

THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF ELECTRICAL ENGINEERING
AND
TELECOMMUNICATIONS



Access Control using Bluetooth TM09

Mei, Jiexiang Marvyn - 2244246
Salim, Agus - 2251992

Supervisor: Dr. Tim Moors
Assessor: Dr. Saeid Nooshabadi

November, 2003. Sydney, Australia.
Bachelor of Engineering in Electrical Engineering

Acknowledgements

Both members of this group thesis would like to thank a few people for their kind help and support throughout the duration of this project.

Firstly, we would like to thank our thesis supervisor, Dr Tim Moors, who not only responded to the problems our group faced during the thesis, but has also done his best to make available whatever resources and logistics support we required, in a timely manner.

We would also like to thank Joseph Yeo, the laboratory technician in-charge of the Bluetooth laboratory, for providing us with the required logistics support we required, including, but not limited to, making available to us the computers that were crucial to the development of the software in this thesis, and also the servicing of computers that broke down.

Abstract

This thesis project proposes a method of access control using Bluetooth. Currently access control methods require physical contact to a device, such as a swipe-card. Bluetooth is a wireless technology that could be used to replace such applications, and provide the convenience of wireless access control. The purpose of this thesis is to demonstrate that such a concept is feasible, by implementing one such application.

The application developed in this thesis involves the shutting off of ringing tones of mobile phones that enter a “silent” zone. Such a policy is enforced by a Bluetooth access point situated in this “silent” zone. The access point will send out a “beacon” signal to the mobile phone, via Bluetooth, telling it to shut down its ringing tone. Another focus of this thesis is to ensure that this procedure is secure. Both the mobile phone and access point will have to carry out an authentication procedure, designed in this thesis.

With the success of this thesis, such a concept could be extended to other forms of access control applications.

Table of Contents

Acknowledgements	ii
Abstract	iii
Table of Contents	iv
1. Introduction	1
1.1 Background	1
1.2 Motivation	1
2. Features of Bluetooth	2
2.1 Strengths	2
2.2 Marketing Aspects	4
2.3 Radio Spectrum	4
2.4 Ad-hoc Radio Connectivity	5
2.5 Network Topologies	6
2.6 Bluetooth Security	8
2.7 Bluetooth Power Consumption and Operating Modes	10
2.8 Bluetooth Inquiry and Connection Establishment	11
2.9 Bluetooth Software Stack	13
3. Bluetooth Applications	15
4. Hardware Development	16
4.1 Mobile Phone	16
4.1.1 Nokia 6310i	17
4.1.2 Sony Ericsson P800	18
4.2 Access Point	19
4.2.1 IPAQ and Microsoft Windows CE	20
4.2.2 Microsoft Windows and Casira	21
4.2.3 Microsoft Windows and USB Dongle	21
4.2.4 Linux and USB Dongle	22
4.2.5 The Chosen Platform and Operating System	22
4.3 Programming Tools and SDKs	23
4.3.1 Mobile Phone	23
4.3.1.1 The Symbian Operating System and its Bluetooth Stack	23
4.3.2.2 Programming on Symbian	24
4.3.2 Access Point	24
4.3.2.1 Linux and Bluez	25
4.3.2.2 Programming on Bluez	26
4.3.3.3 Portability	27

5. Thesis Proposal	28
5.1 Problem Statement	28
5.2 Proposed Solution	28
5.3 Requirements	29
5.4 Possible Approaches	30
5.4.1 Approach 1: Passive Scanning	30
5.4.2 Approach 2: Data Communications and Passive Scanning	31
5.4.3 Approach 3: Regular Polling	32
5.5 Choice of Approach	33
5.6 Specifications	34
5.6.1 Mobile Phone / Access Point Interface	34
5.6.2 Mobile Phone	34
5.6.3 Access Point	35
6. System Design	36
6.1 System Design Considerations	36
6.2 Mobile Phone	37
6.2.1 Software Modules	37
6.2.2 Top-Level State Diagram	39
6.2.3 Inquiry State Diagram	40
6.3 Access Point	41
6.3.1 Software Modules	41
6.3.2 Data Flow Diagram	43
6.3.3 State Diagram	44
6.4 Authentication Protocol	45
6.5 Security and Encryption	47
6.5.1 Public Key and Shared Key Cryptography	47
6.5.2 Introduction to RSA	48
6.5.3 RSA Key Generation	48
6.5.4 Encryption and Decryption	49
6.5.5 Using RSA in ZoneIT	50
6.5.6 Protecting the System – Denial of Service Attacks	50
6.5.7 Protecting the System – Playback Attacks	51

