CHAPTER 1

INTRODUCTION

1.1 DOMAIN INTRODUCTION:

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal exemption from a rule in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them. It is among swiftly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Scrutinize and manipulating the image which includes data compression and image enhancement and detect patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

Purpose of Image processing

The purpose of image processing is divided into 5 groups. They are:

- Visualization - Observe the objects that are not visible.
- Image sharpening and restoration - To create a better image.
- Image retrieval - Seek for the image of interest.
- Measurement of pattern – Measures various objects in an image.
- Image Recognition – Distinguish the objects in an image.

Types

The two types of methods used for Image Processing are Analog and Digital Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs.

Image analysts use various fundamentals of explanation while using these visual techniques. The image processing is not just cramped to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and indemnity data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computer. As raw data from imaging sensors from satellite platform contains inadequacy.

To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

1.2 PROJECT INTRODUCTION:

Authentication schemes which are not resilient to observation are unprotected to shoulder-surfing. Any kind of visual information may be observed, including the blink of a button when it is pressed, or even the exhausted oil fields that the fingers leave on a touch screen. Shoulder-surfing is a big threat for PIN authentication in particular, because it is relatively easy for an observer to follow the PIN authentication process. PINs are short and require just a small numeric keypad instead of the usual alphanumeric keyboard. In addition, PIN authentication is often performed in crowded places, e.g., when someone is unlocking her mobile phone on the street or in the subway. Shoulder-surfing is facilitated in such scenarios since it is easier for an attacker to stand close to the user while escaping her attention. We designed Illusion PIN (IPIN) for touch screen devices.

The virtual keypad of IPIN is composed of two keypads with different digit orderings, blended in a single hybrid image. The user who is close to the screen is able to see and use one keypad, but a potential attacker who is looking at the screen from a bigger distance, is able to see only the other keypad.

GOALS FOR IPIN

IPIN is to meet the following goals:

- The user's keypad is shuffled in every authentication attempt since the attacker may memorize the spatial arrangement of the pressed digits.
- The main **goal** of our work was to design a **PIN**-based authentication scheme that would be resistant against shoulder surfing attacks. To this end, we created **Illusion PIN**.

NEED FOR IPIN

- An IPIN provides more security for you, your employees and customers by providing less risk of robbery and employee theft.
- An ATM transaction by using IPIN is guaranteed and eliminates charge backs, disputes, credit card fees and bad checks.
- By using IPIN your customer makes electronic deposits directly into your bank account, which saves you both time and work.

1.3 LITERATURE SURVEY

[1] This paper presents an integrated evaluation of the Persuasive Cued Click-Points graphical password scheme, including usability and security evaluations, and implementation considerations. An important usability goal for knowledge-based authentication systems is to support users in selecting passwords of higher security, in the sense of being from an expanded effective security space. We use persuasion to influence user choice in click-based graphical passwords, encouraging users to select more random, and hence more difficult to guess, click points.

Advantages:

- To increase the system security.
- To more user friendly.

Disadvantage:

- Difficulty to change the password**.**

[2] Since current signatures are generally not verified carefully, frauds by forging others signature always happen. This paper tried to authenticate user automatically with electronic signatures on mobile device. We collected coordinates, pressure, contact area and other biometric data when users sign their name on touch screen smart phone. Then we used four different classification algorithms, Support Vector Machine, Logistic Regression.

Advantages:

- Increase robustness for on-line signature verification

Disadvantages:

- Forgery Biometrics can be used for verification.

[3] This paper presents Draw-A-PIN, a user authentication system on a device with a touch interface that supports the use of PINs.

In the proposed system, the user is asked to draw her PIN on the touch screen instead of typing it on a keypad. Consequently, Draw-A-PIN could offer better security by utilizing drawing traits or behavioural biometrics as an additional authentication factor beyond just the secrecy of the PIN. In addition, Draw-A-PIN inherently provides acceptability and usability by anchorage user familiarity with PINs.

Advantages:

- Provide two-factor user authentication system for touch screen.

Disadvantages:

- limitations on digit recognition rate and its performance in behavioral biometric verification

[4] Secure user identification is important for the increasing number of eyewear computers but limited input capabilities pose significant usability challenges for established knowledge-based schemes, such as passwords or PINs.

We present Skull Conduct, a biometric system that uses bone conduction of sound through the user's skull as well as a microphone readily integrated into many of these devices, such as Google Glass. At the core of Skull Conduct is a method to analyze the characteristic frequency response created by the user's skull using a combination of Mel Frequency Cepstral Coefficient (MFCC) features as well as a computationally light-weight 1NN classifier.

Advantages:

- It serves as a robust biometric authentication**.**

Disadvantages:

- White noise signals may be unpleasant for the user.

[5] Nowadays, touch screen mobile phones make up a larger and larger share in the mobile market. Users also often use their mobile phones (e.g., Android phones) to store personal and sensitive data. It is therefore important to safeguard mobile phones by authenticating legitimate users and detecting impostors. In this paper, we propose a novel user authentication scheme based on touch dynamics that uses a set of behavioral features related to touch dynamics for accurate user authentication. In particular, we construct and select 21 features that can be used for user authentication.

To evaluate the performance of our scheme, we collect and analyze touch gesture data of 20 Android phone users by comparing several known machine learning classifiers.

Advantages:

- Draw pattern user authentication system for touch screen.

Disadvantages:

- Pass matrix its performance in behavioral verification

[6] According to a previous study [39], authentication schemes that require users to touch or fling on computer monitors or display screens during the login phase are vulnerable to smudge attacks. The attacker can obtain the user's password easily by observing the smudge left on the touch screen.

Advantages:

- It is User friendly.

Disadvantages:

- Android touch screen pattern in behavioral verification

[7] In line with the recent call for technology on Image Based Authentication (IBA) in JPEG committee [1], we present a novel graphical password design in this paper. It rests on the human cognitive ability of association-based memorization to make the authentication more user-friendly, comparing with traditional textual password.

Based on the principle of zero-knowledge proof protocol, we further improve our primary design to overcome the shoulder-surfing attack issue without adding any extra complexity into the authentication procedure.

Advantages:

- To increase the system security.
- To more user friendly.

Disadvantages:

- Difficulty to change the password**.**

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

- Processor : Dual core processor 2.6.0 GHZ
- RAM : 1GB
- Hard disk : 160 GB
- Compact Disk : 650 Mbs
- Keyboard : Standard keyboard
- Monitor : 15 inch color monitor

2.2 SOFTWARE REQUIREMENTS

- Operating System : Windows OS
- Front End : JAVA
- IDE : NetBeans
- Back End : SQL SERVER

2.3 SOFTWARE DESCRIPTION

2.3.1 ABOUT JAVA:

Java is a small, simple, safe, object oriented, interpreted or dynamically optimized, byte coded, architectural, garbage collected, multithreaded programming language with a strongly typed exception-handling for writing distributed and dynamically extensible programs.

Java is an object oriented programming language. Java is a high-level, third generation language like C, FORTRAN, Small talk, Pearl and many others. You can use java to write computer applications that crunch numbers, process words, play games, store data or do any of the thousands of other things computer software can do.

Special programs called applets that can be downloaded from the internet and played safely within a web browser. Java a supports this application and the follow features make it one of the best programming languages.

- It is simple and object oriented

- It helps to create user friendly interfaces.
- It is very dynamic.
- It supports multithreading and internet programming.
- It is platform independent
- It is highly secure and robust.

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun's 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 which can run on any Java virtual machine (JVM) regardless of computer architecture.

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 made available most of their Java technologies as free software 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.

2.3.2 PRIMARY GOALS

There were four primary goals in the creation of the Java language:

- It should use the object-oriented programming methodology
- It should contain built-in support for using computer networks.
- It should be designed to execute code from remote sources securely.
- It should be easy to use by selecting what were considered the good parts of other object-oriented languages.

