Virtual Key and Access Sharing

[1] Lekhana S V; [2] Prateek Shantharama

[1] Computer Science Department, Channabasaweshwara Institute Of Technology, Grad Student
Tumkur, Karnataka, 572103, India

[2] Computer Engineering (ECEE Dept.), Arizona State University, Ph.D., Student
Tempe, Arizona, 85281, USA

Abstract - In these recent times, security and ease of access are the major factors for any real-time application. Locks and keys are a part of our daily routine and the technology used here is outdated and obsolete. We intend to enhance this feature by bringing in technology to provide better security and ease of access. This can be accomplished by virtualizing the keys used in our daily lives and by implementing new locking mechanism. Virtual key is an idea, which involves storing keys in digital format in a mobile phone and accessing a lock via wireless transmission. This project is implemented by storing keys on an android device in digital format and the keys can be accessed by using an android application in which many features such as sharing of keys with trusted parties and locking/unlocking of a lock can be done. The locking mechanism is implemented using a Renesas Microcontroller, which interacts with a magnetic relay to lock/unlock the electromagnet. The communication with the android device is accomplished using a Bluetooth shield to obtain the keys. Many features such as adding new keys, sharing of keys with trusted parties is also provided. The sharing of keys is encrypted and hence there are no security concerns. In addition, the keys can be shared for lifetime or for a specified number of times.

Keywords: *Virtualizing, Renessa, key sharing, Bluetooth.*

1. Introduction

1.1 Introduction to the Project

Smartphones have become a necessity in today's life and almost all are using one. Statistics show that a person is less likely to forget his/her phone than his/her car, home, office keys. New generation smart phones are replacing many things, which we use in our daily life. One of the intentions behind this proposal is to uphold and popularize technology in all domains. Our idea here is to replace traditional keys with virtual ones and allow the keys to be shared with trusted parties for specified count or for lifetime. Locks can be opened just by opening the phone app and clicking on unlock button. Our proposal includes two parts one is phone app and other is the hardware for the host such as car, door etc. The hardware includes a Bluetooth shield and a microcontroller. Bluetooth shield fetches the "key" that is the encrypted string from the smart phone and sends it to the microcontroller. Microcontroller decodes the string and compares it with a predetermined unique string that is stored on its memory by the manufacturer. Upon successful matching, the controller sends a signal to the door opening mechanism.

The app is simple and can be designed for all major smartphone operating systems such as Android, Windows, iOS etc. The app is password protected, which ensures the security of the host device in the case the phone is stolen or is lost. The app will have modules to send, receive and open the virtual key. In a case, we have to send the virtual key to a trusted user we have to specify the count for access. The key can be shared via SMS service. The received keys will not be allowed for duplication or sharing. After the specified access count gets over, the key will be deleted automatically from the phone. If more use is needed then a request has to be sent to the owner for approval. The phone will transmit the key only after receiving the acknowledgement regarding the authenticity of the hardware to which it is transmitting. The app will not transmit the key if the requesting device is anything other than the device built for this purpose.

Bluetooth

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices and building personal area networks (PANs). Invented by telecom vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables. It can connect several devices, overcoming problems of synchronization. Bluetooth is managed by the Bluetooth Special Interest Group (SIG), which has more than 20,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. Bluetooth was standardized as IEEE 802.15.1, but the standard is no longer maintained. The SIG oversees the development of the specification, manages the qualification program, and protects the trademarks. To be marketed as a Bluetooth device, it must be qualified to standards defined by the SIG. A network of patents is required to implement the technology, which is licensed only for that qualifying device

Android

Android is an operating system based on the Linux kernel with a user interface based on direct manipulation, designed primarily for touchscreen mobile devices such as smartphones and tablet computers, using touch inputs, that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in televisions, gaming consoles, digital cameras, and other electronics. As of 2011, Android has the largest installed base of any mobile OS and as of 2013 its devices also sell more than Windows, iOS and Mac OS devices combined. As of July 2013 the Google Play store has had over 1 million Android apps published, and over 50 billion apps downloaded. A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially released running other operating systems.

Renessa

The Renessa Electronics RL78 is a 16-bit CPU core with CISC architectureForEmbedded microcontrollers de veloped and manufactured by Renessa Electronics and introduced in 2011. The RL78 was the first new much core to emerge from the new Renessa Electronics Company from the merger of NEC Electronics and Renessa Technology and incorporated the features of the NEC K0 and Renessa Technology R8C microcontrollers. The RL78 was developed to address extremely low power but highly integrated microcontroller applications, to this end the core offered a novel low power mode of operation called "snooze mode" where the ADC or serial interface can be programmed to meet specific conditions to wake the device from the extreme low power STOP mode of 0.52uA.

1.2 Problem Statement

To come up with a system that replaces traditional keys with the virtual ones, which can be stored as encrypted alphanumeric characters on hand, held devices. Locking and unlocking must be established by communication via Bluetooth. Another important aim of the system is to share the keys with trusted third party for lifetime, specific number of counts. Importance is also given for the security of the product. Security must not be compromised at any cost.

1.3 Objective of the Project

Our objective here is to replace traditional keys with virtual ones and also allow them to share the keys with trusted users for specified count or for lifetime. Locks can be opened just by opening the phone app and clicking on unlock button.

1.4 Scope of the Project

The proposed system can be utilized in any system where a lock is required like cars, home, offices, hotels etc. Security is not compromised in any cost. Hence, the application can be completely trusted. This model is very useful in a college scenario. Assume that all the locks in the college are virtualized according to our proposed system. Then, the owner of all the lab keys can be the H.O.D of the department. He can then share those keys with the lab attenders for lifetime use. He can also share the keys with the students when required for a specified number of times. This completely removes the hassle of physical keys, which can't be duplicated as and when required for sharing with trusted parties without any compromise in security.

2. Literature Survey

For cars, the concept of unlocking the car using the NFC ring is still in developing stage. Here the security of the car is compromised when the ring is stolen or lost. Another possibility is that the information in the NFC ring which is basically a type of NFC tag can be read by all smart phones which will also risk the security of the car. For home doors a product called Lockton is available which unlocks home doors and this is compatible only with Philips made locks and not with any other locks. And this doesn't support cars or any other locks. This product allows sharing but here the company server is involved is giving and getting back the access which may not be reliable. Getting back the shared access involves requesting the company server to make the requested operation which may consume time. Also if the server crashes, your keys will become inaccessible and one cannot unlock the locks. This system doesn't involve any company server or employees to supervise the flow, which reduces the capital in a huge way.