7. Implementation	54
7.1 Mobile Phone	54
7.1.1 Application Framework	54
7.1.2 Audio Disabler Module	55
7.1.3 Bluetooth Server Socket Module	56
7.1.4 Security and Encryption Sub-module	57
7.2 Access Point	58
7.2.1 Scanner Module	58
7.2.2 Communication Module	58
7.2.2.1 Multithreading	59
7.2.2.2 Synchronization	61
7.2.3 Directory Module	62
7.2.3.1 Time Complexity Analysis	64
7.2.3.2 Hash Function	64
7.2.4 Security Module	65
7.2.5 Main Module	66
8. Software Testing	67
8.1 Incremental Testing	67
8.2 Modules Testing	68
8.2.1 Communication Modules Testing	69
8.3 Integration Testing	70
8.4 Final Testing	70
9. Conclusion	72
9.1 Achievements	72
9.2 Commercial Viability	73
9.3 Future Improvements	74
9.4 Final Conclusion	74
10. References	75
11. Appendix	78
11.1 Installing Bluez	78
11.2 Bluez Settings	79
11.3 Source Code	79

List of Figures

FIGURE 1	7
FIGURE 2	8
FIGURE 3	11
FIGURE 4	13
FIGURE 5	17
FIGURE 6	18
FIGURE 7	21
FIGURE 8	37
FIGURE 9	39
FIGURE 10	40
FIGURE 11	41
FIGURE 12	43
FIGURE 13	44
FIGURE 14	45
FIGURE 15	51
FIGURE 16	51
FIGURE 17	53
FIGURE 18	53
FIGURE 19	59
FIGURE 20	60
FIGURE 21	63
FIGURE 22	63

FIGURE 23	64
FIGURE 24	65

List of Tables

TABLE 1: Comparison of Bluetooth, wireless LAN and infra-red technologies	2
TABLE 2: Comparison of the differences in power dissipation in the two modes of activity.....	10

1. Introduction

1.1 Background

The Bluetooth standard and technology came about initially when Ericsson Mobile Communications carried out a study to find a low power and low cost radio interface between mobile phones and their accessories. The study showed that a short-range radio link solution was feasible. To develop the technique and to get broad market support, Ericsson, together with Intel, IBM, Toshiba and Nokia Mobile Phones formed a Special Interest Group (SIG) in 1998. This group was to form a standard for the air interface and the software that controls it, such as to achieve interoperability between different devices from different producers. [1]

1.2 Motivation

The motivation of this thesis is to demonstrate the use of Bluetooth technology in access control applications. One particular application would be to turn off the ringing tone of a Bluetooth-enabled mobile phone as it enters a lecture hall, as the ringing of the mobile phone is undesirable in that location.

Currently, most access control solutions are implemented via the use of conventional technologies, such as bar-coded swipe cards, and access pin code numbers. With the introduction of newer mobile devices with Bluetooth, it is possible to replace older access control technology with applications developed using the Bluetooth technology, allowing access control to be delivered in a wireless fashion.

Because Bluetooth is a relatively new technology, not many Bluetooth applications have been created. Hence writing a Bluetooth software system would be a challenging, but interesting experience for our final year thesis.

2. Features of Bluetooth

2.1 Strengths

The following table compares the Bluetooth radio to wireless LAN and infrared. These three technologies are the most commonly used in many of today's wireless applications. Each of them has their own set of advantages and disadvantages, and this makes each of them suitable to certain applications.