The Java platform is the name for a bundle of related programs, or platform, from Sun which allow for developing and running programs written in the Java programming language. The platform is not specific to any one processor or operating system, but rather an execution engine (called a virtual machine) and a compiler with a set of standard libraries which are implemented for various hardware and operating systems so that Java programs can run identically on all of them.

Different "editions" of the platform are available, including:

- Java ME (Micro Edition): Specifies several different sets of libraries (known as profiles) for devices which are sufficiently limited that supplying the full set of Java libraries would take up unacceptably large amounts of storage.
- Java SE (Standard Edition): For general purpose use on desktop PCs, servers and similar devices.
- Java EE (Enterprise Edition): Java SE plus various APIs useful for multi-tier client-server enterprise applications.

The Java Platform consists of several programs, each of which provides a distinct portion of its overall capabilities. For example, the Java compiler, which converts Java source code into Java byte code (an intermediate language for the Java Virtual Machine (JVM)), is provided as part of the Java Development Kit (JDK). The sophisticated Java Runtime Environment (JRE), complementing the JVM with a just-in-time (JIT) compiler, converts intermediate byte code into native machine code on the fly. Also supplied are extensive libraries (pre-compiled into Java byte code) containing reusable code, as well as numerous ways for Java applications to be deployed, including being embedded in a web page as an applet. There are several other components, some available only in certain editions.

2.3.2.1 JAVA VIRTUAL MACHINE

The heart of the Java Platform is the concept of a "virtual machine" that executes Java bytecode programs. This bytecode is the same no matter what hardware or operating system the program is running under. There is a JIT compiler within the Java Virtual Machine, or JVM. The JIT compiler translates the Java bytecode into native processor instructions at run-time and caches the native code in memory during execution.

The use of bytecode as an intermediate language permits Java programs to run on any platform that has a virtual machine available. The use of a JIT compiler means that Java applications, after a short delay during loading and once they have "warmed up" by being all or mostly JIT-compiled, tend to run about as fast as native programs. Since JRE version 1.2, Sun's JVM implementation has included a just-in-time compiler instead of an interpreter.

2.3.2.2 CLASS LIBRARIES

In most modern operating systems, a large body of reusable code is provided to simplify the programmer's job.

This code is typically provided as a set of dynamically loadable libraries that applications can call at runtime. Because the Java Platform is not dependent on any specific operating system, applications cannot rely on any of the existing libraries. Instead, the Java Platform provides a comprehensive set of standard class libraries, containing much of the same reusable functions commonly found in modern operating systems.

The Java class libraries serve three purposes within the Java Platform. Like other standard code libraries, they provide the programmer a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing. In addition, the class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as network access and file access are often heavily dependent on the native capabilities of the platform..

2.3.2.3 PLATFORM INDEPENDENCE

One characteristic, platform independence, means that programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere.

This is achieved by most Java compilers by compiling the Java language code halfway (to Java bytecode) – simplified machine instructions specific to the Java platform. The code is then run on a virtual machine (VM), a program written in native code on the host hardware that interprets and executes generic Java bytecode. (In some JVM versions, bytecode can also be compiled to native code, either before or during program execution, resulting in faster execution.) Further, standardized libraries are provided to allow access to features of the host machines (such as graphics, threading and networking) in unified ways. Note that, although there is an explicit compiling stage, at some point, the Java bytecode is interpreted or converted to native machine code by the JIT compiler.

The first implementations of the language used an interpreted virtual machine to achieve portability. These implementations produced programs that ran more slowly than programs compiled to native executables, for instance written in C or C++, so the language suffered a reputation for poor performance.

This technique, known as just-in-time compilation (JIT), translates the Java bytecode into native code at the time that the program is run, which results in a program that executes faster than interpreted code but also incurs compilation overhead during execution. More sophisticated VMs use dynamic recompilation, in which the VM can analyze the behavior of the running program and selectively recompile and optimize critical parts of the program.

Dynamic recompilation can achieve optimizations superior to static compilation because the dynamic compiler can base optimizations on knowledge about the runtime environment and the set of loaded classes, and can identify the hot spots (parts of the program, often inner loops, that take up the most execution time).

Another technique, commonly known as static compilation, is to compile directly into native code like a more traditional compiler. Static Java compilers, such as GCJ, translate the Java language code to native object code, removing the intermediate bytecode stage. This achieves good performance compared to interpretation, but at the expense of portability.

2.3.2.4 AUTOMATIC MEMORY MANAGEMENT

One of the ideas behind Java's automatic memory management model is that programmers be spared the burden of having to perform manual memory management. In some languages the programmer allocates memory for the creation of objects stored on the heap and the responsibility of later deal locating that memory also resides with the programmer. If the programmer forgets to deallocate memory or writes code that fails to do so, a memory leak occurs and the program can consume an arbitrarily large amount of memory.

In Java, this potential problem is avoided by automatic garbage collection. The programmer determines when objects are created, and the Java runtime is responsible for managing the object's lifecycle. The program or other objects can reference an object by holding a reference to it (which, from a low-level point of view, is its address on the heap). When no references to an object remain, the Java garbage collector automatically deletes the unreachable object, freeing memory and preventing a memory leak.

Memory leaks may still occur if a programmer's code holds a reference to an object that is no longer needed—in other words, they can still occur but at higher conceptual levels. The use of garbage collection in a language can also affect programming paradigms.

2.3.2.5 PERFORMANCE

Java's performance has improved substantially since the early versions, and performance of JIT compilers relative to native compilers has in some tests been shown to be quite similar. The performance of the compilers does not necessarily indicate the performance of the compiled code; only careful testing can reveal the true performance issues in any system.

2.4 JAVA RUNTIME ENVIRONMENT

The Java Runtime Environment, or JRE, is the software required to run any application deployed on the Java Platform. End-users commonly use a JRE in software packages and Web browser plugins. Sun also distributes a superset of the JRE called the Java 2 SDK (more commonly known as the JDK), which includes development tools such as the Java compiler, Javadoc, Jar and debugger.

Moreover, in runtime engine environments such as Java there exist tools that attach to the runtime engine and every time that an exception of interest occurs they record debugging information that existed in memory at the time the exception was thrown (stack and heap values). These Automated Exception Handling tools provide 'root-cause' information for exceptions in Java programs that run in production, testing or development environments.

2.5 ECLIPSE:

Eclipse is created by an Open Source community and is used in several different areas, e.g. as a development environment for Java or Android applications. Eclipse's roots go back to 2001.

The Eclipse Open Source community has over 200 Open Source projects covering different aspects of software development.

The Eclipse projects are governed by the Eclipse Foundation. The Eclipse Foundation is a non-profit, member supported corporation that hosts the Eclipse Open Source projects and helps to cultivate both an Open Source community and an ecosystem of complementary products and services.

The Eclipse IDE can be extended with additional software components. Eclipse Call these software components plug-ins. Several Open Source projects and companies have extended the Eclipse IDE.

2.5.1 ECLIPSE PUBLIC LICENSE

The Eclipse Public License (EPL) is an Open Source software license used by the Eclipse Foundation for its software. The EPL is designed to be business-friendly. EPL licensed programs can be used, modified, copied and distributed free of charge and receiver of EPL-licensed software can choose to use this software in closed source programs. Only modifications in the original EPL code must be released.

The Eclipse Foundation also validates that source code contributed to Eclipse projects is free of Intellectual property (IP) issues. This process is known as IP cleansing.

The permissive EPL and the IP cleansing effort of the Eclipse Foundation makes reusing the source code of Eclipse projects attractive.

Software Testing:

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing can be stated as the process of validating and verifying that a computer program/application/product: meets the requirements that guided its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the software development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software.

Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers and other stakeholders. Software testing is the process of attempting to make this assessment.

Not all software defects are caused by coding errors. One common source of expensive defects is requirement gaps, e.g., unrecognized requirements which result in errors of omission by the program designer.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new computer hardware platform, alterations in source data, or interacting with different software. A single defect may result in a wide range of failure symptoms.