Even though our phone gets lost or stolen we may always get the access from the manufacturers of the car or doors via E-mail. Here the data is encrypted so any person from the manufacturing company cannot know the code of the lock. Every Bluetooth lock will have a unique code which protects the integrity of the system. Multiple ownership of the lock is also possible. It means if we want we can also share the key permanently with required and trusted users. In case when the phone battery is dead, we can receive keys from E-mail from the co-owner for a preferably less time in a borrowed mobile. The borrowed mobile can be returned to the owner after unlocking the car without having fear of the mobile owner might get access to the lock. Keeping number of access time in minds can also design the app. The key can be shared based on a preferred number of accesses. It will get deleted after the number of times the access has been made. So the system is flexible and can be designed to suite any purpose.

3. System Requirement Specification

3.1 Requirements overview

The main feature of the system is to lock and unlock the lock interface using android device. Another important facility provided to the user is to share the key with trusted third party. Facility is also made to view all the sent keys, received keys and the owner's keys.

The User can

- Lock/Unlock the lock interface.
- Share the keys for specified number of counts.
- Share the key for lifetime.
- View/Edit all the keys.

3.1.1 Functional Requirements

3.1.1.1 Lock and unlock the Lock interface

Description and Priority
The User can lock or unlock the lock interface via Bluetooth communication. This system feature is of high priority as it is required for every time when the User locks or unlocks the lock. Avoid duplication of entries while making a new entry to prevent memory consumption.

Stimulus/Response Sequences
Whenever the user presses the lock/unlock button for a certain key, the state of the lock interface is read. If the lock is locked, unlock signal is sent similarly if the lock is not locked, a lock signal is sent. As soon as the user presses the button the application will ask for turn on the Bluetooth to open the lock. Then the key sent to the Bluetooth shield and the lock is opened.

Functional Requirements
There must not be any delay while transferring the key to the Bluetooth shield. Permission has to be obtained to gain the Bluetooth access. The key has to be decrypted properly before transferring to the shield.

3.1.1.2 Share the key

Description and Priority
Whenever the user presses the lock/unlock button for a certain key, the key has to be sent to the specified third party. This can be implemented via short messaging service that contains the encrypted key. This system feature is of high priority as it is required for every time when the user shares the key to trusted third party.

Stimulus/Response Sequences
When the user clicks the share button for a certain key, the application has to show the contact list from the cell phone and should allow the user to select a single entry from the contact list. Once the name is selected from the contact list, a SMS is sent to that particular number which contains the encrypted key.

Functional Requirements
If the contact list is empty, suitable error message has to be displayed. The key has to be encrypted properly before it sends the key to the trusted third party. Suitable error message has to be displayed if any interruption or error occurred.

3.1.1.3 View all the keys

Description and Priority
The system enables the user to view all the keys that present in the phone. The user can view all the sent keys, received keys and his own keys. This system feature is of low priority as it is required for every time when the user views the keys.

Stimulus/Response Sequences
When the user click view keys button a web view has to open with three distinct tabs labeled sent key, received key and my keys. All the keys have to be displayed in the list view.

Functional Requirements
Suitable error message has to be displayed if any interruption or error occurred. The web view has to display clearly which can be seen from cell phones of the all sizes.

3.1.2 Non-Functional Requirements

Performance Requirements
The design or layout of every form will be very clear and very interactive to the user. When the user opens the software, the login window will appear. In the screen layout the background color is very light and the graphics and font style will be in proper manner and well organized. In each and every window there will be alert, confirm etc. message box for displaying message. This software will be easily understandable and operable by the user.

Safety Requirements
The system provides authentication to product in the beginning itself, only authorized personnel can be able to use the product. If any damage occurs to the product, we can restore all the information from the copy of all materials are preserved in another storage device. Hence user need not back up the product at regular intervals.

Security Requirements
The system provides authentication to product in the beginning itself, only authorized personnel is able to use the product through login interface. A strong encryption technique has to be applied.

3.1.3 Behavioral Requirements

Software Quality Attributes
SQA of the products are:

- Flexibility
- Interoperability
- Maintainability
- Robustness
- Usability

3.1.4 User Requirements

User Interfaces
The design or layout of every form will be very clear and very interactive to the user. When the user open the software the welcome window will appear. Form each and every window the user can easily go to any desired window that is, there will be an absolute & relative referencing. In the screen layout the background color is very light and the graphics and font style will be in proper manner and well organized.

Hardware Interfaces
The standard input and output will be used as the software's interface with the user. It is assumed that the smart phone will have monitor, virtual keyboard as its standard input and output devices.

Software Interfaces
The operating system required is android OS 4.2 or above. It must have Bluetooth feature to open lock.

3.2 Software Requirements and Hardware Requirements

3.2.1 Software Requirements

- Operating System: Android 4.2 or higher.

3.2.2 Hardware Requirements

- Bluetooth shield
- Magnetic lock interface
- Renessa 30 pin microcontroller
- Bluetooth enabled smart phone with android 4.2 or higher version

4. SYSTEM ANALYSIS

4.1 Existing System

Existing system involves physical keys to lock or unlock the lock mechanism. There will be a separate physical

IJCSN - International Journal of Computer Science and Network, Volume 7, Issue 4, August 2018
ISSN (Online) : 2277-5420
www.IJCSN.org
Impact Factor: 1.5

key for every lock interface. With this system, user requires to carry every keys of the lock mechanism that he/she come across. If the user needs to share the keys with trusted third party, he/she needs to physically share the key to the other user.

4.2 Proposed Systems

Virtual key and access sharing is a system that replaces traditional keys with virtual ones which are stored in a android device in the form of secret text. While the android device is near the lock interface, communication is occurs between the lock interface and the android device that consists of key and there by provides the access to the lock. The main feature of the system is providing the facility of access sharing to the trusted people. The user can send and receive the keys by other by sending the key to third party via short message service.

4.3 Software process model employed

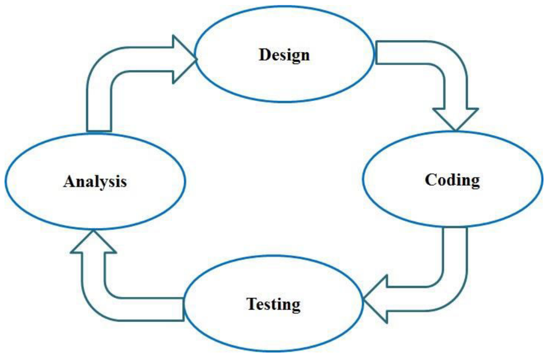


Fig 4.3 Project Development Model

The software development life cycle is a process involving four stages in the development of the software product as shown in the fig 4.3.

