechanism for the diffused mobile phone feature-set is achieved [2].

2.3. Neural Networks:

Neural network is a technique that mimics the biological neurons for information processing [3]. Neural network is capable of providing an estimation of the parameters without precise knowledge of all contributing variables. A classical neural network structure consists of an input layer, output layer, and at least one hidden layer [3]. Sample data is iteratively fed into the network to produce some outputs based on the current state of its initial predetermined weights. These outputs are compared to the true output, and an error value is computed. This value is then propagated backwards through the network so that the weights can be recalculated at each hidden layer to reduce the error value. The sequence is reiterated until the overall error value falls below a predefined threshold. Neural network is capable of producing better result than the statistical methods [3].

Chapter 3

SYSTEM ANALYSIS

3.1. Functional Requirements

3.1.1. Importing Training Data

The system initially trains itself from the data collected from the user during the registration process. The neural network learns using the back propagation algorithm. The features are given in the training data set, based on which validity of the user is identified. After training, the accuracy of system is tested.

3.1.2. Generating Template and storing in Database

Keystroke patterns such as dwell time, flight time, latency time between keys, finger pressure, space occupied by finger touch are extracted and the neural network is trained by adjusting its weight vectors. Based on this a template is prepared and encrypted. This encrypted template is then stored in database and used for further authentication.

3.1.3. Authenticating the user

During the authentication phase, keystroke patterns of the user are fed again to the neural network which further classifies the user as legitimate or fake.

3.2. Non-Functional Requirements

3.2.1. Performance Requirements

The performance of the system completely depends upon how fast and accurately the algorithm extracts the features and stores in the database. It also depends on how accurately the system differentiates between the legitimate user and an imposter and also the amount of time required with minimum error to authenticate the user.

3.2.2. Safety Requirements

The safety requirements include not allowing any unauthorized user to access the system's database where all the detailed information about user's credential is stored.

3.2.3. Security Requirements

The system should be able to differentiate between unauthorized and valid user accurately and should immediately take preventive measures if an invalid user tries to tamper the system.

3.2.4. Software Quality Attributes

3.2.4.1. Maintainability

It is necessary to maintain the software, the one who understands how the software or the system works would be able to understand as well as maintain the code. Along with this, the maintenance of the data is also important so that the important details about keystroke features are not lost and while authentication there are no discrepancies found.

3.2.4.2. Robustness

The code must be robust. The transition from one phase to another must be smooth and fast without any difficulties.

3.3. User Requirements

3.3.1. User Interface

A user interface is the one which is viewed by the user onto the computer. After installing the software, the user will first enrol itself in order to store its identity. After this the neural network would process these and will use this information to classify the user as legitimate or fake.

3.3.2. Graphical User Interface

GUI is the representation of the way a particular page would look and would be visible to the user. It is very important to have a very good GUI as it would impress the user and force him to make use of it such that it looks good on his machine. The user would be happy to see the ease of use of application, an intuitive GUI and the way it would be presented to the user would make him feel to use it even more.

3.3.3. Application Programming interface

This interface basically describes how user and the system are well connected with each other. It also describes the dynamic state of the system of how it is going to work. It also tells about the flow of the system from one phase to another which tells us that how the system validates a user from the extracted features. It also shows how the network re-learns itself.

3.3.4. Hardware Interface

These are the interface that tells us the hardware the authors are using to make it user friendly for the users. It tells basically what are the hardware requirements necessary for this system to work successfully i.e. to calculate features and extract keywords. Database is required for the system because keystroke patterns are to be stored and compared to authenticate the user. The neural network will also continue to learn even after the training is complete. Hence, the learning component will also play an important role.

3.3.5. Communication Interface

This will explain how the communication is done between the user and the system. It tells us how the machine will respond to the users based on the extracted keystroke features.

3.4. Specific Requirements

3.4.1. Software Requirement

Android Software Development Kit

SQLite Database

3.4.2. Hardware Requirement:

Android device

RAM: Minimum 1GB

Basic Input and Output Devices: Android Keyboard, Android screen

3.5. Use-Case Diagrams and description

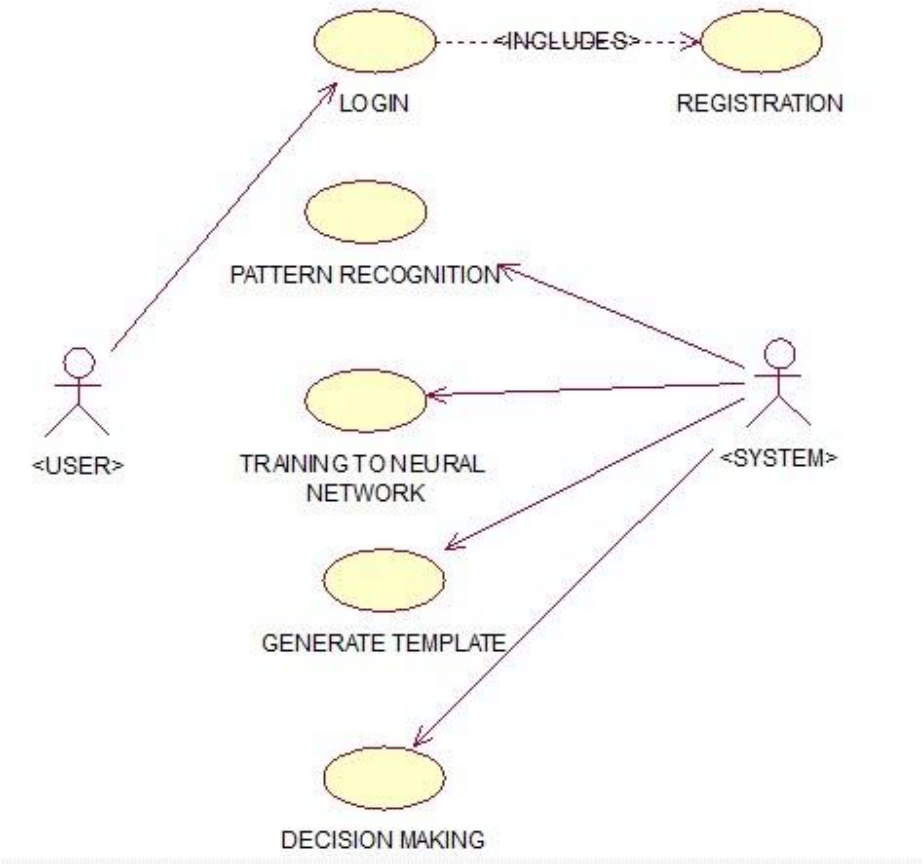


Figure 3.5.1 Use case Diagram

Description

The system extracts the behavioral data from the user after the user enters the password. These features are fed to the neural network and based on these features network authenticates the user. The network compares the extracted features with that of the template and uses it for authenticating the user.