A fundamental problem with software testing is that testing under all combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to be versus what it is supposed to do)—usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

Logic coverage criterion:

A common way to test software is to execute tests that satisfy a coverage criterion. A coverage criterion imposes requirements on the tests. For example, one coverage criterion is statement coverage, which demands that every statement in the software be executed by the test set. A logic coverage criterion imposes requirements on tests related to logic expressions (predicates) in source code or other artifacts. For example, one logic coverage criterion is predicate coverage, which requires that each logic expression evaluate to TRUE in at least one test and FALSE in at least one test.

Logic coverage criteria differ in fault detection capability and test set size.

Many logic coverage criteria exist, but they can be broadly classified into two categories: semantic and syntactic. Semantic criteria make no assumption as to predicate format, whereas syntactic criteria do, with the most common format being minimal Disjunctive Normal Form (DNF). A common method for evaluating logic coverage criteria is to assess detection of faults in the minimal DNF fault hierarchy of Lau and Yu [30]. Kaminski and Amman [22] established a complementary minimal Conjunctive Normal Form (CNF) fault hierarchy.

Mutation Testing

Mutation testing was originally proposed by DeMillo et al. and Hamlet requires testers to create tests to detect a specified set of faults. Mutant programs that vary from the original program by a single syntactic change are generated. If possible, testers find inputs to distinguish the mutants from the original program to achieve a high mutation score. For a mutant to be distinguished (killed), the statement with the mutation must be reached (reachability), the program state for the mutant must differ from the program state of the original program after the mutated statement is executed (infection) and the difference in program state must propagate to the output (propagation). Mutation score is defined as the number of mutants distinguished (killed) divided by the number of non-equivalent mutants generated

The motivation for this project has two parts, one based on logic coverage criteria and the other based on logic mutation. In summary, testing approaches based on current logic coverage criteria or current mutation approaches share the drawback of not guaranteeing detection of certain logic faults (even when all mutants are killed) and/or are costly in terms of the number of tests required.

These drawbacks call for new logic coverage criteria and new logic mutation approaches and this research addresses exactly these drawbacks.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Person identification can be done precisely by Biometrical method, where physiological or behavioral characteristics are used for this purpose. Handwritten signature is a behavioral trait it can be used for person identification accurately. There are two types of identification modes either online or offline mode which depends upon the signature, acquisition method. In offline acquisition method the shape of the signature is used for authenticating signer. While in online signature verification uses dynamic characters (i.e.) dynamic time dependent of the signature to authenticate the signer. This paper describes the implementation on field programmable gate arrays (FPGAs) of an embedded system for online signature verification.

To prevent unauthorized parties from accessing data stored on their smartphones, users have the option of enabling a “lock screen” that requires a secret code (e.g., PIN, drawing a pattern, or biometric) to gain access to their devices. We present a detailed analysis of the smartphone locking mechanisms currently available to billions of smartphone users worldwide. We are able to show how existing lock screen mechanisms provide users with distinct tradeoffs between usability (unlocking speed vs. unlocking frequency) and security.. In the proposed system, the user is asked to draw her PIN on the touch screen instead of typing it on a keypad. It can also be implemented as a protection method for sensitive applications on the phone, like banking and email apps. We evaluate DRAW-A-PIN in different settings through extensive and unsupervised experiments.

Secure user identification is important for the increasing number of eyewear computers but limited input capabilities pose significant usability challenges for established knowledge based schemes, such as passwords or PINs.

The contributions of this work are two-fold. First, we present Skull-Conduct, a biometric system that uses bone conduction for secure user identification and authentication on eyewear computers. The system combines Mel Frequency Cepstral Coefficient (MFCC) based features with a light-weight 1NN classifier that can directly run on Google Glass.

Disadvantages

- Authentication schemes which are not resilient to observation are vulnerable to shoulder-surfing.
- Shoulder-surfing is a big threat for PIN authentication.
- Easy for an attacker to follow the PIN authentication process.

3.2 PROPOSED SYSTEM

Shoulder-surfing is a big threat for PIN authentication in particular, because it is relatively easy for an observer to follow the PIN authentication process. PINs are short and require just a small numeric keypad instead of the usual alphanumeric keyboard. In addition, PIN authentication is often performed in crowded places. Illusion PIN is a PIN-based authentication scheme for touch screen devices which offers shoulder-surfing resistance. Based on this, the core idea of Illusion PIN is to make the keypad on the touch screen to be interpreted with a different digit ordering when the viewing distance is adequately large. IPIN uses the technique of hybrid images to blend two keypads with different digit orderings in such a way, that the user who is close to the device is seeing one keypad to enter her PIN, while the attacker who is looking at the device from a bigger distance is seeing only the other keypad. To overcome shoulder-surfing attacks on authentication schemes by proposing Illusion PIN (IPIN), we create the keypad of Illusion PIN with the method of hybrid images and we call it a hybrid keypad.

Advantages

- Resistant against shoulder surfing attacks.
- Provide high security to user login pattern.
- Visibility algorithm is to give better visual representation during keypad authentication.