Planning is an objective of each and every activity, where we want to discover things that belong to the project. An important task in creating a software programisextracting the requirements or requirements analysis. Implementation is the part of the process where software engineers actually program the code for the project. Software testing is an integral and important phase of the software development process. This part of the process ensures that defects are recognized as soon as possible. Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the writing of an API, be it external or internal.

5. System Design

5.1 Design Specific Activities

5.1.1 Data Design

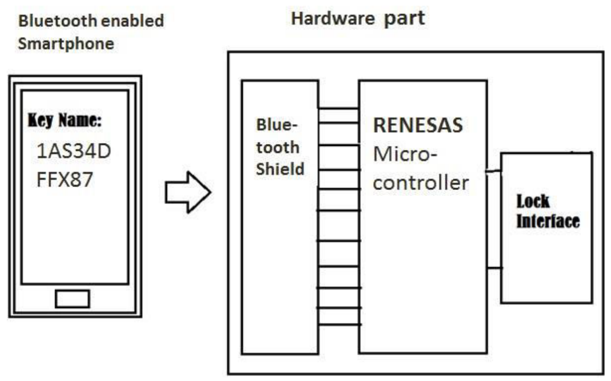


Fig 5.1 System Architecture

LOCK/UNLOCK:

- The key which is stored in the form of alphanumeric character is sent to Bluetooth shield from the mobile phone.
- It is then transmitted to microcontroller which authenticates and locks or unlocks the lock interface.
- For this, the microcontroller is programmed using embedded c.

SHARE KEYS:

- Key is encrypted by DES encryption technique and sends to third party via SMS.
- As he/she receives the SMS, application is automatically invoked.
- SMS that contains key is automatically deleted form the inbox.

5.1.2 Architectural Design

Renessa Microcontroller:

- 30-pin plastic SSOP (7.62 mm (300))

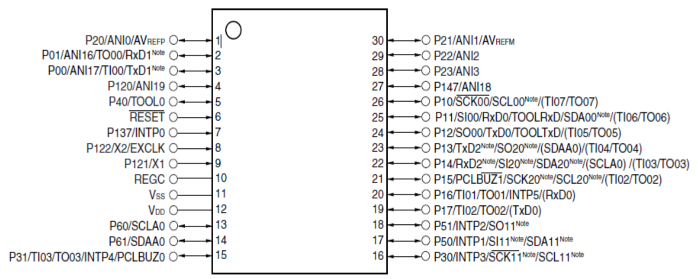

P20/ANI0/AVREFP 1
P01/ANI16/TO00/RxD1Note 2
P00/ANI17/TI00/TxD1Note 3
P120/ANI19 4
P40/TOOL0 5
RESET 6
P137/INTP0 7
P122/X2/EXCLK 8
P121/X1 9
REGC 10
VSS 11
VDD 12
P60/SCLA0 13
P61/SDAA0 14
P31/TI03/TO03/INTP4/PCLBUZ0 15
30 P21/ANI1/AVREFM
29 P22/ANI2
28 P23/ANI3
27 P147/ANI18
26 P10/SCK00/SCL00Note/(TI07/TO07)
25 P11/SI00/RxD0/TOOLRxD/SDA00Note/(TI06/TO06)
24 P12/SO00/TxD0/TOOLTxD/(TI05/TO05)
23 P13/TxD2Note/SO20Note/(SDAA0)/(TI04/TO04)
22 P14/RxD2Note/SI20Note/SDA20Note/(SCLA0) /(TI03/TO03)
21 P15/PCLBUZ1/SCK20Note/SCL20Note/(TI02/TO02)
20 P16/TI01/TO01/INTP5/(RxD0)
19 P17/TI02/TO02/(TxD0)
18 P51/INTP2/SO11Note
17 P50/INTP1/SI11Note/SDA11Note
16 P30/INTP3/SCK11Note/SCL11Note