Chapter 4

ANALYSIS MODELING

4.1. E.R Diagram

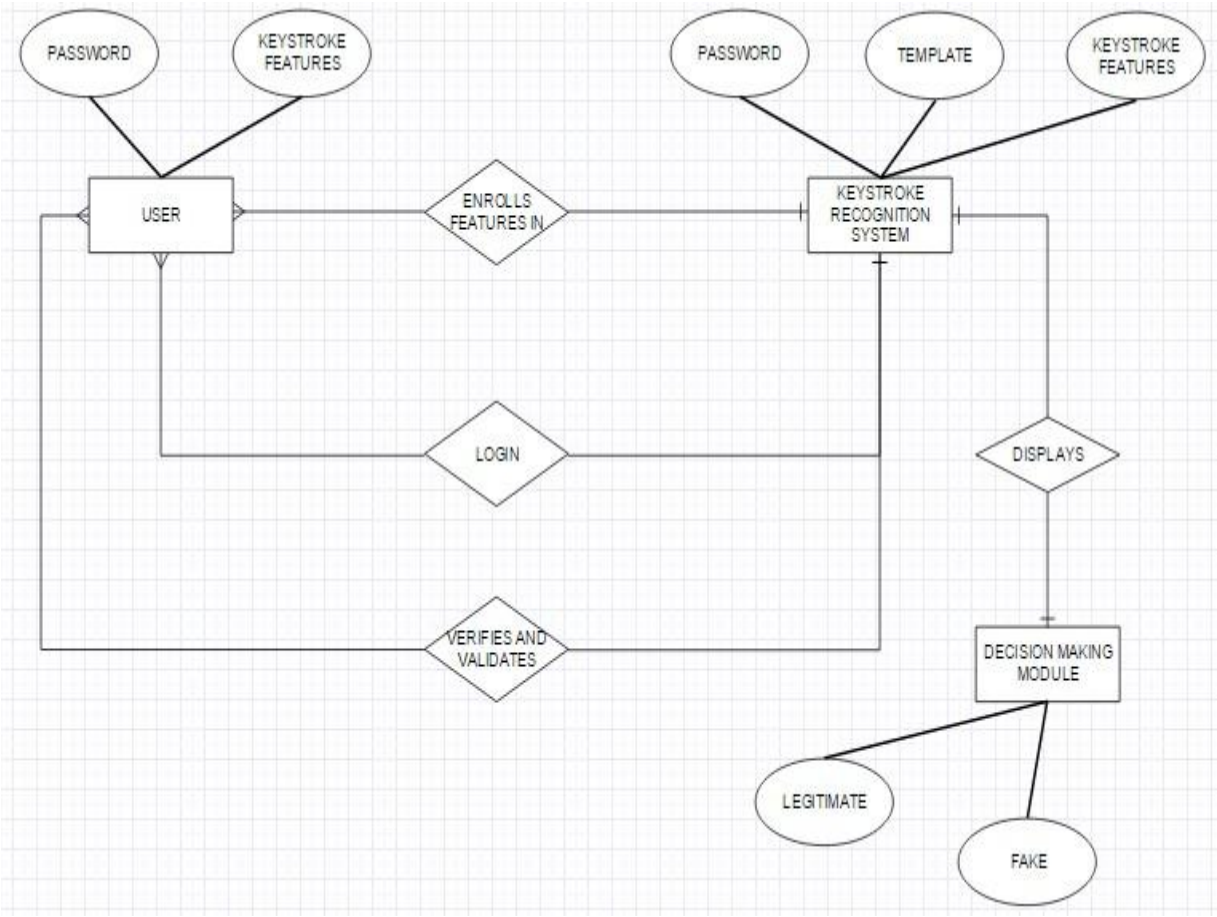


Figure 4.1. 1 E-R Diagram

4.2. Activity Diagrams / Class Diagrams

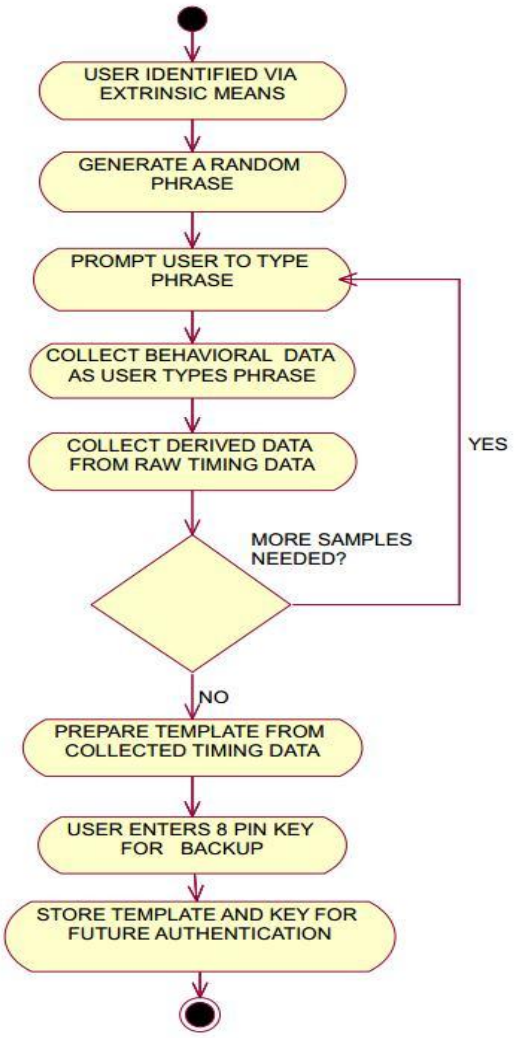


Figure 4.2. 1 Training phase

Description

In the enrolment phase, the user starts the training of the network by entering in to the system. The system randomly generates few phrases and prompts the user to type them. The system keeps a track of user's typing patterns and records them. This is given as input to the neural network which learns using the back propagation method and produces rules to classify a user as legitimate or fake.

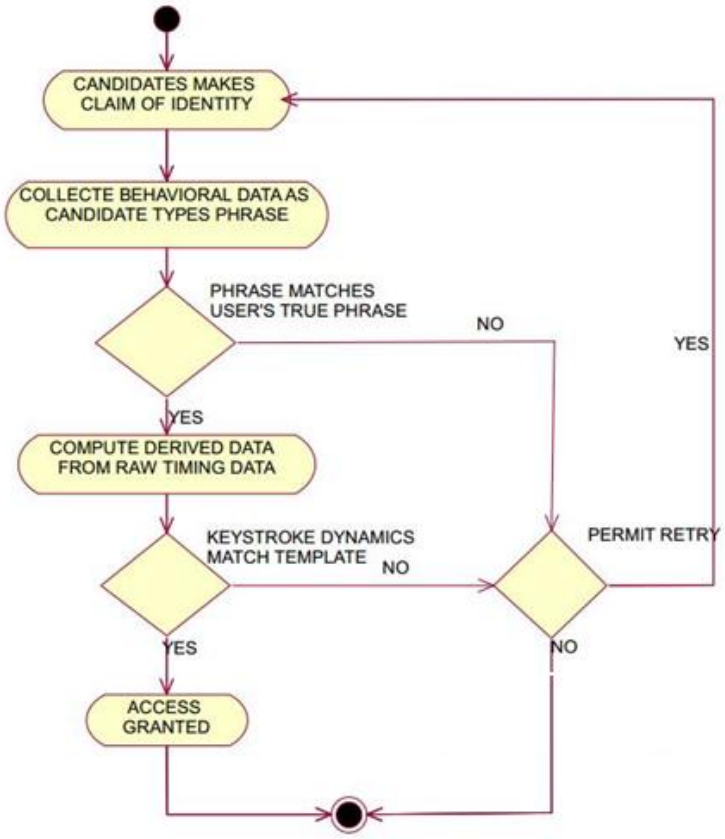


Figure 4.2. 2 Testing Phase

Description

Whenever a user tries to log in to the system, the system collects the timing patterns of the user. The network then authenticates the user based on these characteristics. If a user is authenticated successfully, the system provides all accesses to the user.