	Bluetooth	Wireless LAN	Infrared
Typical Range	Medium (10 m)	Long (100 m)	Short (1 m)
Line-of-sight	No	No	Yes
Bandwidth	1 Mbps shared	11 Mbps shared	115 kbps & 4 Mbps dedicated
Interference	Other RF devices	Other RF devices	None
Security	Less secure than infrared. Uses link-layer authentication. Still requires application layer security.	Insecure unless protected. e.g. WEP & WPA encryption	Very secure, due to short range and line-of-sight requirement
Power Consumption	High. Needs to maintain a connection	Very high. Needs to maintain a connection.	Low. No constant connection like wireless radios.
Component Cost	About \$20, expected to drop to \$5.	About \$25.	Less than \$2.

Table 1: Comparison of Bluetooth, wireless LAN and infra-red technologies. Source: [2]

The main features of Bluetooth that makes it suitable for use with our project are:

- Minimal hardware dimensions.
- Low price of Bluetooth components.
- Low power consumption for Bluetooth connections.
- Inherent security features (described in section 2.6).
- Medium range.

The low cost and small size of the Bluetooth radios means that it can be integrated into many portable devices cheaply. The products offered from companies in the Bluetooth SIG, such as mobile phones, PDAs etc, creates a huge market potential for Bluetooth devices and their applications.

Low power consumption is especially important in this project because the software system requires the Bluetooth radio on the mobile phones to be turned on all the time. This helps to prolong battery life, which is scarce in mobile phones.

The inherent security features and medium range of Bluetooth makes Bluetooth relatively secure as compared to other wireless radios such as wireless LAN. The security features makes it hard to listen to the data transmissions. The medium range means that hackers would have to be within close physical range to the Bluetooth radio in order to listen to its traffic. All these are important, because this project deals with access control, in which security plays a very important role. Note, however, that Bluetooth security is insufficient for this application because it does not stop other Bluetooth devices from “attacking” the system. An additional application-layer authentication procedure will be needed to filter out these malicious Bluetooth users, as described in section 6.5.6.

The relative low data transfer rate of Bluetooth is not a problem because most data transfers in this project will only involve small quantities of data.

2.2 Marketing Aspects

In the last decades, consumer products such as PCs, laptops, personal digital assistants, cell phones etc have increased in popularity. This is based on the continuous cost and size reduction of these devices [3]. The transfer of information between these devices has been hindered because of the need of cables. Bluetooth provides a solution to this by eliminating the need for cabling. It also provides the means for connecting several units to each other, such as setting up small radio Personal Area Networks between any types of Bluetooth devices [1].

2.3 Radio Spectrum

Bluetooth operates on the unlicensed 2.4 GHz spectrum. This means that the spectrum is open to the public without the need for licenses, as long as they meet requirements specified by the FCC. Moreover, Bluetooth applications are targeted at consumers who do a lot of travelling. Hence the spectrum on which Bluetooth operates on must be available worldwide. The 2.4 GHz spectrum is free in most countries of the world and thus meets this criteria.

The use of this spectrum introduces a lot of interference sources to the Bluetooth radio transmissions. One source of interference is high-powered transmitters such as microwave ovens and lighting devices, which also transmit at the 2.4 GHz band. Another source of interference is co-user interference [3], which comes from other Bluetooth users.

Interference immunity can be obtained by interference suppression or avoidance [3]. However these techniques will not be discussed in this report.

2.4 Ad-Hoc Radio Connectivity

The Bluetooth radio system stands out from other radio systems due to its ad-hoc connectivity. The majority of radio systems used today are based on cellular radio architectures. A mobile network is established on a wired backbone infrastructure, and consists of one or more base stations located in different locations to provide cellular coverage. The mobile terminals then access the network in these coverage areas. Hence there is a clear separation between the base stations and the terminals of such systems.