Fig. 5.2 Pin configuration

1.5.3 30-pin products

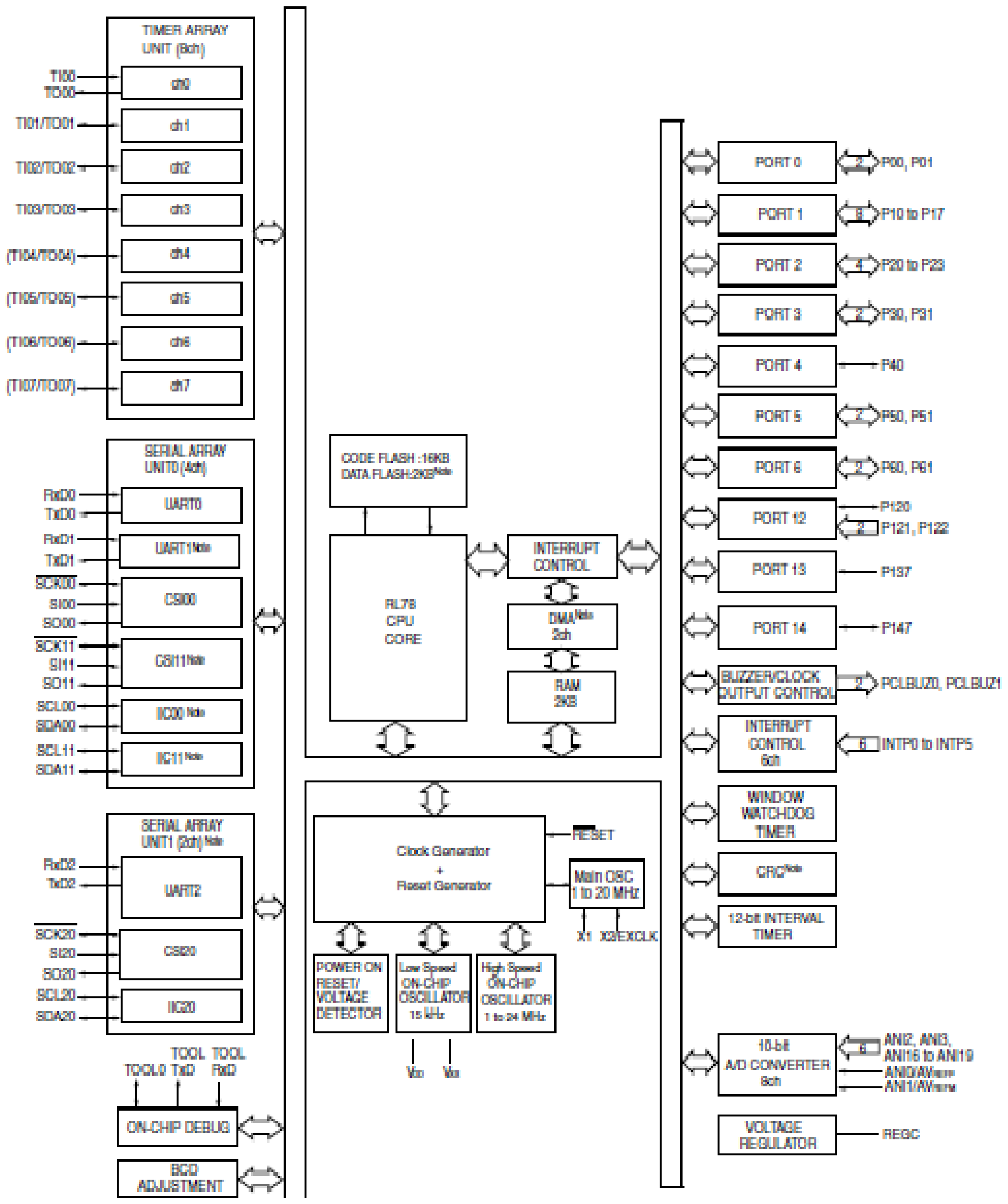


Fig. 5.3 Architecture of Renessa.

A/D CONVERTER

The A/D converter is a 10-bit resolution Note converter that converts analog input signals into digital values, and is configured to control analog inputs, including up to 11 channels of A/D converter analog inputs (ANI0 to ANI3 and ANI16 to ANI22).

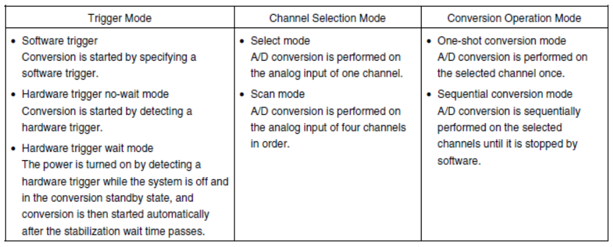
The A/D converter has the following function.

• **10-bit resolution A/D conversion Note**

10-bit resolution A/D conversion is carried out repeatedly for one analog input channel selected from ANI0 to ANI3 and ANI16 to ANI22 (ANI0 to ANI3, ANI16 to ANI19 for 30-pin products). Each time an A/D conversion operation ends, an interrupt request (INTAD) is generated (when in the select mode).

Note 8-bit resolution can also be selected by using the ADTYP bit of A/D converter mode register 2 (ADM2). Various A/D conversion modes can be specified by using the mode combinations below.

Table 5.1 A/D Converter Modes



Trigger Mode	Channel Selection Mode	Conversion Operation Mode
• Software trigger Conversion is started by specifying a software trigger. • Hardware trigger no-wait mode Conversion is started by detecting a hardware trigger. • Hardware trigger wait mode The power is turned on by detecting a hardware trigger while the system is off and in the conversion standby state, and conversion is then started automatically after the stabilization wait time passes.	• Select mode A/D conversion is performed on the analog input of one channel. • Scan mode A/D conversion is performed on the analog input of four channels in order.	• One-shot conversion mode A/D conversion is performed on the selected channel once. • Sequential conversion mode A/D conversion is sequentially performed on the selected channels until it is stopped by software.

Operation of UART (UART0 to UART2) Communication

This is a start-stop synchronization function using two lines: serial data transmission (TXD) and serial data reception (RXD) lines. By using these two communication lines, each data frame, which consists of a start bit, data, parity bit, and stop bit, is transferred asynchronously (using the internal baud rate) between the microcontroller and the other communication party. Full-duplex UART communication can be performed by using a channel dedicated to transmission (an even-numbered channel) and a channel dedicated to reception (an odd-numbered channel).

Data transmission/reception

- Data length of 7, 8, or 9 bits (Only UART0 can be specified for the 9-bit data length)
- Select the MSB/LSB first
- Level setting of transmit/receive data and select of reverse (selecting whether to reverse the level)
- Parity bit appending and parity check functions
- Stop bit appending and stop bit check functions

Interrupt function

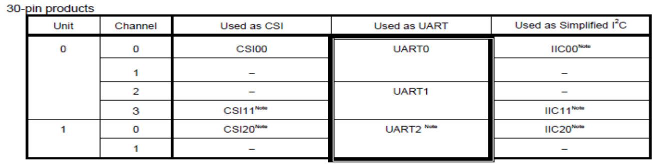
- Transfer end interrupt/buffer empty interrupt
- Error interrupt in case of framing error, parity error, or overrun error

Error detection flag

- Framing error, parity error, or overrun error

UART0 is compatible with SNOOZE mode, when RxD0 pin input is detected while in the STOP mode, the SNOOZE mode makes data reception that does not require the CPU possible.

Table 5.2 30 Pin Products



30-pin products

Unit	Channel	Used as CSI	Used as UART	Used as Simplified I^2C
0	0	CSI00	UART0	IIC00^{Note}
	1	–		–
	2	–	UART1	–
	3	CSI11^{Note}		IIC11^{Note}
1	0	CSI20^{Note}	UART2 ^{Note}	IIC20^{Note}
	1	–		–

Android Architecture

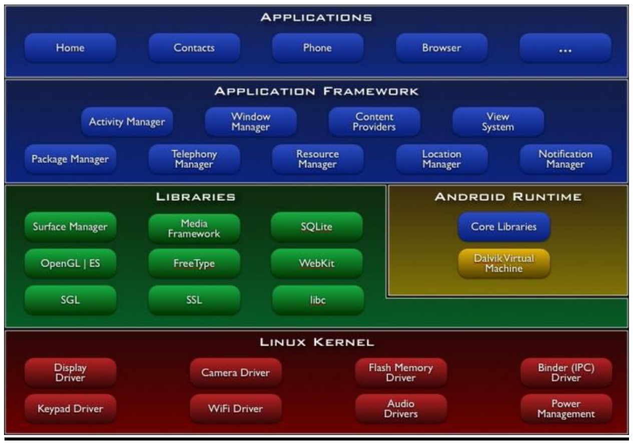


Fig 5.4 Android architecture.

Applications

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more. Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data.
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files.
- A Notification Manager that enables all applications to display custom alerts in the status bar.

- An Activity Manager that manages the lifecycle of applications and provides a common navigation back stack.

Libraries
Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:
System C library - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices.
Media Libraries - based on Packet Video's Open CORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG.
Surface Manager - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.
LibWebCore - a modern web browser engine which powers both the Android browser and an embeddable web view.
SGL - the underlying 2D graphics engine.
3D libraries - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer.
Free Type - bitmap and vector font rendering.
SQLite - a powerful and lightweight relational database engine available to all applications.