4.3. Functional Modeling

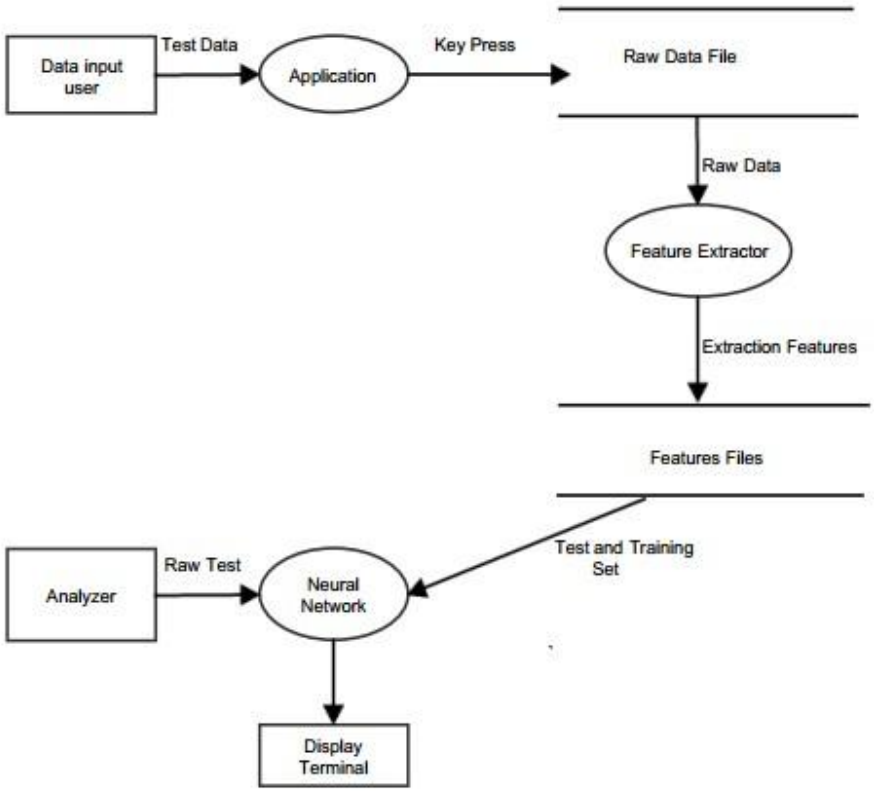
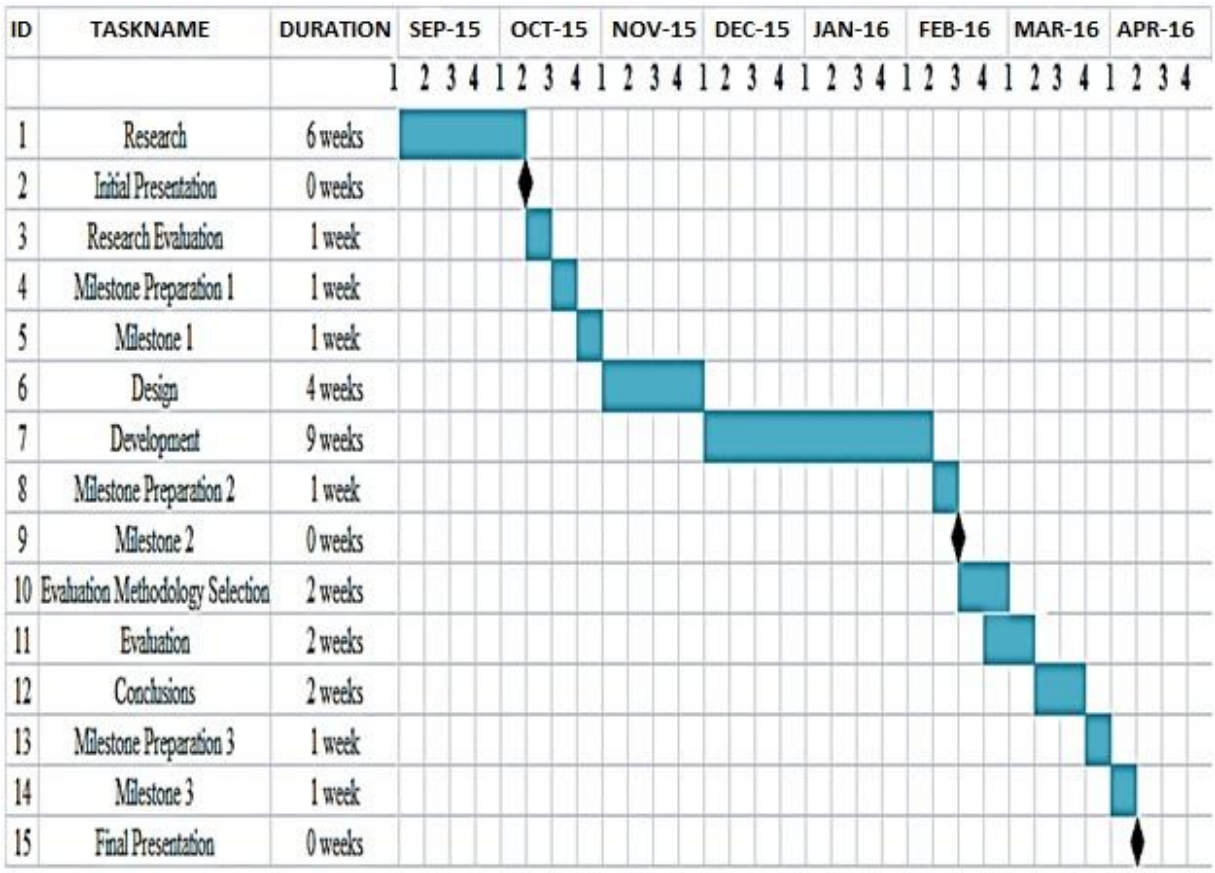


Figure 4.3. 1 Data Flow Diagram

Description:

Initially, the user enters the password using his normal keystroke patterns, the feature extractor then extracts the keystroke data from the raw data. These extracted features are then fed to the neural network which with the help of the trained data, tests this data and classifies the user as legitimate or fake.

4.4. TimeLine Chart



Milestones	Description
Milestone 1	Initial Feasibility tests, planning and research completion.
Milestone 2	Completion of Designing and Coding part.
Milestone 3	Final modifications and completion of Project.

Figure 4.4. 1 TimeLine Chart

Chapter 5

DESIGN

5.1 Architectural Design

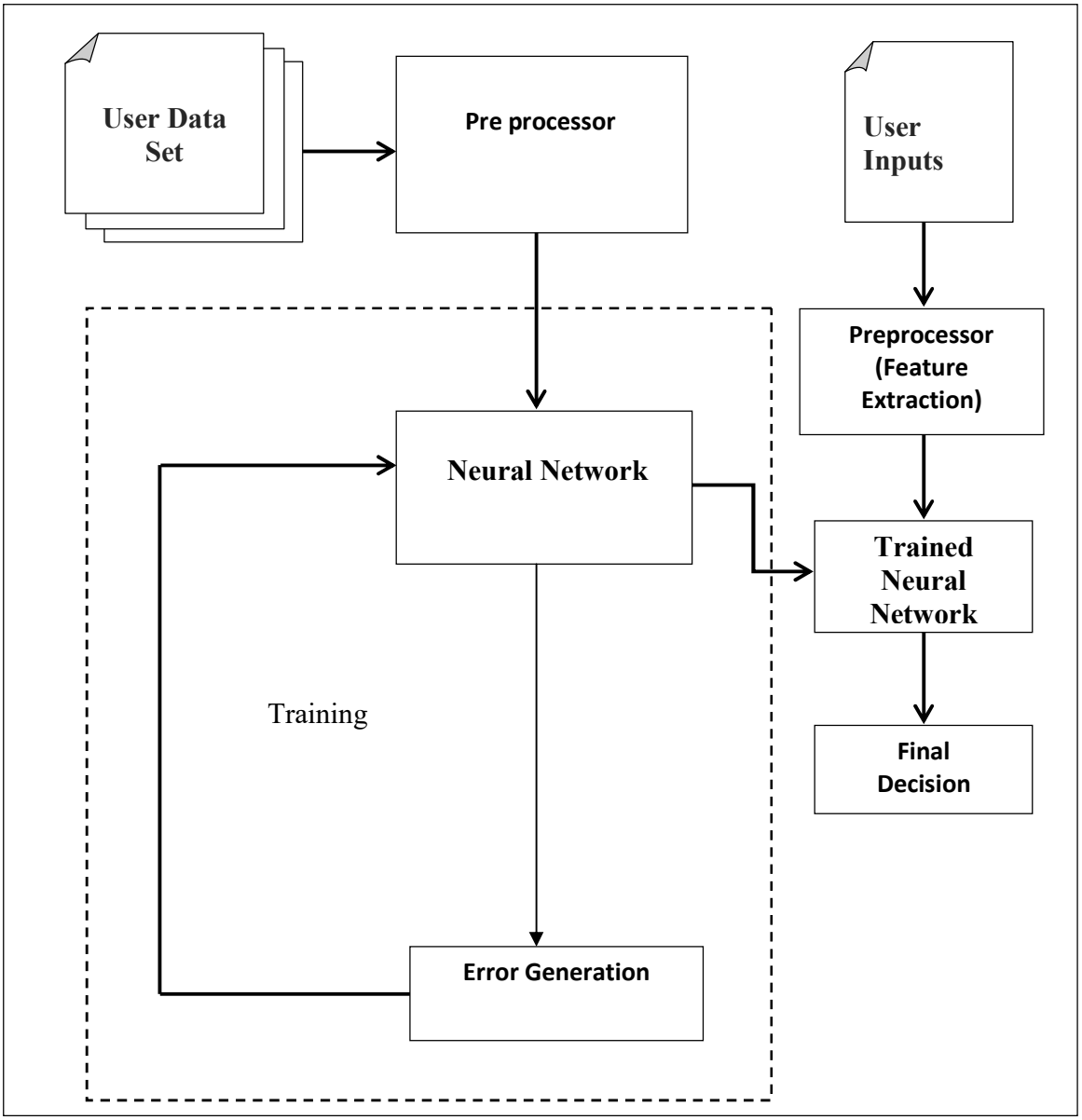


Figure 5.1. 1 Architectural Design

Architectural Flow

The entire architecture has several components which can be essentially grouped into two modules. The first module is the learning module which is responsible for training of the algorithm used, i.e. Error Back Propagation Algorithm. The learning module is active only during development phase. The second module is the validation module which is responsible for predicting the validity of the user. This module is active only after the network is trained successfully and is active thereafter.