In contrast, there are no distinctive base stations or terminals in the Bluetooth system, nor are there any differences between radio units. There is no wired infrastructure to support connectivity, and there is no predefined central controller for units to rely on for making interconnections [3].

This means that during the implementation of the system, we would not have to worry about the architecture of the resulting network. This is also important because the Bluetooth-enabled devices will be constantly on the move, resulting in a constantly changing network. The lack of an infrastructure means that the network can accommodate such changes easily.

2.5 Network Topologies

Bluetooth devices can be organized into groups of two to eight devices, which together form a piconet. Each piconet will contain at least one master, and all other units participating in the piconet will be slaves. The master of a piconet controls the communications within a piconet [4].

The number of units in a piconet is deliberately limited to eight (one master, seven slaves) in order to keep a high capacity link between all the units. It also limits the overhead required for addressing. Note that the master/slave role only lasts for the duration of the piconet. Once the piconet is cancelled, these roles are also cancelled. Any unit can become master or slave. By definition, the unit that establishes the piconet becomes the master [3].

Two or more piconets can be interconnected to form a scatternet. The connection point between two piconets is a Bluetooth unit that is a member of both piconets. One device can be a master in one piconet, and a slave in another. A device can also be a slave on more than one piconet, but it cannot be a master in more than one piconet, as this will mean that the two are actually one piconet (having a single master).

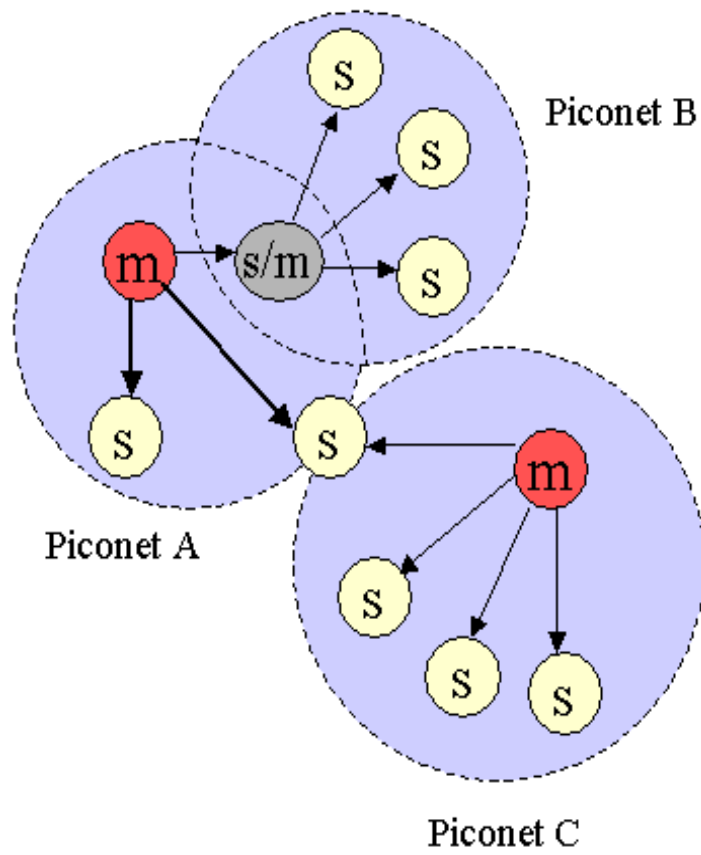


Figure 1: Shows a scatternet comprising of three piconets. Two piconets are linked to each other by a node which exists in both piconets. (Figure adapted from [1].)

2.6 Bluetooth Security

Security in Bluetooth is provided in three ways [1]:

1. Pseudo-random frequency hopping.
2. Authentication
3. Encryption

Frequency hopping is a spread spectrum technique that was intended for noise resilience. However it is also a good way to prevent eavesdropping [5], because it is very hard for an eavesdropper to track the frequency changes of a transmission over a Bluetooth network. The following figure shows how the frequency of a Bluetooth channel can change over time.