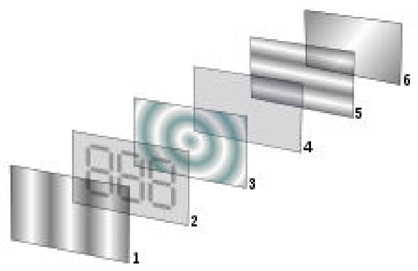
Android Runtime
Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.
Every Android application runs in its own process, with its own instance of the Dalvin virtual machine. Dalvin has been written so that a device can run multiple VMs efficiently. The Dalvin VM executes files in the Dalvin Executable (.decks) format which is optimized for minimal memory footprint. The VM is register based, and runs classes compiled by a Java language compiler that have been transformed into the .decks format by the included "dx" tool. The Dalvin VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Linux Kernel
Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

LCD Architecture:
ALPHA-NUMERIC LCD DISPLAY
A **liquid crystal display** (**LCD**) is a flat panel display, electronic visual display, based on on Liquid Crystal Technology. A liquid crystal display consists of an array of tiny segments (called pixels) that can be manipulated to present information. Liquid crystals do not emit light directly instead they use light modulating techniques. LCDs are used in a wide range of applications, including computer monitors, television, instrument panels, aircraft cockpit displays, signage, etc. They are common in consumer devices such as video players, gaming devices, clocks, watches, calculators, and telephones.
LCDs are preferred to cathode ray tube (CRT) displays in most applications because

1. The size of LCDs comes in wider varieties.
2. They do not use Phosphor; hence images are not burnt-in.
3. Safer disposal
4. Energy Efficient
5. Low Power Consumption

It is an electronically modulated optical device made up of any number of segments filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in color or monochrome



Reflective twisted nematic liquid crystal display.

1. Polarizing filter film with a vertical axis to polarize light as it enters.
2. Glass substrate with ITO electrodes. The shapes of these electrodes will determine the shapes that will appear when the LCD is turned ON. Vertical ridges etched on the surface are smooth.
3. Twisted hematic liquid crystal.
4. Glass substrate with common electrode film (ITO) with horizontal ridges to line up with the horizontal filter.
5. Polarizing filter film with a horizontal axis to block/pass light.
6. Reflective surface to send light back to viewer. (In a backlit LCD, this layer is replaced with a light source.)

JHD162A is one such LCD which is used here. It has a Panel with 2 rows and 16 column and with blocks as shown below with 5x7 pixel-selection pattern.

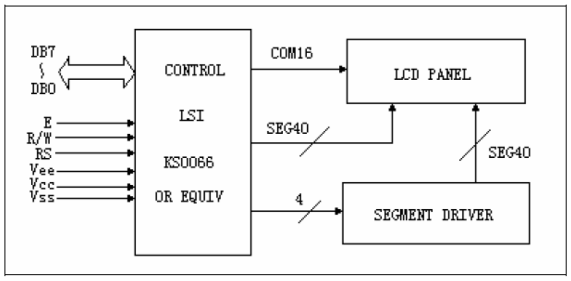


Fig 5.5 LCD Control Panel

Table 5.3 Operating Voltage

Parameter	Symbol	Testing Criteria	Standard Values			Unit
			Min.	Typ.	Max	
Supply voltage	V_{DD}-V_{SS}	-	4.5	5.0	5.5	V
Input high voltage	V_{IH}	-	2.2	-	V_{DD}	V
Input low voltage	V_{IL}	-	-0.3	-	0.6	V
Output high voltage	V_{OH}	$-I_{OH}$=02mA	2.4	-	-	V
Output low voltage	V_{OL}	I_{OL}=1.2mA	-	-	0.4	V
Operating voltage	I_{DD}	V_{DD}=5.0V	-	1.5	3.0	mA

Pin Details

Pin NO.	Symbol	Level	Description
1	VSS	0V	Ground
2	VDD	5.0V	Supply voltage for logic
3	VO	---	Input voltage for LCD
4	RS	H/L	H : Data signal, L : Instruction signal
5	R/W	H/L	H : Read mode, L : Write mode
6	E	H, H → L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	LED A(+)	4.2V	Back light anode
16	LED K (-)	0V	Back light cathode

Fig 5.6 Pin Details of LCD Display

InterfacingJHD162Awith Microcontroller

Alpha Numeric displays form an integral part of the Embedded Systems. The Microcontroller controls the Data displayed here. The Control pins like Read Strobe, Read/Write and Enable are controlled through the Microcontroller Ports as per the waveforms above.

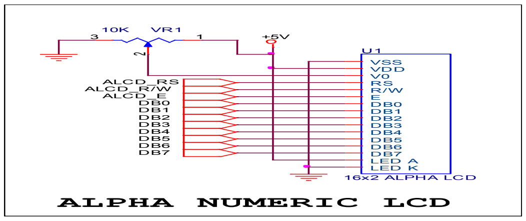


Fig 5.7 Circuit of LCD Display

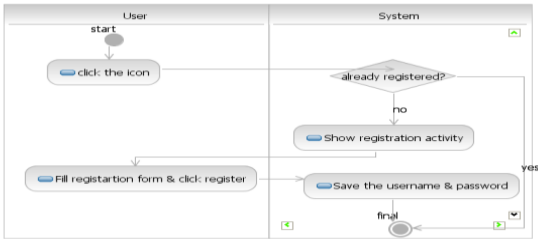

User
System
start
click the icon
already registered?
no
Show registration activity
Fill registartion form & click register
Save the username & password
yes
final