The learning and validation modules both consist of a Pre processor which extracts the timing characteristics of the user and normalizes them. These features include dwell time, flight time, overall time to type the password, finger pressure, size of finger touch, digraph time, trigraph time. These extracted features are consolidated and fed to the Feed Forward Network for training. The Error Generation module lies immediately below the Feed Forward Network which generates the error between actual output and desired output, and propagates this error.

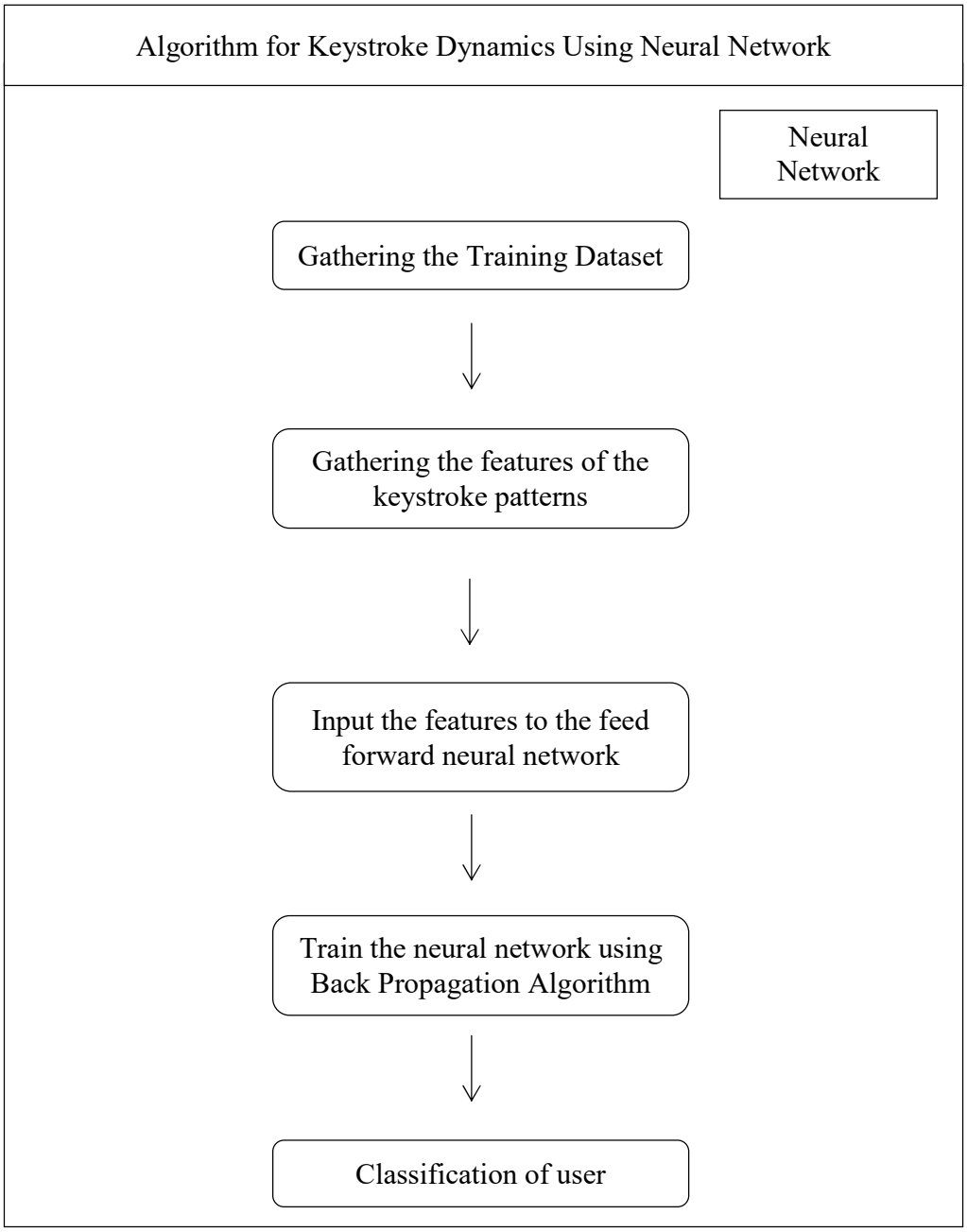


Figure 5.1. 2 Architectural Flow: Training

During the training phase, the training dataset is fed as an input to the Feed Forward Network along with the desired output values. The Error generator module generates the error and propagates the error backwards to adjust the weight vectors. After the complete training phase, the weights are stabilized as the network gets fully trained.

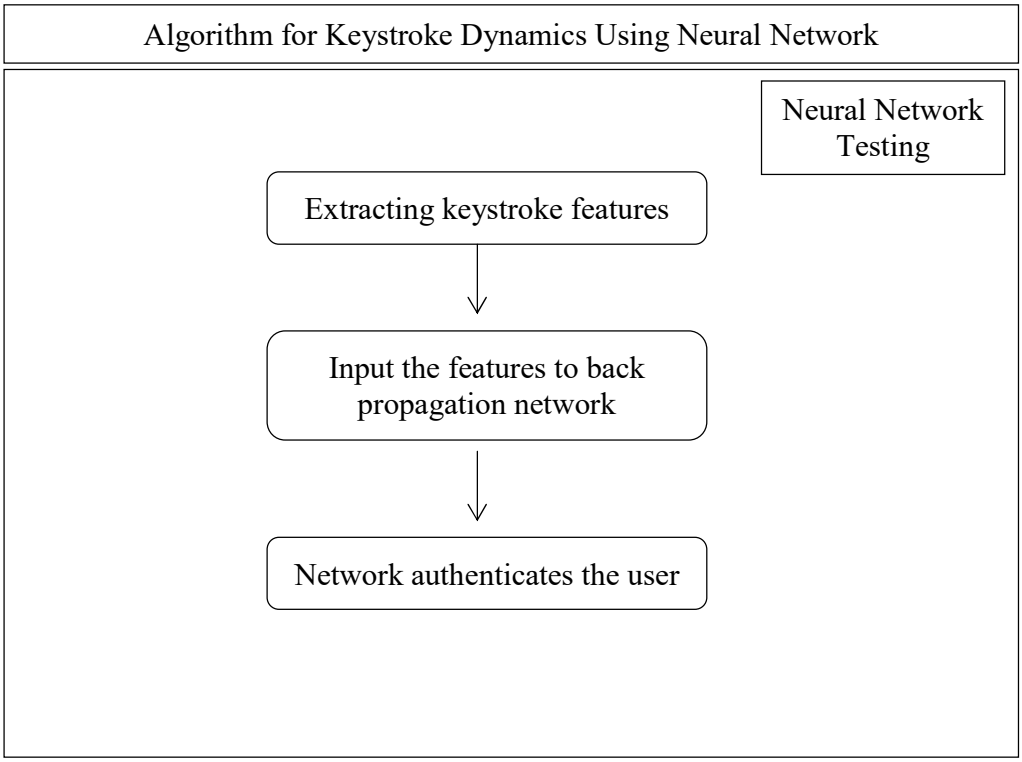


Figure 5.1. 3 Architectural Flow: Testing

During the testing phase, the inputs from the user are fed to the network. The network according to its stabilized weights and inputs from user, classifies the user as legitimate or fake. The entire testing process is effectively summarized in Fig 5.1.3. On each successful attempt, the network is again trained with that set of values, thus adapting to changing typing patterns throughout its lifecycle.