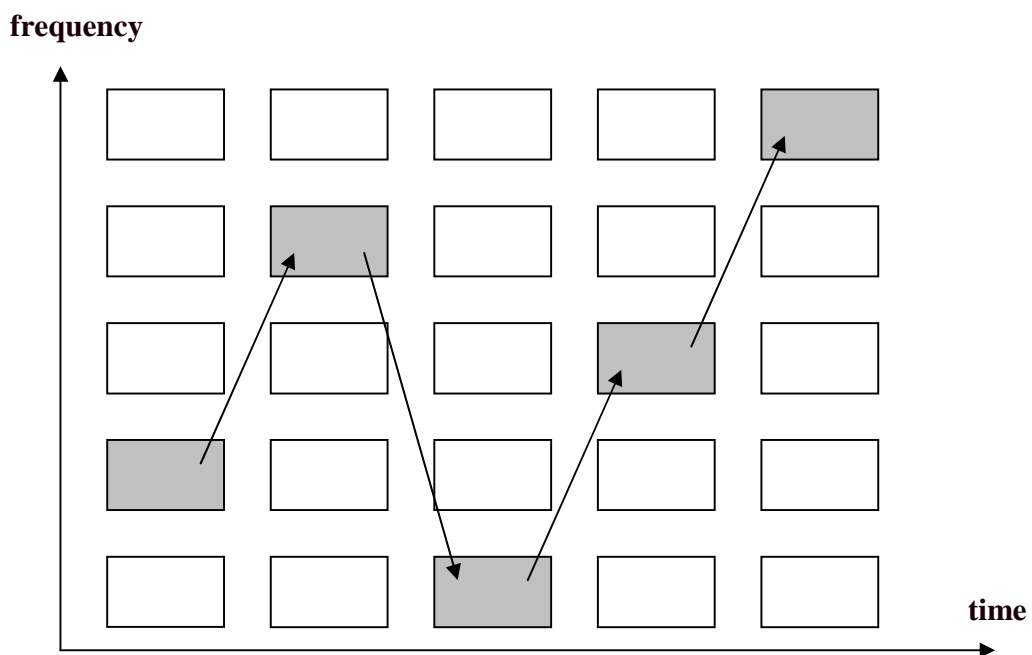


Figure 2: Frequency of a Bluetooth channel changing pseudo-randomly.

Authentication allows a user to limit connectivity to specified devices. Encryption makes data readable only to authorized devices/parties [1], hence preventing eavesdropping.

All Bluetooth-enabled devices implement the Generic Access Profile, which defines a security model that includes three security modes [1]:

1. Insecure mode. No security procedures are carried out.
2. Service-level enforced security. No security procedures are carried out before a channel is established.
3. Link-level enforced security. Security procedures are initiated before link setup is complete.

The in-built security measures of Bluetooth are important for this project, which deals with access control. However, these inherent security measures will not be sufficient for this application, as will be explained later.

2.7 Bluetooth Power Consumption and Operating Modes

Bluetooth supports four modes of activity in order to save power. In order of decreasing power consumption, the four modes are [6]:

1. Active mode
2. Sniff mode
3. Hold mode
4. Park mode

In this thesis, we are concerned mainly with the active mode and the hold modes, since they correspond mainly with the main procedures of the mobile phone (see section 6.2.2).

In active mode, the master and slave communicate with each other on the same channel. This would obviously use the most amount of power. This mode is used when there is data communications between both master and slaves. [6]

In hold mode, there is no active communications between master and slave. The slave merely listens to the channel, to see if it should exit this mode. During this time, the slave is able to scan, page or inquire about devices in the same area. This mode consumes significantly less power than the active mode, as noted in the following table. [6]

Product	Modes of Activity	
	Power Dissipation in Park Mode	Power Dissipation in Active Mode
CSR Bluecore 01	0.3 mW	0.6 – 135 mW

Table 2: Comparison of the differences in power dissipation in the two modes of activity.

(Table taken and extracted from [6].)