Fig 5.2.2 User Login Activity Diagram

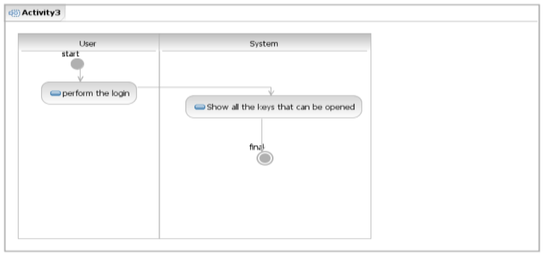


Fig 5.2.3 Open the keys Activity Diagram

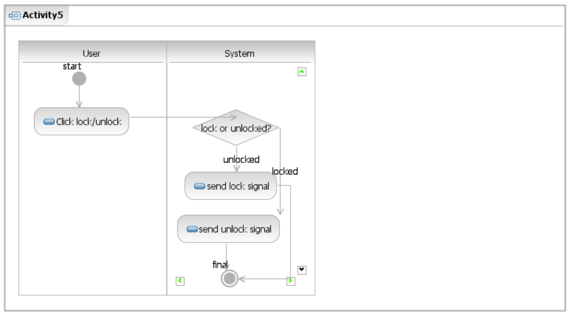


Fig 5.2.4 Lock or Unlock Activity Diagram

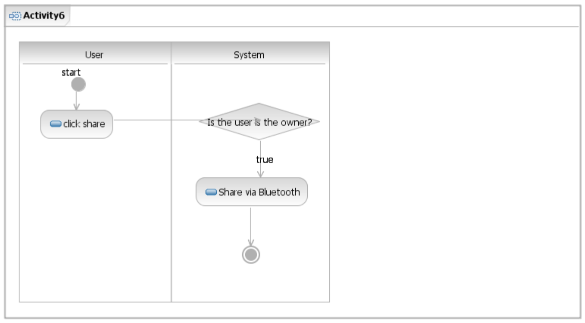


Fig 5.2.5 Share Key Activity Diagram

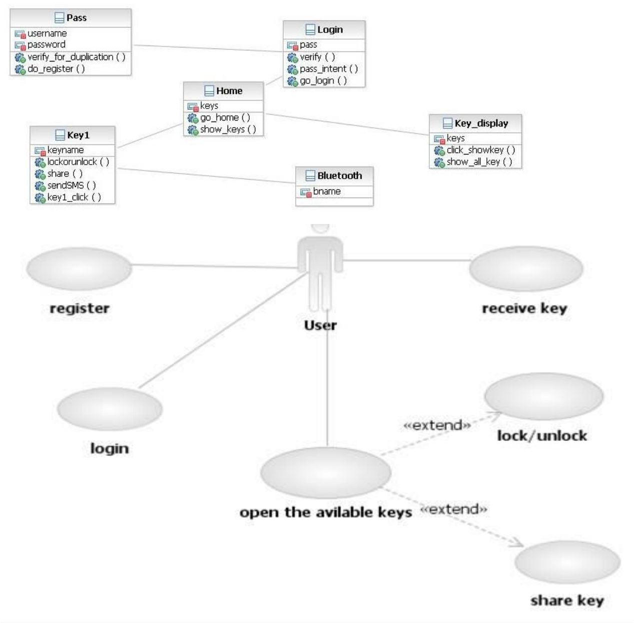


Fig 5.3.2 Use Case Diagram

6. System Implementation

6.1 Implementation Platform

Android

Google within the Open Handset Alliance (OHA) delivers a complete set of software for mobile devices: an operating system, middleware and key mobile applications. Android is built on the open Linux Kernel. bit utilizes a custom virtual machine, Dalvin virtual machine (DalvikVM) , designed and written by Dan Bornstein and some other Google engineers , that has been designed to optimize memory and hardware resources in a mobile environment.

Anatomy of an Android Application

There are **four building blocks** to an Android application:

- **Activity**
- **Intent Receiver**
- **Service**
- **Content Provider**

- Once we have decided what components we need for our application, we should list them in a file called **AndroidManifest.xml**.
- This is an **XML** file where **we declare the components** of our application and what their capabilities and requirements are.

Activity

Activities are the **most common** of the four Android building blocks. An **activity** is usually a **single screen** in our application. **Each activity** is **implemented** as a **single class** that **extends** the **Activity base class**.

Intent and Intent Filters

Android uses a **special class** called **Intent** to **move from screen to screen**. The **two** most **important parts** of the **intent** data structure are the **action** and **the data to act upon**. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, we would create an Intent with the VIEW action and the data set to a Website-URI. There is a related class called an **Intent Filter.**

While intent is effectively a request to do something, an intent filter is a **description** of what intents an activity is capable of **handling**. **Activities publish** their **Intent Filters** in the **AndroidManifest.xml** file.

To navigate forward, an activity calls **start Activity(my Intent).**

Intent Receiver

We can use an **Intent Receiver** when we want code in our application **to execute in reaction to an external event**, for example, when the phone rings, or when the data network is available. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. intent receivers are also registered in AndroidManifest.xml, but we can also register them from code using Context.registerReceiver().

Service

A **Service** is **code** that is **long-lived** and **runs without** a **UI**. A good example of this is a media player playing songs from a play list. In this case, the media player activity could start a service using **Context.startService()** to run in the background to keep the music going. We can connect to a service (and start it if it's not already running) with the **Context.bindService()** method.

Content Provider

Applications can **store** their **data** in **files**, **a SQLite database**, **preferences** or any other mechanism that makes sense. A **content provider**, however, is **useful** if you want our **application's data to be shared with other applications**. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

Android User Interfaces

User Interfaces (UI) in Android can be built within **two ways**, by **defining XML-Code** or by **writing Java-**

Code. Defining the GUI structure in XML is highly preferable.

Hierarchy of Screen Elements

The **basic functional unit** of an **Android application** is the **activity**. To give our activity a screen presence and design its UI, we work with **view** and **view groups** - basic units of user interface expression on the Android platform.

Views

A **view** is an **object** extending the base class android.view.View. It's a data structure whose properties store the layout and content for a specific rectangular area of the screen. The **View class serves** as a **base class** for **all widgets** - a set of fully implemented subclasses that draw interactive screen elements. The list of widgets available includes i.e. Text View, Edit Text, Button, Radio Button, Checkbox, Scroll View.

View groups

A **view group** is an object of class android.view.Viewgroup. A view group is a special type of view object whose function is to contain and manage a subordinate set of views and other view groups. View groups let us add structure to our UI and build up complex screen elements that can be addressed as a single entity. The **View group** class **serves** as a **base class for layouts** - a set of fully implemented subclasses that provide common types of screen layout. The layouts give us a way to build a structure for a set of views. To attach the tree to the screen for rendering, our Activity calls its setContentView() method and passes a reference to the root node object.

The AndroidManifest.xml

The AndroidManifest.xml is a required file for every Android application. It describes global values for our package, including the application components (activities, services, etc) that the package exposes to the 'outer world', what kind of data each of our Activities and co. can handle, and how they can be launched. We can also **specify permissions** in AndroidManifest.xml. Almost every AndroidManifest.xml (as well as many other Android XML files) will include the **namespace declaration (xmlns:android=http://schemas.android.com/apk/res/android)** in its first element. This makes a variety of standard Android attributes available in the file, which will be used to supply most of the data for elements in that file. Almost every manifest includes a single **<application> tag**, which itself contains several tags describing Applications, Intent Receivers, etc… that are available in this application. If we want to make an Activity launch able directly through the user (i.e., if we want to see the application on the main menu) we will need to make it support the **MAIN action** and **LAUNCHER category**.