5.2. User Interface Design

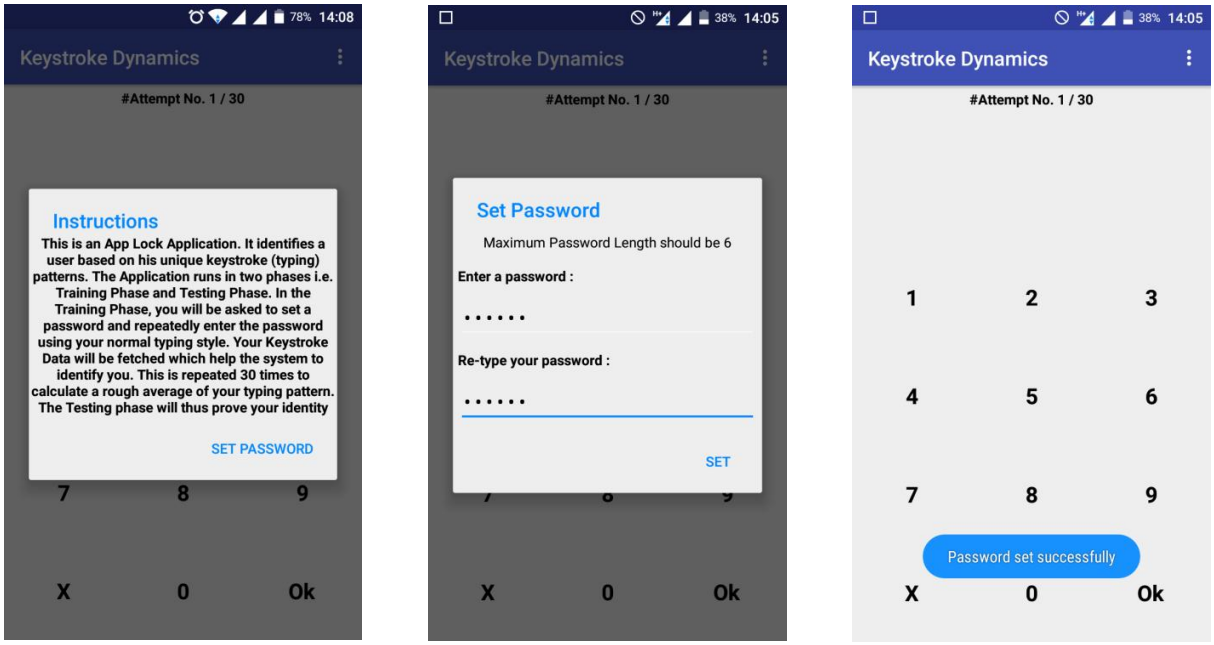


Figure 5.2. 1 User Interface of Password Module

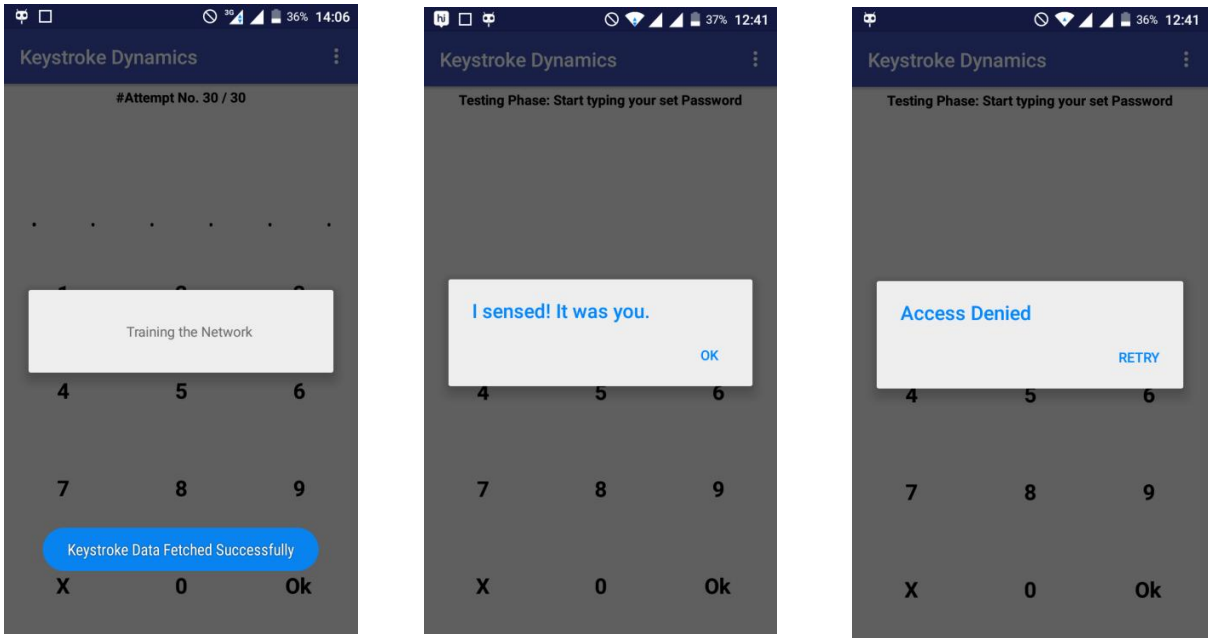


Figure 5.2. 2 User Interface of Training and Testing Module

Chapter 6

IMPLEMENTATION

6.1. Algorithm/Method Used

Error Back Propagation Algorithm(EBPTA)

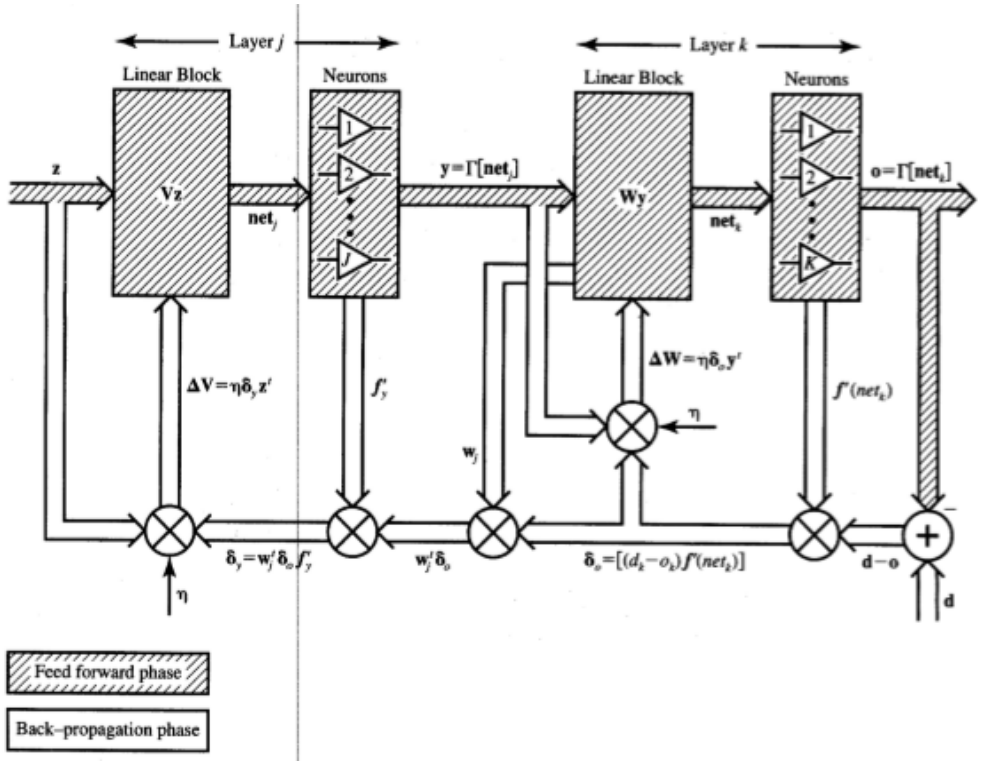


Fig. 6.1.1 Error back-propagation training

In our project we have used error back propagation training algorithm (EBPTA), which is applied to feed forward networks consisting of processing elements with continuous differentiable activation function. For a given set of training input-output pair, this algorithm provides a procedure for changing the weights in a BPN to classify the given input patterns correctly. The basic concept for this weight update algorithm is simply the gradient-descent method. This is the method where the error is propagated back to the hidden unit. The aim of the neural network is to train the network to achieve a balance between network's ability to respond

and its ability to give reasonable responses to the input that is similar but not identical to the one that is used in training. To update weights the error that is the difference between desired and actual output must be calculated at the output layer. Thus, the training of the BPN is basically done in three stages- the feed forward of the input training pattern, the calculation and back propagation of the error and updating of weights. The testing of the BPN involves computation of feed forward phase only.

The error back propagation learning algorithm can be outlined in following steps:

Step 1: Initialize weights and learning rate.

Feed-forward phase (Phase I):

Step 2: Each input unit receives input signal x_i and sends it to the hidden unit (i = 1 to n).