2.8 Bluetooth Inquiry and Connection establishment

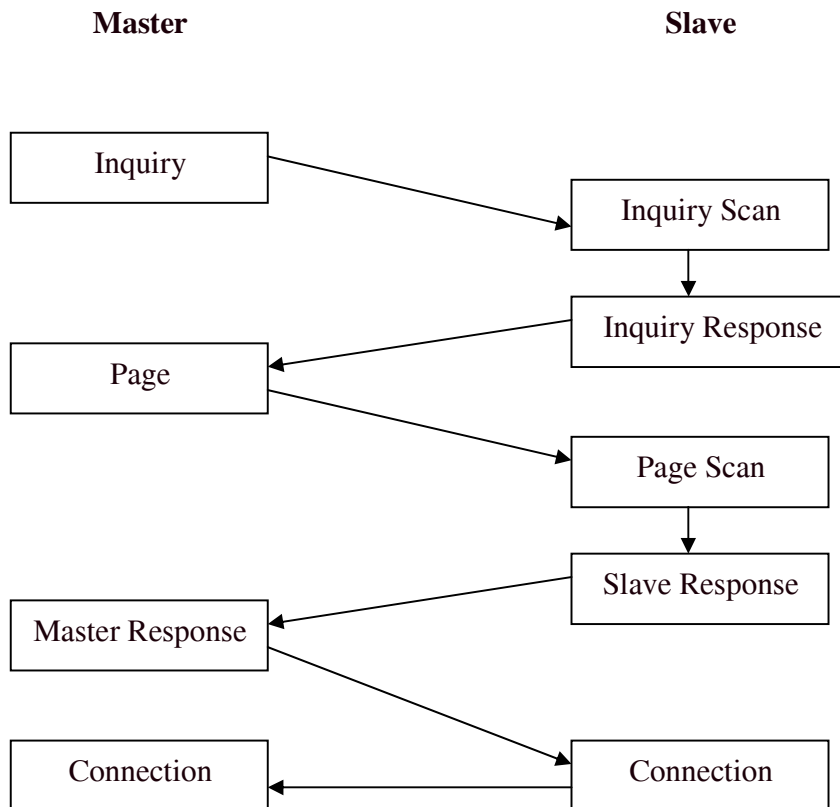


Figure 3: Typical Bluetooth connection procedure. (Taken from [7].)

The inquiry procedure is an important part of Bluetooth. It is a process where one Bluetooth device tries to find all neighbouring Bluetooth devices. A device that is trying to scan for other devices is said to be in 'inquiry mode'. The device that listens for an inquiry request is in 'inquiry scan mode'. This 'inquiry scan mode' is usually set in a Bluetooth device by setting it to be 'discoverable'.

When inquiry is initiated, the device goes into 'inquiry mode' and accelerates its hopping frequency. On the other hand, the device in the 'inquiry scan mode' reduces its hopping frequency. This algorithm will allow the inquirer to catch up with the transmit frequency of devices that is in 'inquiry scan mode'. This is important because of the frequency

hopping algorithm employed in Bluetooth. When the frequencies coincide, the scanned device will act as slave and send its address and clock information to the master. [8]

After inquiry, the inquirer will be able to initiate connection to the inquired device. Since the connection is initiated by the inquirer, it will act as the master. This initial connection is called paging. Paging is done by the master by sending paging requests to possible slave frequency slots. This frequency slots are calculated from the Bluetooth address and the clock information received during inquiry.

At the time of connection establishment, the slave will synchronize its timing to that of the master's. Throughout the connection, the master never changes its hopping sequence or phase (current hop slot, determined by the master's clock). In contrast, the slave will have to synchronize with the master's clock all the time [7].

2.9 Bluetooth Software Stack

Bluetooth can be defined as a layered protocol architecture consisting of protocols such as the cable replacement protocol, the telephony protocol, the adopted protocol and the core protocols [1]. The layers are specified to add abstraction and to adapt the Bluetooth technology to other existing protocols.