<manifest>

This is the **root node** of each AndroisManifest.xml. It contains the **package-attribute**, which points to any package in our Activity

<uses-permission>

Describes a security permission, which our package must be granted in order for it to operate correctly. The permissions get granted by the user during installation of our application.

<permission>

Declares a security permission that can be used to restrict which applications can access components or features in our (or another) package.

<instrumentation>

Declares the code of an instrumentation component that is available to test the functionality of this or another package.

<application>

Root element containing declarations of the application-level components contained in the package. This element can also include global and/or default attributes for the application, such as a label, icon, theme, required permission, etc.

<activity>

An Activity is the primary thing for an application to interact with the user. The initial screen the user sees when launching an application is an activity, and most other screens they use will be implemented as separate activities declared with additional activity tags.

Note: Every Activity must have an <activity> tag in the manifest whether it is exposed to the world or intended for use only within its own package. If an Activity has no matching tag in the manifest, you won't be able to launch it.

<intent-filter>

Declares what kind of Intents a component supports. In addition to the various kinds of values that can be specified under this element, attributes can be given here to supply a unique label, icon, and other information for the action being described.

<action>

An action-type that the component supports.

<category>

A category-type that the component supports.

<data>

An MIME type, URI scheme, URI authority, or URI path that the component supports. You can also associate 1+ pieces of meta-data with your activity.

<meta-data>

Adds a new piece of meta data to the activity, which clients can retrieve through ComponentInfo.metaData.

<receiver>

An Intent Receiver allows an application to be told about changes to data or actions that happen, even if it is not currently running. As with the activity tag, you can optionally include 1+ <intent-filter> elements that the receiver supports or <meta-data> values, just all the same as with <activity>.

<service>

A Service is a component that can run in the background for an arbitrary amount of time. As with the activity tag, you can optionally include one or more <intent-filter> elements that the service supports or <meta-data> values; see the activity's <intent-filter> and <meta-data> descriptions for more information.

<provider>

A Content Provider is a component that manages persistent data and publishes it for access by other applications. We can also optionally attach one or more <meta-data> values.

Note: Of course all <tags> have to be </closed> or closed <directly/>.

Resources and the magic R.java

The resources of a project and the R.java are very close related.

Resources

Resources are external files (non-code files) that are used by our code and compiled into our application at build time.

Android supports a number of different kinds of resource files, including XML, PNG, and JPEG files.

The magic R.java

A project's **R.java** is an **auto-generated file** indexing all the resources of our project.

The log Cat

Android provides a much more powerful debugging feature – The **Log Cat**. The log Cat is a part of the **DDMS (Dalvin Debug Monitor service)** that provides a mechanism for collecting and viewing system debug output.

Important Layouts and View Groups

The most important Layouts, except **Linear Layout**, we will get to know closer here are:

Relative Layout (Views are placed relative to each other)

Table Layout (Uses tables like in HTML)

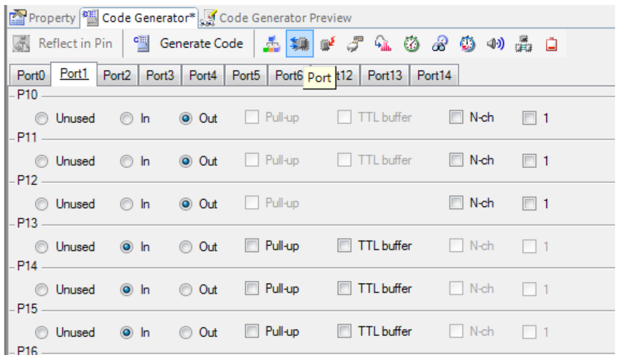
View Groups:

Grid View (similar to Table Layout, supports 'Adapters')

Scroll View (If your content exceeds the screen-size, we'll need to scroll) **Tab Host** (Displays content in tabs)

CUBE SUITE:

This is the software, which provides integrated development environment Cube Suite offers the ultimate in simplicity, usability, and security for the repetitive editing, building and debugging that typifies software development. Easy to install and operate, Cube Suite offers a highly user-friendly development environment featuring significantly shorter build times.



P10	○ Unused	○ In	● Out	☐ Pull-up	☐ TTL buffer	☐ N-ch	☐ 1
P11	○ Unused	○ In	● Out	☐ Pull-up	☐ TTL buffer	☐ N-ch	☐ 1
P12	○ Unused	○ In	● Out	☐ Pull-up		☐ N-ch	☐ 1
P13	○ Unused	● In	○ Out	☐ Pull-up	☐ TTL buffer	☐ N-ch	☐ 1
P14	○ Unused	● In	○ Out	☐ Pull-up	☐ TTL buffer	☐ N-ch	☐ 1
P15	○ Unused	● In	○ Out	☐ Pull-up	☐ TTL buffer	☐ N-ch	☐ 1
P16							

Fig 6.1.1 Port Selection in Cube Suite

6.2 Implementation Language

Embedded C is a set of language extensions for the C Programming language by the C Standards committee to address commonality issues that exist between C extensions for different embedded systems. Historically, embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations. In 2008, the C Standards Committee extended the C language to address these issues by providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as, fixed-point arithmetic, named address spaces, and basic I/O hardware addressing. Embedded C use most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements (if, switch. case), loops (while, for), functions, arrays and strings, structures and union, bit operations, macros, unions,etc.

JAVA Android

The version history of the Android mobile operating system began with the release of the Android beta in November 2007. The first commercial version, Android

1.0, was released in September 2008. Android is under ongoing development by Google and the Open Handset Alliance (OHA), and has seen Google releases a number of updates to its base operating system since its initial release Android's source code under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software. Initially developed by Android, Inc., which Google backed financially and later bought in 2005,Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

7. Testing And Result

7.1 Integration Testing

Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing system. There are three types of integration testing: Top-Down Integration: It is an incremental approach to construction of program structures. Modules are integrated by moving downwards throw the control hierarchy beginning with the main control module. Bottom-Up Integration: As its name implies modules. Regression Testing: In this context of integration test strategy, regression testing is the re-execution of some subsets of test that have already been conducted to ensure that changes have not propagated unintended side effects.

7.2 Acceptance Testing

At the culinitanation of integration testing, software is completely assigned as a package; interfacing errors have been covered and corrected and a final series of software tests – validation testing may begin. Validation can be found in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.

Alpha Testing

It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combination of data may be regularly used; and output that seemed clear to the tester may be unintelligible to a user in the field.

Beta Testing

It is conducted at one or more customer sites by the end user of the software. The beta test is a live application of the software in the environment that cannot be controlled by the developer. The customer records all programs that are encountered during beta testing and reports these to the developer at regular interval.