Step 3: Each hidden unit z_j (j = 1 to p) sums its weighted input signals to calculate net input:

$$z_{in} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_1 (binary or bipolar sigmoidal activation function):

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 4: For each output unit y_k (k = 1 to in), calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back propagation of error (Phase II):

Step 5: Each output unit y_k(k = 1 to m) receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j$$

Also, send δ_k to the hidden layer backwards.

Step 6: Each hidden unit (z_j, j = 1 to p) sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term gets multiplied with the derivative off to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

On the basis of the calculated, update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i$$

Weight and bias updation (Phase III):

Step 7: Each output unit (y_k, k = 1 to m) updates the bias and weights:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Each hidden unit (z_j, j = 1 to p) updates its bias and weights:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

Step 8: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

6.2. Feature Extraction

For the feature extraction section, we have used the inbuilt pressure analyzer component in android devices to identify the pressure, the touch screen sensor to identify the co-ordinates of touch and size of finger touch, milliseconds precision timer module to accurately identify different timing characteristics. These extracted features are further normalized using unit-based normalization technique in order to bring them in the range of [0, 1]. The code snippet for feature extraction and normalization is given below,

Feature Extraction code for Button 0,

```
t0 = (TextView)findViewById(R.id.t0);
t0.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
        checkPassword("0");
        normalizeData(0);
        }
        });

t0.setOnTouchListener(new View.OnTouchListener() {

        @Override
    public boolean onTouch(View v, MotionEvent event) {
        t0_x = (double) event.getRawX();
        t0_y = (double) event.getRawY();
        t_size[current_pointer] =  (double) event.getSize();
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
        lastDown = System.currentTimeMillis();
        starttime[current_pointer] = lastDown;
        t_pressure[current_pointer] = ((double) event.getPressure() / 1000);
        if (current_pointer != 0) {
        difference[current_pointer - 1] = ((starttime[current_pointer] -
        starttime[current_pointer - 1] - presstime[current_pointer - 1] * 1000) / 1000);
```

```
                }
                } else if(event.getAction() == MotionEvent.ACTION_UP) {
                t0_keyPressedDuration = ((System.currentTimeMillis() - lastDown) / 1000);
                presstime[current_pointer] =  t0_keyPressedDuration;
                if (current_pointer != 0) {
                digraph[current_pointer - 1] = (presstime[current_pointer - 1] +
                presstime[current_pointer] + difference[current_pointer - 1]);
                if (current_pointer > 1) {
                trigraph[current_pointer - 2] = (presstime[current_pointer - 2] +
                presstime[current_pointer - 1] + presstime[current_pointer] +
                difference[current_pointer - 1] + difference[current_pointer - 2]);
                } } }
                return false;
            }
});
```

Normalization:

```
t0.getLocationOnScreen(location);
t0.post(new Runnable() {
        public void run() {
                contentWidth = t0.getWidth()+location[0];
                contentHeight = t0.getHeight()+location[1];
                if(contentWidth!=0){
                t0_x=((t0_x-location[0])/(contentWidth-location[0]))/10;
                t0_y=((t0_y-location[1])/(contentHeight-   location[1]))/10;
                t_x[current_pointer - 1]=t0_x;
                t_y[current_pointer - 1]=t0_y;
}} });
```

6.3. Training

We have used the Error Back Propagation Training Algorithm (EBPTA) to train the neural network. This algorithm creates a neural network with 3 layers - input layer, hidden layer and output layer. The input layer consists of 45 input nodes which correspond to the 45 characteristic features unique to a user. One hidden layer with 16 hidden nodes has been taken which gives the best efficiency. Two output nodes correspond to the two classes to which classification is done i.e. Legitimate, Illegitimate. The code snippet for EBPTA is given below,

```
public class errorBackPropagation {
public static double input_layer[];
public static double hidden_layer[];
public static double weight_input_to_hidden[][];
public static double input_hidden_layer[];
public static double input_output_layer[];
public static double output_layer[];
public static double target_output[];
public static double weight_hidden_to_output[][];
public static final double learning_rate=0.7;
public static double error_output[];
public static double error_hidden[];
public static double change_in_weight_hidden_and_output[][];
public static double change_in_weight_hidden_and_input[][];
Context con;
errorBackPropagation(Context con){
    this.con=con;
}
public void initialiseWeightsAndBias(){
    weight_input_to_hidden=new double[input_layer.length][hidden_layer.length];
    weight_hidden_to_output=new double[hidden_layer.length][output_layer.length];
}
public void initialiseInputLayer(){
```

```
    input_layer=new double[45];
}
public void initialiseHiddenLayer(){
    hidden_layer=new double[16];
}
public void initialiseOutputLayer(){
   output_layer=new double[2];
}
public void initialiseTargetOutput(){
   target_output=new double[2];
}
public void setLegitimatetargetOutput(){
    target_output[0]=1;
    target_output[1]=0;
}
public void initialiseError(){
    error_output=new double[output_layer.length];
    error_hidden=new double[hidden_layer.length];
}
public void initialiseChangeInWeightsAndBias(){
    change_in_weight_hidden_and_output=new
    double[hidden_layer.length][output_layer.length];
    change_in_weight_hidden_and_input=new double[input_layer.length][hidden_layer.length];
}
public void initialiseInputForHiddenLayer(){
    hidden_layer=new double[hidden_layer.length];
}
public void initialiseInputForOutputLayer(){
    input_output_layer=new double[output_layer.length];
}
public double activationFunction(double lambda,int i) {
```

```
    double a=Math.exp(lambda);
    double b=1.0/(double)a;
    double c=1.0+b;
    double x=1.0/(double)c;
    return x;
}
public double derivativeOfActivationFunction(double output_layer) {
    return output_layer*(1-output_layer);
}
public void functionHiddenLayer(){
    for(int i=0;i<input_hidden_layer.length;i++){
    for(int j=0;j<input_layer.length;j++){
    input_hidden_layer[i]+=input_layer[j]*weight_input_to_hidden[j][i];
    }
    hidden_layer[i]=activationFunction(input_hidden_layer[i],i);
}}
public void functionOutputLayer(){
    for(int i=0;i<input_output_layer.length;i++){
    for(int j=0;j<hidden_layer.length;j++){
    input_output_layer[i]+=hidden_layer[j]*weight_hidden_to_output[j][i];
    }
    output_layer[i]=activationFunction(input_output_layer[i],i);
}}
public void functionErrorOutputLayer(){
    for(int i=0;i<output_layer.length;i++){
    error_output[i]=(target_output[i]-
    output_layer[i])*derivativeOfActivationFunction(output_layer[i]);
} }
public void functionChangeInWeightsHiddenToOutput() {
    for(int i=0;i<output_layer.length;i++){
    for(int j=0;j<hidden_layer.length;j++){
```

```
    change_in_weight_hidden_and_output[j][i]=learning_rate*error_output[i]*hidden_layer[j];
}}}
public void functionErrorHiddenLayer(){
    for(int j=0;j<hidden_layer.length;j++){
    double x=0;
    for(int k=0;k<output_layer.length;k++){
    x+=error_output[k]*weight_hidden_to_output[j][k];
    }
    error_hidden[j]=x*derivativeOfActivationFunction(hidden_layer[j]);
}}
public void functionChangeInWeightsInputToHidden() {
    for(int i=0;i<hidden_layer.length;i++){
    for(int j=0;j<input_layer.length;j++){
    change_in_weight_hidden_and_input[j][i]=learning_rate*error_hidden[i]*input_layer[j];
}}}

//Weight Updation
public void updateWeights(){
    for(int i=0;i<output_layer.length;i++){
    for(int j=0;j<hidden_layer.length;j++){
    weight_hidden_to_output[j][i]=weight_hidden_to_output[j][i]+change_in_weight_hidden_an
    d_output[j][i];
    }}
    for(int i=0;i<hidden_layer.length;i++){
    for(int j=0;j<input_layer.length;j++){
    weight_input_to_hidden[j][i]=weight_input_to_hidden[j][i]+change_in_weight_hidden_and_i
    nput[j][i];
}}}
public void randomizeWeights(){
    int i=0;
    for(;i<input_layer.length;i++) {
```

```
    for(int j=0;j<hidden_layer.length;) {
    double min = 0 ;
    double max = (3)/Math.sqrt(45);
    double x = Math.random();
     if(x>=min && x<=max) {
    weight_input_to_hidden[i][j] = x;
    j++;
    }}}
    i=0;
    for(;i<hidden_layer.length;i++) {
    for(int j=0;j<output_layer.length;j++) {
    double min = 0 ;
    double max = (3)/Math.sqrt(45);
    double x = Math.random();
    if(x>=min && x<=max) {
    weight_hidden_to_output[i][j] = x ;
}}}}

//Testing Methods
public void getInputLayer(double input_layer[]){
this.input_layer=input_layer;
}}
```