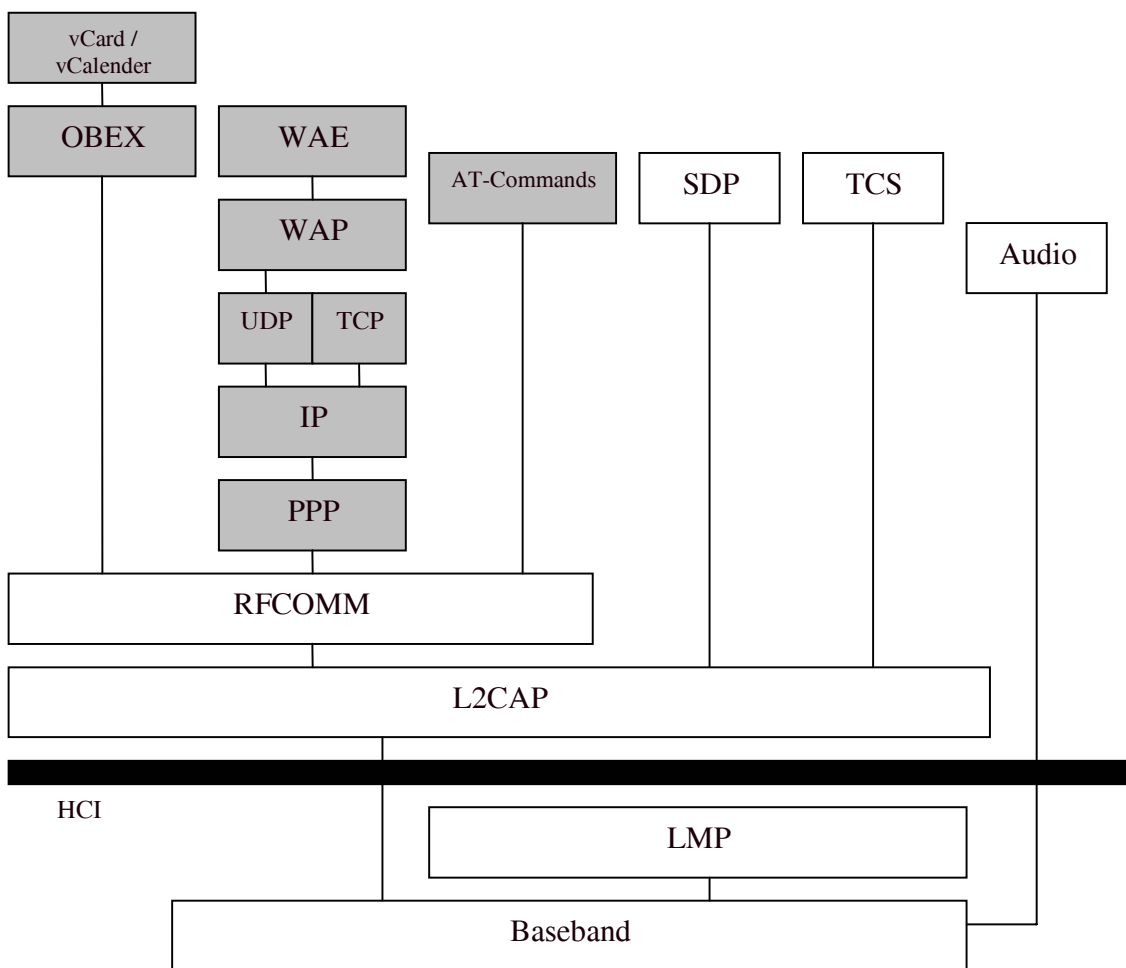


Figure 4: The Bluetooth Protocol Stack. (Diagram reproduced from [1].)