7.6 Results

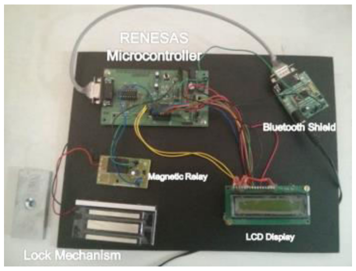


Fig 7.6.1 Hardware Design

Snapshots:

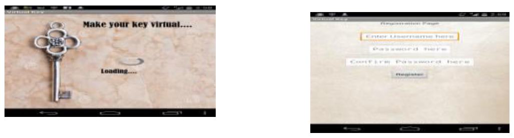


Fig 7.6.2 Splash Screen and registratio page

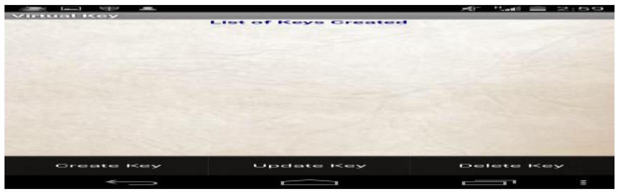


Fig 7.6.3 Key View

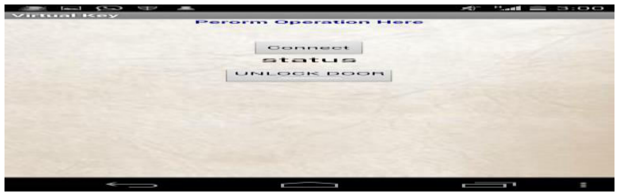


Fig 7.6.4 Lock/Unlock

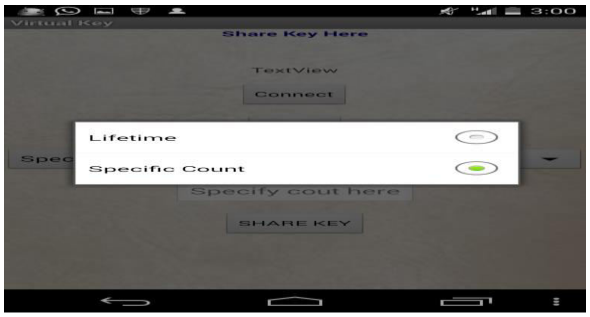


Fig7.6.5 Key Share

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 Conclusion

Security is an aspect, which goes hand in hand with every technical advancement. The future of technology is the merging of ubiquitous technologies into a single easily manageable device. Our project is intended on upholding this feature in the aspect of real-life key management. Here, the use of keys in our day to day life can be further enhanced if the keys were in form of digital data. This is the fundamental idea in our project. By virtualizing keys, we can provide new features and better enhanced techniques for easier access and improved sharing techniques. We do this by virtualizing all the keys used by a person and storing it in a smart phone, which has the power to support emerging technologies. In our case, the keys used by a person in his everyday life such as home key, car key, office key etc, are virtualized and stored in a digital format in an android device. The key management is simplified with the help of an android application, which is easy to use by any user, and provides better key management techniques. A user can view the keys to all his locks on a single platform. This removes the hassle of carrying many keys in our pocket. In addition, the virtual keys can easily be shared with trusted parties remotely by using SMS feature. The owner of the key can also control the number of access. The corner stone of our project is security. We must be able to guarantee the user that his keys are secure and only he has the privilege of unlocking and sharing. This is done so by protecting the application with a password, which only the owner of the key knows. Hence, even if the phone is lost or stolen, key security is not compromised. Also, when a user shares his keys, he can restrict the number of access and once this is crossed, the key is automatically deleted. Also, every key has a single owner and he is the only one who has the privilege to share keys. Hence, shared keys cannot be shared again nor can they be duplicated. The keys are stored in the form of digital data for simplified access and sharing. To implement this in real life, the locking mechanism also has to be changed. We have done this by using a magnetic lock, controlled by a microprocessor. The interaction between the locking mechanism and the keys is done via Bluetooth communication. This locking mechanism can be easily installed on doors. The processor in the lock interacts with the phone and receives keys via Bluetooth communication. The processor first determines if the key is valid or not before unlocking the lock. Enhancements can be made to our proposal and be implemented on a large scale such as an Institution or an entire office. This simplifies administrative tasks as the control of all keys are under a single person. This is a small effort towards new thinking and possibilities with the help of technical advancements in all fields. We hope that our proposal is implemented on a large scale such that key management is simplified.

8.2 Future Enhancement

The system can be further specialized for its materialistic usage by incurring few of the advancements mentioned below:

- Implementing for other platforms such as vehicle locks, briefcase etc.
- Completely integrating the proposed system on a real platform by keeping the hardware part on a fixed door and the iron piece on a movable door.
- Providing battery on our proposed system so that the device is foolproof in case of power failure or forcible removal of power supply
- Implementing in a university (like medical colleges, schools, etc.) by incorporating few changes as required by the institution.

References

[1] *"Text Compression and Encryption Through Smart Devices for Mobile Communication", IEEE 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing by Raffaele Pizzolante at al.*

[2] *"A Routing-Driven Elliptic Curve Cryptography Based Key Management Scheme for Heterogeneous Sensor Networks", IEEE Transactions on wireless communications, vol. 8, no. 3, march 2009 by Xiaojiang Du, Mohsen Guizani, Yang Xiao and Hsiao-Hwa Chen.*

[3] *"Specification of the Bluetooth System & mdash, Core", available online at http://www.bluetooth.com/developer/specification/Bluetooth_11_Specifications_Book.pdf.*

[4] *"Specification of the Bluetooth System &mdash, Profiles", available online at http://www.bluetooth.com/developer/specification/Bluetooth_11_Profiles_Book.pdf.*

[5] *G. Miklos et al., "Performance Aspects of Bluetooth Scatternet Formation," poster presentation at Mobile Ad Hoc Networks and Computing (MobiHOC 2000), IEEE/ACM workshop, Aug. 2000.*

[6] *T. Salonidis et al., "Distributed Topology Construction of Bluetooth Personal Area Networks," Proc. IEEE Infocom 2001, IEEE Communication Society, New York, 2001.*

Authors Profile:

Miss.Lekhana SV is a graduate student from Channabasavweshwara Institute of Technology, Tumkur. She has been certified with web technologies and well versed in Cryptography. She is been working as A Graduate assistant under Prateek Shantharama, Ph.D. Student, Arizona State University, USA.

Mr.Prateek Shantharama is a Phd student in Arizona State University, USA. He has finished his post graduation in The National Institute of Engineering, Mysore and graduation from Siddaganga Institute of Technology, Tumkur. He is well versed in Networking, 5G, SDN, Wireless Technologies.