CHAPTER 4

SYSTEM DESIGNS

4.1 SYSTEM ARCHITECTURE

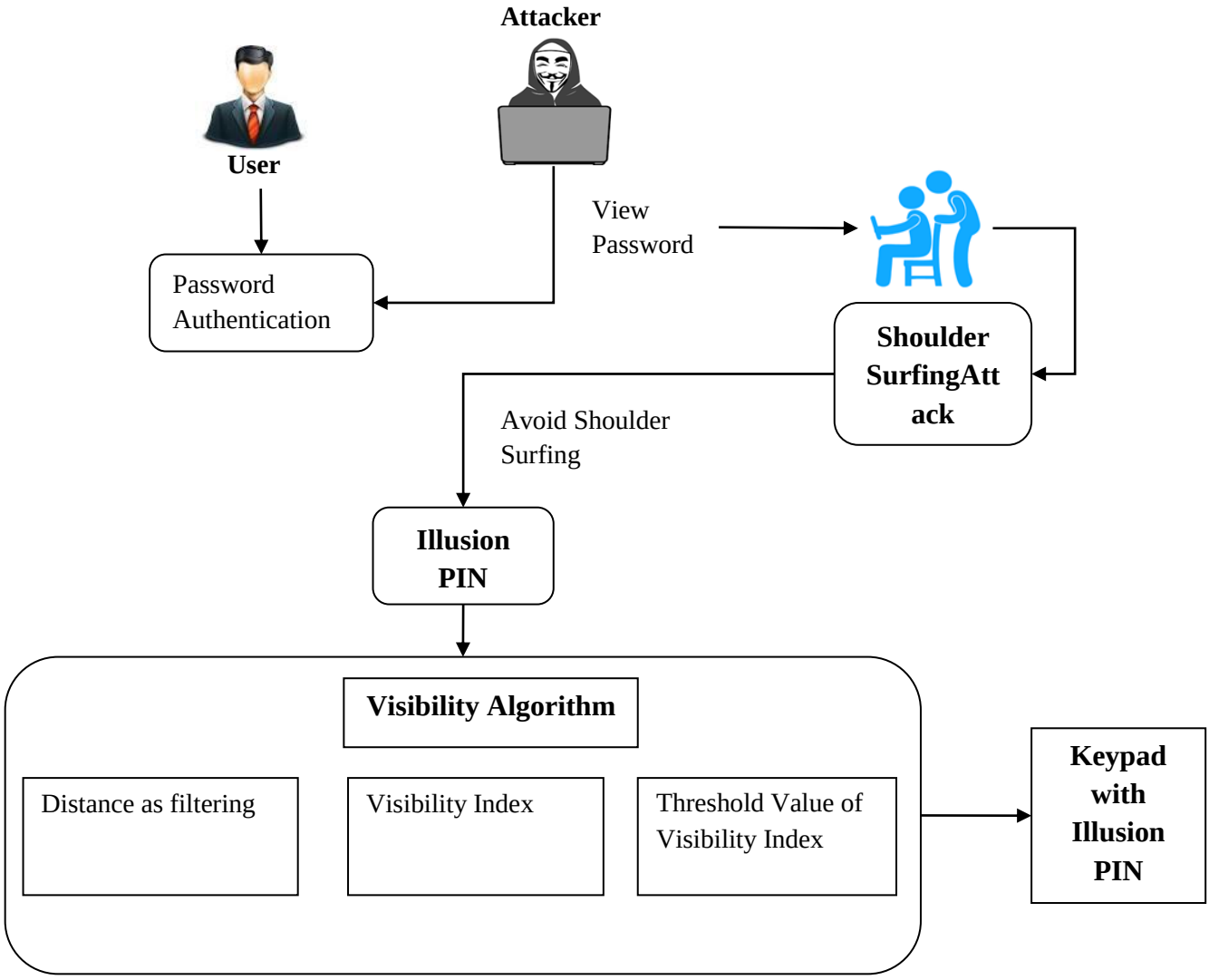


Figure 1:System Architecture

4.2 FLOW DIAGRAM

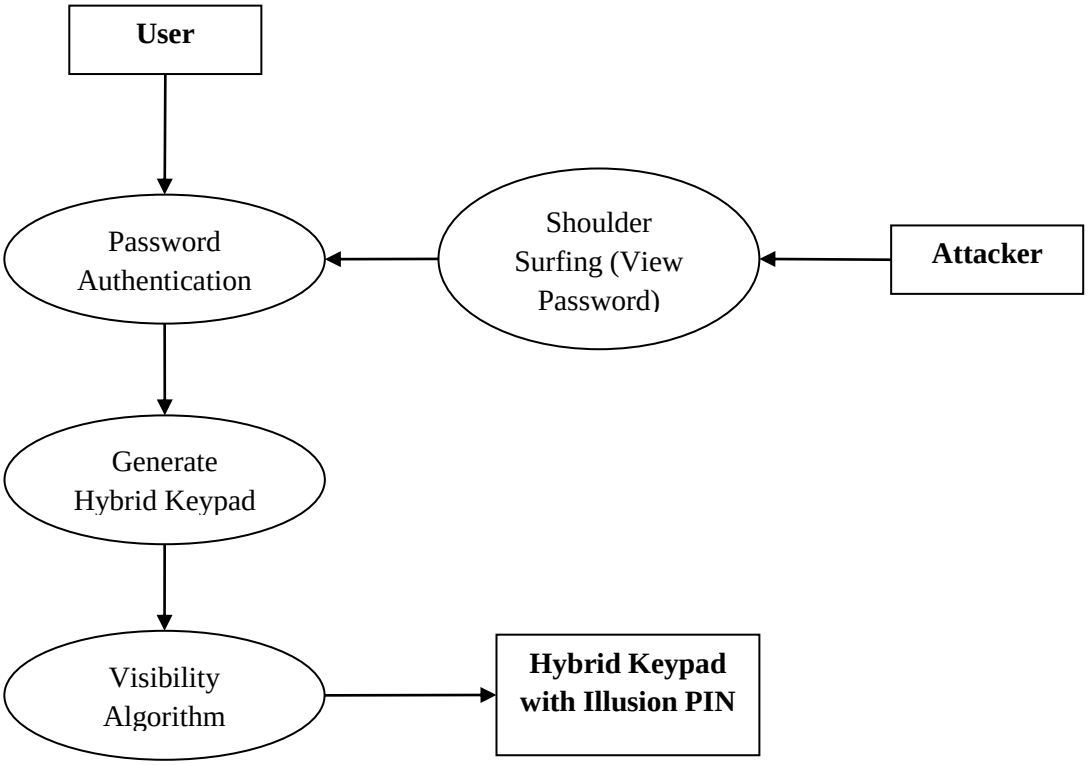


Figure 2:Flow Diagram

4.3 UML DIAGRAMS

4.3.1 USE CASE DIAGRAM

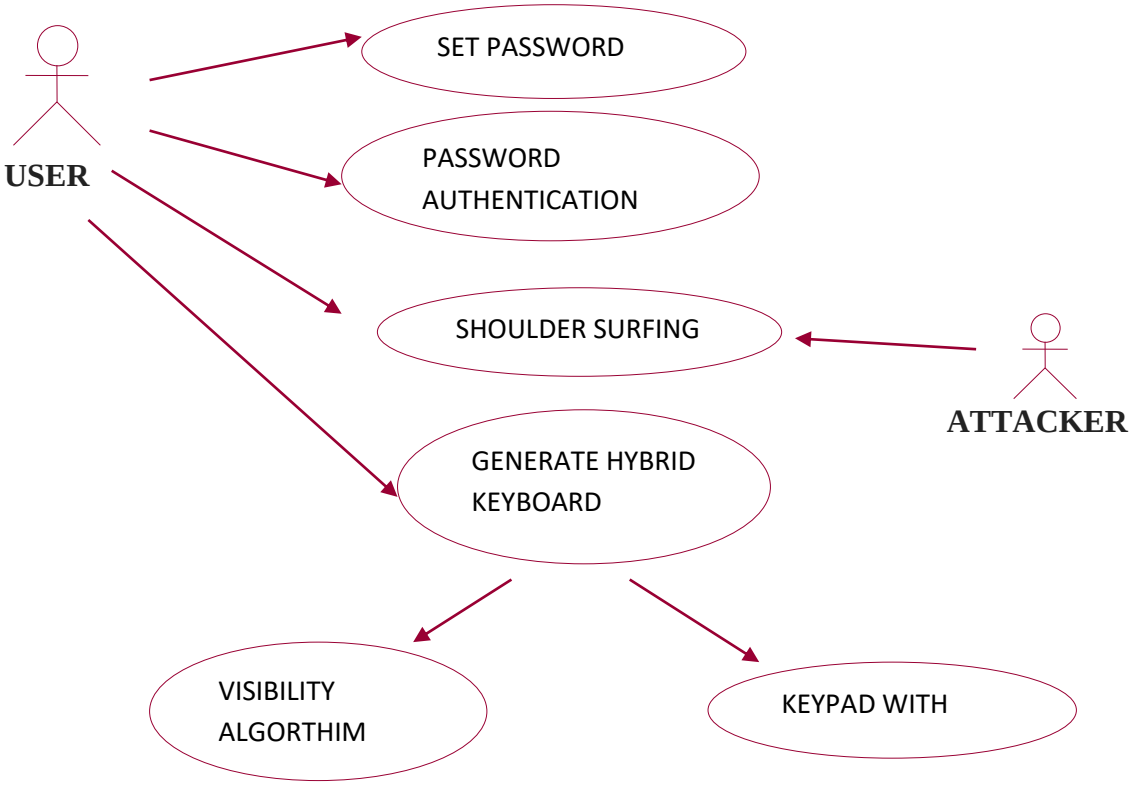


Figure 3: Use case Diagram

4.3.2 SEQUENCE DIAGRAM

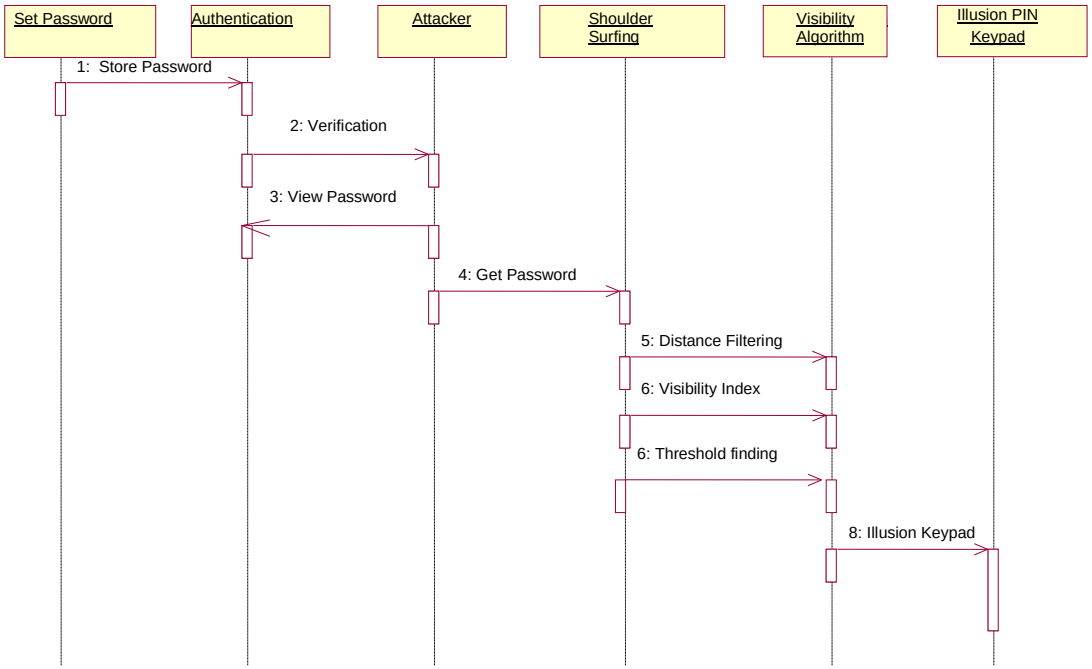


Figure 4: Sequence Diagram

4.3.3 CLASS DIAGRAM

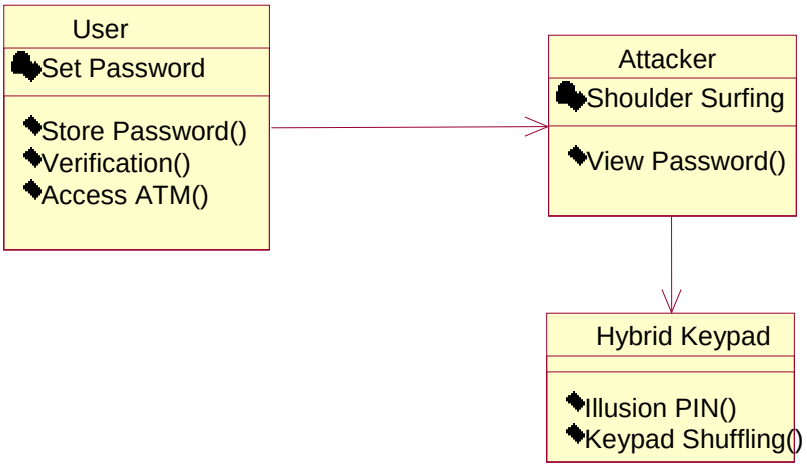


Figure 5: Class Diagram

4.3.4 ACTIVITY DIAGRAM

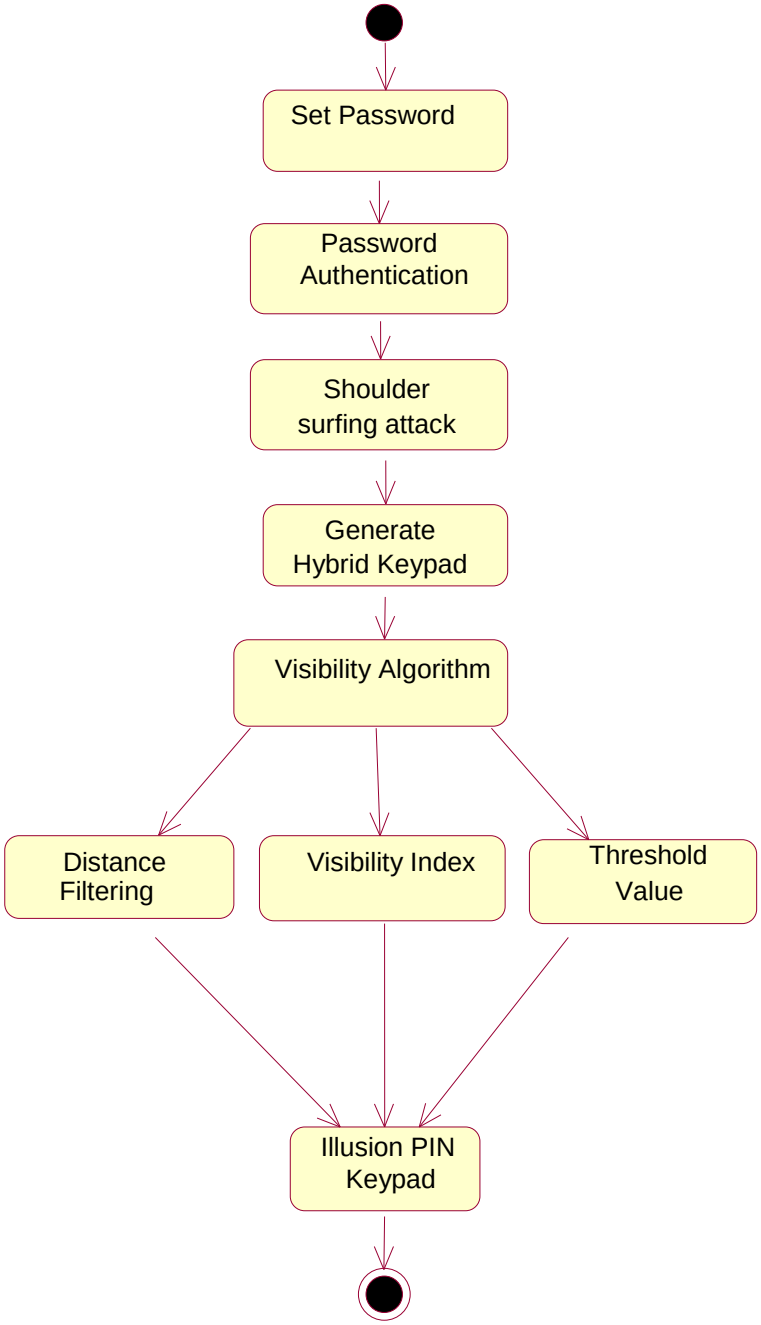


Figure 6: Activiy Diagram

Table Name: Customer Table

Field	Type	Constraints
Customer number	Int(15)	Not null
PIN	Int(15)	Not null
Checking account number	Int(15)	Not null
Checking account balance	Int(15)	Not null
Savings account number	Int(15)	Not null
Savings account balance	Int(15)	Not null
Customer name	Varchar(15)	Not null
Customer Password	Varchar(15)	Not null

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 MODULES

- User Password Authentication
- Shoulder Surfing Attack
- Illusion PIN Generation
- Visibility Algorithm
- Performance Evaluation

5.2 MODULE DESCRIPTION

User Password Authentication

A personal identification number is a numeric or alphanumeric password or code used in the process of authenticating or identifying a user to a system and system to a user. The User PIN Authentication page enables user to add user PIN records into the device one at a time, and to edit or delete user PIN records that have already been saved in the device. PINs are used in ATM or POS transactions, secure access control (e.g. computer access, smart phone access, door access, car access), internet transactions, or to log into a restricted website. If PIN Authentication is selected for one or more Device Functions on the Authentication Manager page, the user will be prompted for a PIN before they can access those Device Functions. If the PIN is entered incorrectly the user will be returned to the previous screen. When a PIN is entered correctly all functions that use that PIN are then accessible to the user.