Chapter 7

TESTING

7.1. Test Cases

Number	Description	Expected Output	Actual Output	Result
1	Password Module: It ask the user to set a password	Password is set successfully by the user	Password should be set successfully by the user	Pass
2	Feature Extractor module: It extracts the keystroke features by asking user to type the password for several times	Keystrokes features is extracted successfully	Keystrokes features should be extracted successfully	Pass
3	Training module: It trains the neural network using EBPTA algorithm according to the features extracted from the user	The neural network is trained successfully according to the features provided by the user	The neural network should be trained successfully according to the features provided by the user	Pass

4	Testing Module: The testing or decision module decides whether the user entering the password is legitimate or illegitimate	The decision is legitimate or illegitimate according to the keystroke features of the user	The decision should be whether the user entering the password is legitimate or illegitimate	Pass

7.2. Bottom-Up Integration Testing

The testing strategy that we use is Bottom-up Integration Testing. The System is divided into two modules namely the training module (M1) and testing module (M2). The training module further has 3 components which are Pre processor (C1), Feed Forward Neural Network (C2) and Error Generator (C3) .The testing module has 2 components which are Pre processor (C4), Trained Neural Network (C5) and Decision module (C6). We start by unit testing all 3 individual components of module M1 against the test cases to see if they correctly perform the function they are intended for and then they are integrated it into a complete module. Now the M1 is tested as a whole. Same type of integration testing is done for module M2. In the end, the entire system S which is the integration of M1 and M2 is tested as a whole against the test cases. The flow of testing is shown in Figure 7.2.1

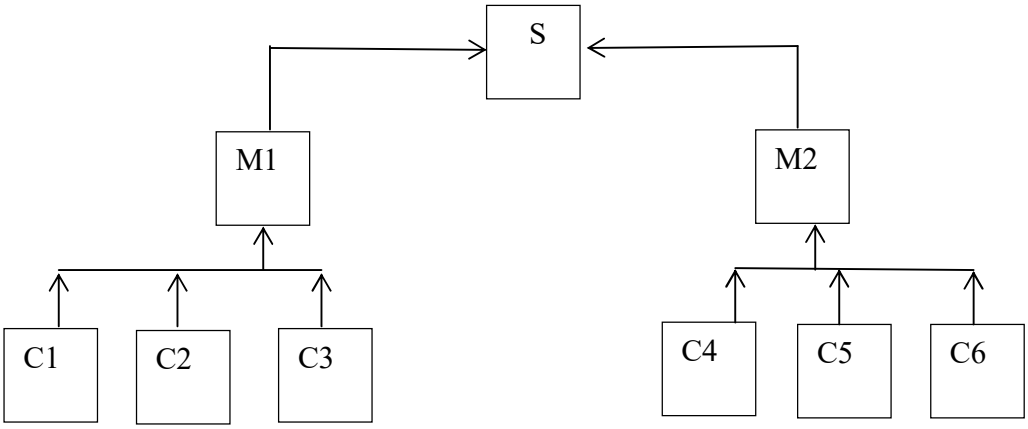


Figure 7.2. 1 Flow Of Testing

The final testing of the entire system is a black box testing which is done to assess the efficiency and accuracy. The system takes user's features as input and outputs the class to which it belongs to. A testing dataset consisting of legitimate values and illegitimate values was prepared by taking user inputs from different people for the same password. This dataset was fed to the fully trained neural network to assess the efficiency and accuracy of the system. The final outcome achieved after this process was used to adjust the parameters in the project rate, number of hidden neurons to achieve better test results.

Section 8

RESULTS AND DISCUSSIONS

The results presented in this section are an outcome of testing 3 users over the same password '171094' over 10 iterations each. We derived two set of results discussed below.

The first set of results was derived during the training phase which indicates the dissimilarity between the users over different features which can be used to distinguish a legitimate user from others. The graphs shown below are the average values for 3 users over same password '171094' for 10 iterations.