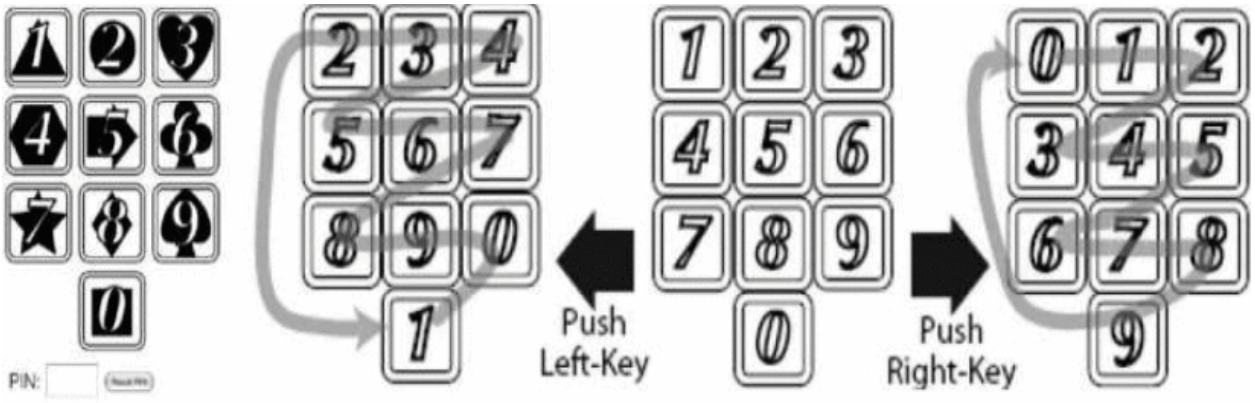
Shoulder Surfing Attack:

Shoulder surfing is using direct observation techniques, such as looking over someone's shoulder, to get information. Shoulder surfing can also be done long distance with the aid of binoculars or other vision-enhancing devices. shoulder surfing is a type of social engineering technique used to obtain information such as personal identification numbers (PINs), passwords and other confidential data by looking over the victim's shoulder. This attack can be performed either at close range (by directly looking over the victim's shoulder). To implement this technique attackers do not require any technical skills; keen observation of victims' surroundings and the typing pattern is sufficient. Crowded places are the more likely areas for an attacker to shoulder surf the victim.

However, the advent of modern-day technologies like hidden cameras and secret microphones makes shoulder surfing easier and gives more scope for the attacker to perform long range shoulder surfing. A hidden camera allows the attacker to capture whole login process and other confidential data of the victim, which ultimately could lead to financial loss or identity theft.

Illusion PIN Generation:

Illusion PIN is a PIN-based authentication scheme for touch screen devices which offers shoulder-surfing resistance. The design of Illusion PIN is based on the simple observation that the user is always viewing the screen of her device from a smaller distance than a shoulder-surfer. Based on this, the core idea of Illusion PIN is to make the keypad on the touch screen to be interpreted with a different digit ordering when the viewing distance is adequately large. This way, when the shoulder surfer is standing far enough, he is viewing the keypad as being different from the one that the user is utilizing for her authentication, and consequently he is unable to extract the user's PIN. Also, the keypad is shuffled in every authentication attempt (or every digit entry) to avoid disclosing the spatial distribution of the pressed digits. We create the keypad of Illusion PIN with the method of hybrid images and are call a hybrid keypad.



Visibility Algorithm

The visibility algorithm receives as inputs a hybrid keypad I and a viewing position N in the 3D space. It returns a binary prediction on whether the user's keypad of I is visible to an observer who is in position N. We use this prediction either to estimate the minimum safety distance that corresponds to a given hybrid keypad, or to create a hybrid keypad that respects a given safety distance. Algorithm 1 provides the pseudocode of the visibility algorithm.

- **Distance-As-Filtering**

  In the first step of the visibility algorithm, we simulate the way I is perceived from the viewing position N by using the distance-as-filtering. The distance-as-filtering hypothesis states that we can simulate the way an image is perceived from a particular viewing distance by filtering the image with an appropriate low-pass filter. Every observer was looking directly at an image and his viewing position was completely defined by his viewing distance. Perception of an image depends on the visual angle that it subtends, no matter where the observer stands.

- **Visibility Index:**

  The visibility index is the cornerstone of our algorithm and we would like to clarify its behavior and the intuition behind it. The visibility index is the mean value of the 10 MSSIM index values from the pairs of corresponding buttons.

- **Threshold Value of the Visibility Index**

  Based on the aforementioned remarks, it is able to marginally recognize the digits of a user's keypad. Then, the visibility algorithm calculates the visibility index v for the inputs I and N, and compares it with vth. If v >=vth, we predict that the user's keypad cannot be interpreted by the observer. If v <= vth, we predict that the observer is able to interpret the digits of the user's keypad

Performance Evaluation

Here developed an algorithm to estimate whether or not the user's keypad is visible to an observer at a given viewing position. Proposed method tested the estimated visibility of Illusion PIN through a user study of simulated shoulder-surfing attacks on smartphone devices.

CHAPTER 6
CONCLUSION AND FUTURE WORK

The main goal of our work was to design a PIN-based authentication scheme that would be resistant against shoulder surfing attacks. To this end, we created Illusion PIN. We quantified the level of resistance against shoulder-surfing by introducing the notion of safety distance, which we estimated with a visibility algorithm. In the context of the visibility algorithm, we had to model at a basic level how the human visual system works. Illusion PIN is a Hybrid PIN-based authentication scheme that would be resistant against shoulder surfing attacks. Two keypads are blended in a single key digit that will show different key pad to the attacker. Illusion PIN gives best results when compared to other PIN Authentication scheme.

In future work, we can create this Illusion PIN for android application. This will help to improve mobile security.

APPENDIX 1

SAMPLE SOURCE CODE

Codings

Admin

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package virtualkeyboard.gui;

import javax.swing.JOptionPane;

/**
 *
 * @author Admin
 */
public class Admin extends javax.swing.JFrame {

    /**
     * Creates new form Admin
     */
    public Admin() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jPasswordField1 = new javax.swing.JPasswordField();
        jLabel3 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Administrator");

        jLabel1.setText("Password");
```

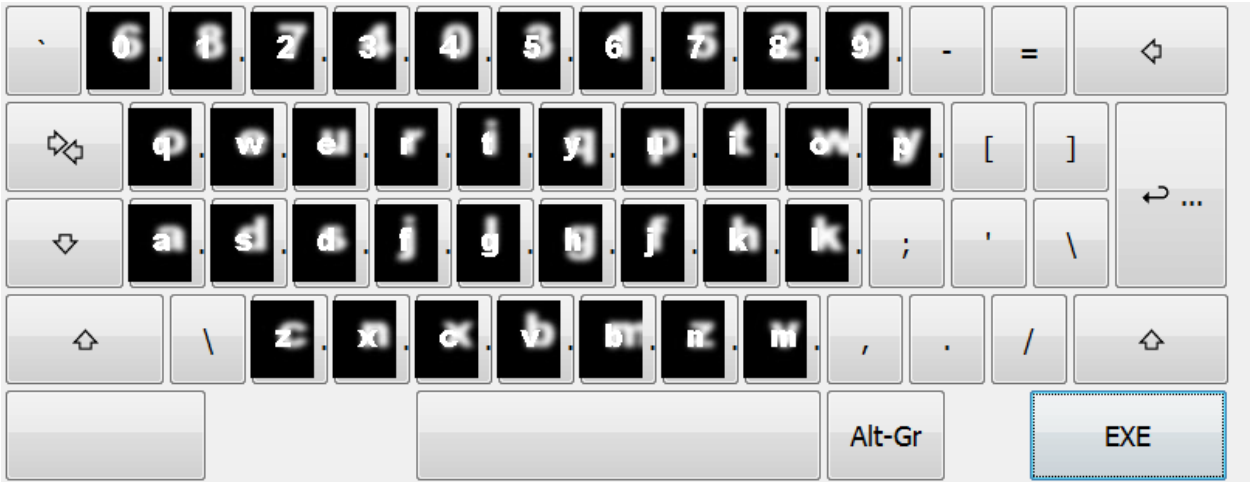
```
        jLabel2.setText("UserName ");

        jLabel3.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/virtualkeyboard/images/boots.png")));       //
NOI18N

        jButton1.setText("Submit");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                        .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 101,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jTextField1,   javax.swing.GroupLayout.PREFERRED_SIZE,
164, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(85, 85, 85)
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 101,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jPasswordField1,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    164,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addGap(64, 64, 64))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGap(0, 0, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(jLabel3)
                        .addGap(103, 103, 103))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(jButton1)
                        .addGap(117, 117, 117))))
```

```
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGap(28, 28, 28)
                .addComponent(jLabel3,      javax.swing.GroupLayout.PREFERRED_SIZE,      61,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,    16,
Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextField1,       javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2,    javax.swing.GroupLayout.PREFERRED_SIZE,    32,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1,    javax.swing.GroupLayout.PREFERRED_SIZE,    32,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jPasswordField1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(30, 30, 30)
                .addComponent(jButton1)
                .addGap(66, 66, 66))
        );

        pack();
    }// </editor-fold>

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

        String uname = jTextField1.getText().trim();
        String pass = jPasswordField1.getText().trim();
        if(uname.equals("admin") && pass.equals("admin"))
        {
            new New_User().setVisible(true);
        }
        else
        {
            JOptionPane.showMessageDialog(rootPane, "Invalid UserName or Password");
        }
    }

    /**
     * @param args the command line arguments
     */
  /*  public static void main(String args[]) {
```

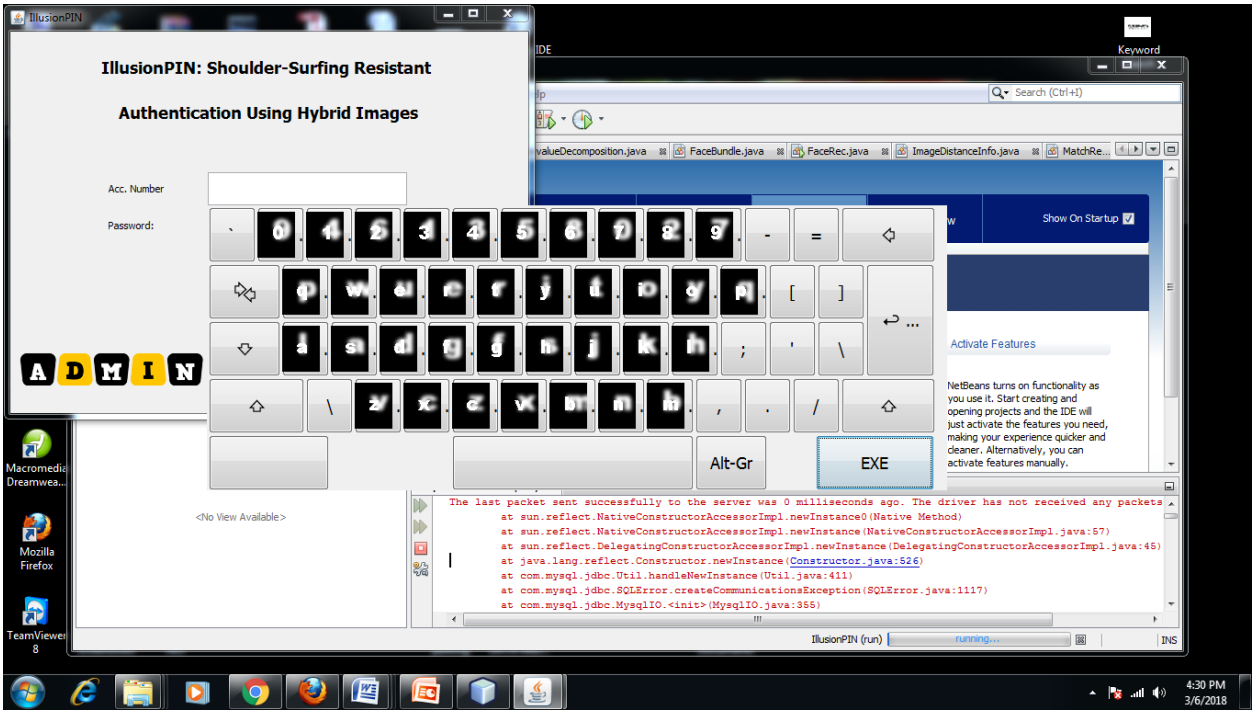
```
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Admin().setVisible(true);
        }
    });
}*/

// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JPasswordField jPasswordField1;
private javax.swing.JTextField jTextField1;
// End of variables declaration
```

APPENDIX 2

SAMPLE SCREENSHOTS

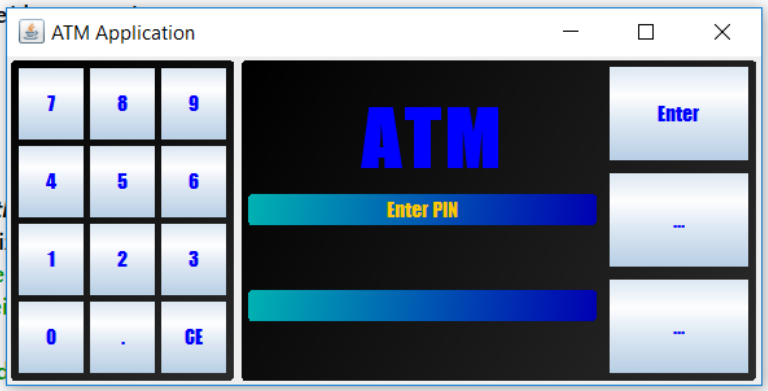


S-1: Illusion PIN



S-2: Account Login

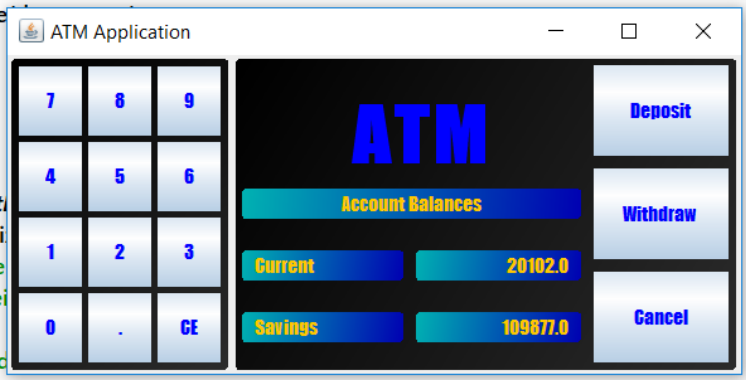
```
public ATMApplication(String datasource) {
    try {
        init(datasource);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void init(String datasource) throws Exception {
    ATMFrame frame = new ATMFrame
    if (packFrame) {
        frame.pack();
    } else {
        frame.validate();
    }
    Dimension screenSize = Toolkit.geti
    Dimension frameSize = frame.getSi
    if (frameSize.height > screenSize.he
        frameSize.height = screenSize.hei
    }
    if (frameSize.width > screenSize.wid
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
```



S-3: Create ATM frame

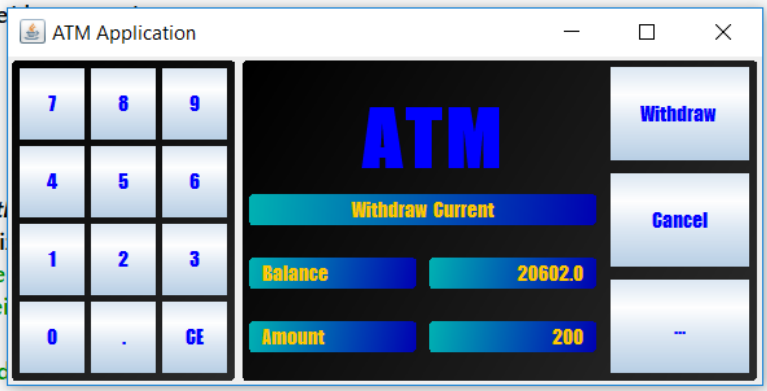
.