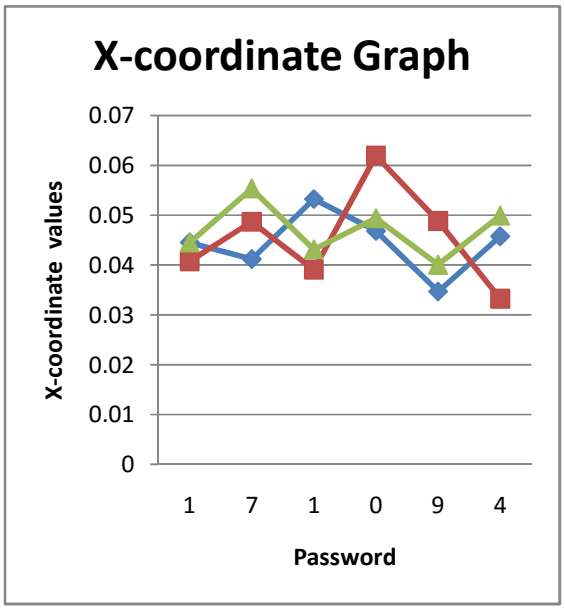


Figure 8.1 X-coordinate Graph

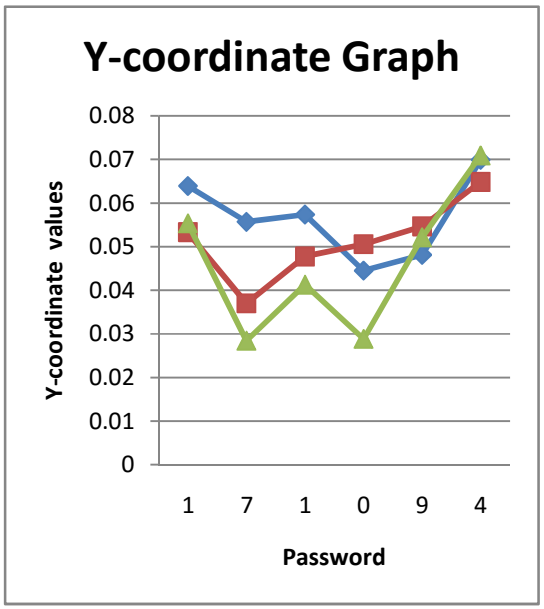


Figure 8.2 Y-coordinate Graph

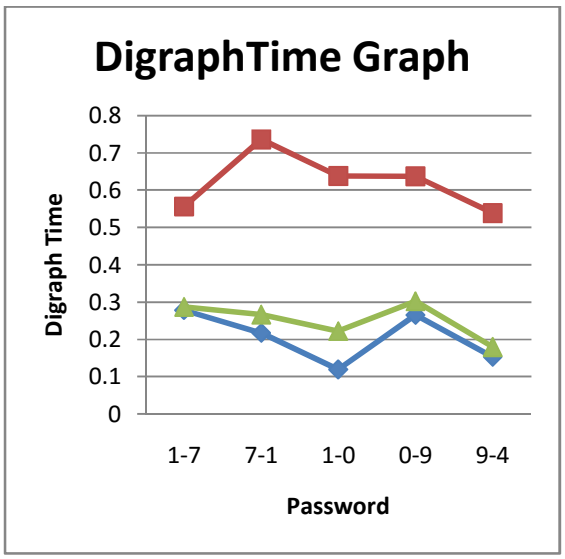


Figure 8.3 DigraphTime Graph

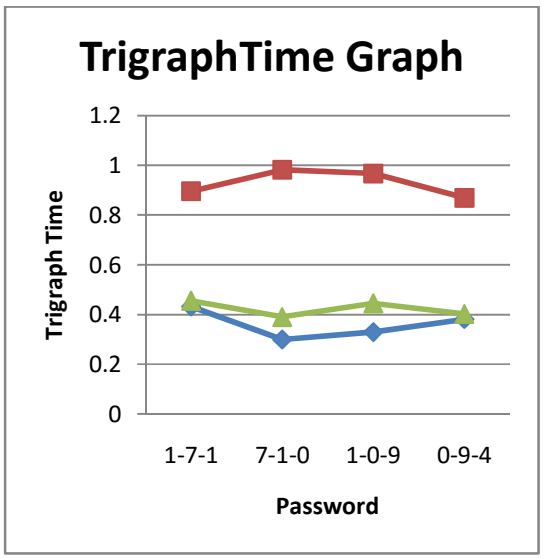


Figure 8.4 TrigraphTime Graph

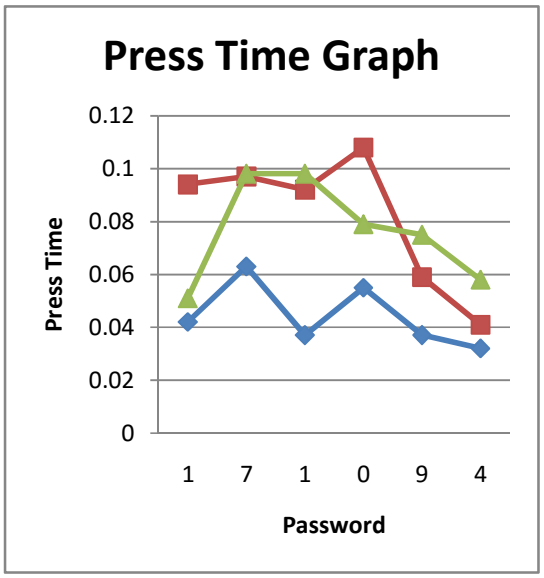


Figure 8.5 PressTime Graph

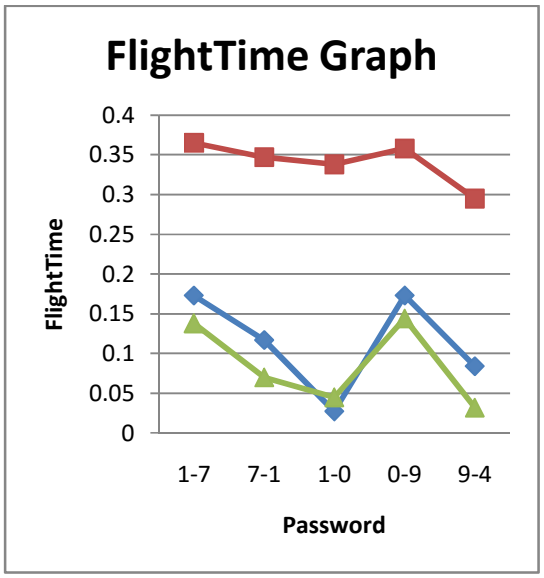


Figure 8.6 FlightTime Graph

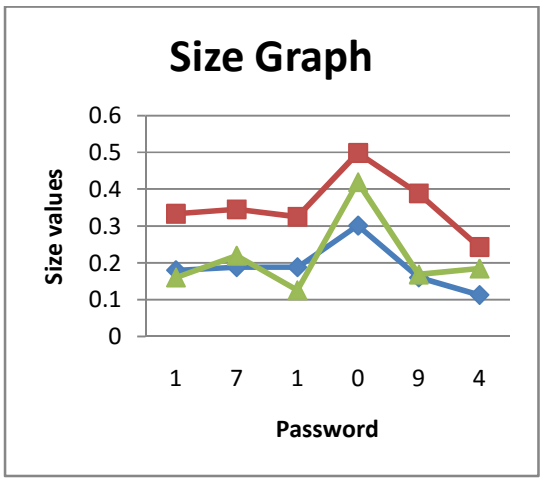


Figure 8.7 Size Graph

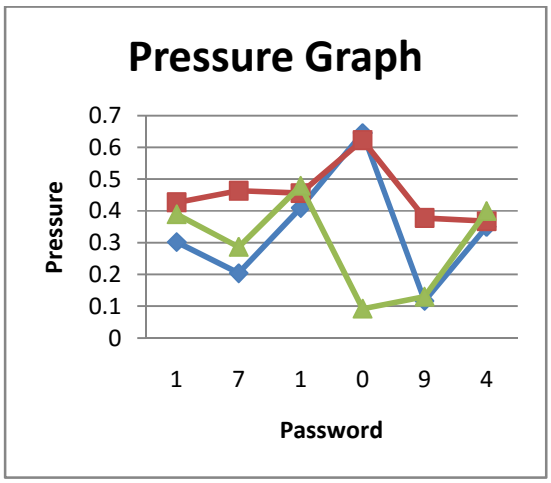


Figure 8.8 Pressure Graph

The second set of results was derived during the final step of integration testing where the system is tested as whole. Here, we tested the system with 40 different users without distinguishing between legitimate and illegitimate user. The trained network then predicted its results for different users. These results were then used to calculate the False Rejection Rate (FRR) and False Acceptance Rate (FAR),

False Rejection Rate (FRR): Number of times when a legitimate user was denied access by the system.

$$\text{FRR} = \frac{Number\ of\ false\ rejections}{Total\ number\ of\ genuine\ match\ attempts}$$

For our case, FRR was 9 i.e. 9 times a legitimate user was denied access by the system.

False Acceptance Rate (FAR): Number of times when an imposter was granted access to the system.

$$\text{FAR} = \frac{Number\ of\ false\ matches}{Total\ number\ of\ imposter\ match\ attempts}$$

For our case, FAR was 7 i.e. 7 times an imposter was granted access by the system.

Section 9

CONCLUSIONS & FUTURE SCOPE

Security has become one of the most important concerns in today's world. A strong secure authentication system cannot solely rely on password based authentication, however a basic password authentication hardened with keystroke dynamics can improve security to a great extent. This means the user not only needs to enter the correct password but also use legitimate user's correct typing pattern. Moreover Advanced mobile technology provides a rich sensor environment to cope with the new security challenges unforeseen by traditional computing devices. Although physiological biometrics, such as fingerprint, provides a highly accurate one time authentication, it can be easily spoofed. Mobile keystroke dynamics enables a non-intrusive continuous authentication modality which has the potential to provide a highly accurate and secure solution to overcome mobile identity challenges. Key taping pressure and accelerometer sensory data provide additional useful signatures for mobile user authentication. Future work in mobile keystroke authentication study should investigate its effectiveness on more complex and realistic use cases and testing on a larger subject pool. In addition, other sensor modalities, such as gyroscope and face image, provided by mobile device should be considered for a comprehensive authentication solution for mobile security.

Section 10

REFERENCES / BIBLIOGRAPHY

[1] User Authentication using keystroke dynamics for cellular phone : P. Campisi, E. Maiorana, M.LoBosco, A. Neri.

[2] Intrusion Detection Using Keystroke Dynamics & Fuzzy Logic Membership Function : Mahalaxmi Sridhar, Siddesh Vaidya ,Piyush Yawalkar.

[3] Effect of Dimensionality Reduction on Performance in Artificial Neural Network for User Authentication: Sucheta Chauhan, Prema K.V.

[4] Keystroke dynamics advances for mobile devices using deep neural network, Yunbin Deng, Yu Zhong.

[5] I sensed it was you: Authenticating Mobile users with sensor-enhanced keystroke dynamics, Cristiano Giuffrida, KarnilMajdanik, Mauro Conti, Herbert Bos.

[6] A model to secure Mobile devices using keystroke dynamics through soft computing techniques In: International journal of Soft computing and Engineering, ISSN:2331-2307, Volume-2, Issue-3, M. Karnan, N. Krishnaraj.

[7] Strengthening user authentication on mobile phones with keystroke dynamics, Grant Ho, TapDynamics.

[8]) User authentication with keystroke dynamics using fixed text In: International conference on Biometrics and Kansei Engineering, Mariusz Rybnik, PiotrPanasiuk, Khalid Saeed

APPENDIX

Akshat Shah, Parth Shah, Hitarth Shah and Chetashri Bhadane, 'Strengthening User authentication Using Keystroke Dynamics', International Journal of Engineering and Research Technology (IJERT), Volume. 4, Issue. 11, November – 2015, ISSN: 2278-0181.

ACKNOWLEDGEMENT

This project consumed huge amount of work, research and dedication. Still, implementation would not have been possible if we did not have a support of our college and teachers. Therefore we would like to extend our sincere gratitude to all of them.

We are highly indebted to our **project guide Prof. Chetashri Bhadane** for her guidance and constant supervision as well as for providing necessary information regarding the project & also for her support in completing the project.

We would also like to express our gratitude towards our **Principal Dr. Hari Vasudevan** and our **Head of Department, Dr. Narendra M. Shekokar** for providing us with all the facilities to complete our project. Lastly, we would like to thank our family and friends for supporting us in our project.