```
public ATMApplication(String datasource) {
    try {
        init(datasource);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void init(String datasource) throws Exception {
    ATMFrame frame = new ATMFrame
    if (packFrame) {
        frame.pack();
    } else {
        frame.validate();
    }
    Dimension screenSize = Toolkit.geti
    Dimension frameSize = frame.getSi
    if (frameSize.height > screenSize.he
        frameSize.height = screenSize.hei
    }
    if (frameSize.width > screenSize.wid
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
```



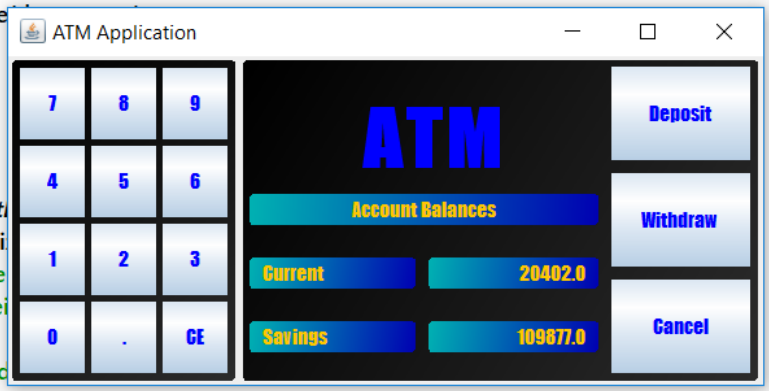
S-4: Display Main Menu

```
public ATMApplication(String datasource) {
    try {
        init(datasource);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void init(String datasource) throws Exception {
    ATMFrame frame = new ATMFrame
    if (packFrame) {
        frame.pack();
    } else {
        frame.validate();
    }
    Dimension screenSize = Toolkit.geti
    Dimension frameSize = frame.getSi
    if (frameSize.height > screenSize.he
        frameSize.height = screenSize.hei
    }
    if (frameSize.width > screenSize.wid
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
```



S-5: Withdrawal

```
public ATMApplication(String datasource) {
    try {
        init(datasource);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void init(String datasource) throws Exception {
    ATMFrame frame = new ATMFrame
    if (packFrame) {
        frame.pack();
    } else {
        frame.validate();
    }
    Dimension screenSize = Toolkit.geti
    Dimension frameSize = frame.getSi
    if (frameSize.height > screenSize.he
        frameSize.height = screenSize.hei
    }
    if (frameSize.width > screenSize.wid
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
```



S-6: Checking Balance

REFERENCES

[1]Graphical Password Authentication: Implementation and Evaluation of Personalized Persuasive Cued Click Points -Asher D'Mello Rohan Bagwe Victor Fernandes Ankita Karia.

[2]Biometric Online Signature Verification-Fincy Francis1, Aparna M.S, Anitta Vincent.

[3]DRAW-A-PIN: Authentication using finger-drawn PIN on touch devices - Toan Van Nguyen a, Napa Sae-Baeb, Nasir Memon.

[4]Skull Conduct: Biometric User Identification on Eyewear Computers Using Bone Conduction Through the Skull - Stefan Schneegass, Youssef Oualil, Andreas Bulling

[5]PIN methodology: Touch screen Mobile Authentication Using Multi-Touch Sequential Gestures-Toan Van Nguyen a, Napa Sae-Baeb, Nasir Memon

[6]A. J. Aviv, K. Gibson, E. Mossop, M. Blaze and J. M. Smith, "Smudge attacks on smart phone touch screens."

[7]T. Pering, M. Sundar, J. Light, and R. Want, "Photographic authentication through untrusted terminals."

[8] A. Paivio, T. Rogers, and P. Smythe, "Why are pictures easier to recall than words?" Psychonomic Sci., vol. PS-11, pp. 137– 138, 1968.

[9] D. Nelson, U. Reed, and J. Walling, "Picture superiority effect," J. Exp. Psychol.: Human Learn. Memory, vol. 3, pp. 485–497, 1977.

[10] S. Brostoff and M. Sasse, "Are passfaces more usable than passwords? A field trial investigation," People Comput. XIV – Usability or Else!: Proc. HCI 2000., pp. 405–424, 2000.

[11] A. De Angeli, M. Coutts, L. Coventry, G. Johnson, D. Cameron, and M. Fischer, "Vip: A visual approach to user authentication," in Proc. Working Conf. Adv. Vis. Interfaces, 2002, pp. 316–323.

[12] B. Ives, K. Walsh, and H. Schneider, "The domino effect of password reuse," Commun. ACM, vol. 47, no. 4, pp. 75–78, 2004.

[13] J. Long and K. Mitnick, No Tech Hacking: A Guide to Social Engineering, Dumpster Diving, and Shoulder Surfing. Amsterdam, The Netherlands: Elsevier, 2011.

[14] T. Kwon, S. Shin, and S. Na, “Covert attentional shoulder surfing: Human adversaries are more powerful than expected,” IEEE Trans. Syst., Man, Cybern.: Syst., vol. 44, no. 6, pp. 716–727, Jun. 2014.

[15] (2014). Google glass snoopers can steal your passcode with a glance [Online]. Available: http://www.wired.com/2014/06/ google-glass-snoopers-can-steal-your-passcode-with-a-glance/ [16] M. Sasse, S. Brostoff, and D. Weirich, “Transforming the weakest linka human/computer interaction approach to usable and effective security,” BT Technol. J., vol. 19, no. 3, pp. 122–131, 2001.

[17] (2015). Mobile marketing statistics compilation [Online]. Available: http://www.smartinsights.com/mobile-marketing/mobilemarketing-analytics/mob ile-marketing-statistics/

[18] D. Hong, S. Man, B. Hawes, and M. Mathews, “A password scheme strongly resistant to spyware,” in Proc. Int. Conf. Security Manag., 2004, pp. 94–100.

[19] D. Tan, P. Keyani, and M. Czerwinski, “Spy-resistant keyboard: Towards more secure password entry on publicly observable touch screens,” in Proc. OZCHI-Comput.-Human Interaction Special Interest Group Australia, 2005, pp. 1–10.

[20] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, “Reducing shoulder-surfing by using gaze-based password entry,” in Proc. 3rd Symp. Usable Privacy Security, 2007, pp. 13–19.

[21] H. Zhao and X. Li, “S3pas: A scalable shoulder-surfing resistant textual-graphical password authentication scheme,” in Proc. 21st Int. Conf. Adv. Inform. Netw. Appl. Workshops, 2007, vol. 2, pp. 467–472.

[22] X. Bai, W. Gu, S. Chellappan, X. Wang, D. Xuan, and B. Ma, “Pas: Predicate-based authentication services against powerful passive adversaries,” in Proc. Annu. Comput. Security Appl. Conf., 2008, pp. 433–442.

[23] Z. Zheng, X. Liu, L. Yin, and Z. Liu, “A stroke-based textual password authentication scheme,” in Proc. 1st Int. Workshop Edu. Technol. Comput. Sci., 2009, vol. 3, pp. 90–95.

[24] L. Wang, X. Chang, Z. Ren, H. Gao, X. Liu, and U. Aickelin, “Against spyware using captcha in graphical password scheme,” in Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl., 2010, pp. 760–767.

[25] D. Kim, P. Dunphy, P. Briggs, J. Hook, J. Nicholson, J. Nicholson, and P. Olivier, “Multi-touch authentication on tabletops,” in Proc. 28th Int. Conf. Human Factors Comput. Syst., 2010, pp. 1093–1102.

[26] (2014). Black hat: Google glass can steal your passcodes [Online]. Available: https://www.technologyreview.com/s/529896/blackhat-google-glass-can-steal-y our-passcodes/