The core protocols are four protocol layers consisting of [1, 9]:

- **Baseband.** Specifies details of the air interface, including frequency, the use of frequency hopping, modulation scheme, and transmit power. Also specifies the protocol for connection establishment within a piconet, addressing, packet format, order of transmission, timing sequence, power control and channel coding.
- **Link manager protocol (LMP).** Responsible for link setup between Bluetooth devices and ongoing link management such as encryption and security. Allows service discovery, which detects other Bluetooth devices when in range.
- **Logical link control and adaptation protocol (L2CAP).** Adapts upper layer protocols to the baseband layer. It is also responsible for the multiplexing data to upper layer. This is done by assigning a particular PSM (multiplexer number).
- **Service discovery protocol (SDP).** Query devices of possible services available. Does not require a connection to be made.

We are interested in three particular layers, as they are the communication layers that we are likely to implement our system in. These layers are [1, 9, 10]:

- **HCI** (Host Controller Interface) sits between the L2CAP and LMP. It is the first interface between programmer and the protocol, allowing access to the hardware capabilities. It is also responsible for the multiplexing data to upper layer.
- **L2CAP** layer, as mentioned above acts as an adaptation layer to the upper layer, but it also allows connectionless data transfer.
- **RFCOMM** layer is a cable communication protocol designed to emulate serial ports.

We decided to program at the L2CAP layer because RFCOMM only provides one-to-one connections while L2CAP allows multiplexing of data via the PSM number. This means that two Bluetooth devices can have more than one connection to each other, allowing different Bluetooth programs to run simultaneously. This may not be important for our current implementation, but it may be useful for future implementations.

3. Bluetooth Applications

Most of today's Bluetooth applications can be grouped into several categories, namely

1. Office applications
2. Ad-hoc Networking
3. Access control applications

Office applications often relate to the setting up of a wireless office environment. For example, the keyboard and mouse of a computer are connected to the computer itself via a Bluetooth link, or connecting several computers to a Bluetooth-enabled printer. It also involves the synchronization of information between computers. For example, a business man who travels a lot would write a lot of information on his PDA. When he gets home from work, he will want this information to be added to his laptop computer. This can be done easily using if both his PDA and his laptop computer are Bluetooth-enabled.

Ad-hoc networking involves several devices in range, forming a community of networks (Personal Area Networks) that only exists as long as it is required. Current applications include the exchange of electronic business cards between users.

Bluetooth devices can be used in access control applications. One application is that it can be for mobile payment, like a credit card. A Bluetooth device such as a Bluetooth-enabled mobile phone could communicate with a Bluetooth-enabled cinema ticketing machine, for example, allowing the user to purchase a ticket without queuing, and then billing the appropriate amount to his phone bill. Alternatively, each Bluetooth device could contain a unique code which is used to identify a particular user, and how much credit he has in his credit account. Bluetooth can also be used as an access card. A user can access controlled areas of a building, or certain resources, by just being there, without having to swipe his/her card in a swipe machine.

4. Hardware and Development

4.1 Mobile Phone

The mobile phone would run the phone version of the ZoneIT software. The software would turn the ringing tone volume of the mobile phone down if the phone detects a nearby access point.

Minimum requirements for the mobile phone are:

1. Programmable.
2. Bluetooth-enabled.
3. API must have access to Bluetooth functionality.

To allow Bluetooth programming, a Bluetooth protocol (software) stack is required. This protocol stack is explained earlier in section 2.9, and it is usually implemented in the SDK. The SDK can be seen as simply a library that allows a programmer to control the Bluetooth devices.

Our team had to make a decision regarding the mobile phone we should consider using in the system. The following are the mobile phones which were considered.

4.1.1 Nokia 6310i



Figure 5: The Nokia 6310i.

This Bluetooth-enabled mobile phone was provided to our team by the University, through our supervisor, Dr Tim Moors. The initial plan was to program the phone using Java, which was the only way to program this phone, according to a Nokia representative we spoke to. However the Nokia representative also mentioned that the Bluetooth functionality of the functionality was not available to the programmer. Hence, it was not possible to use this mobile phone for our project.

4.1.2 Sony Ericsson P800



Figure 6: The Sony Ericsson P800.

This mobile phone was available to our team through Agus, who happened to own the phone. After doing some research on this phone, our team discovered that the phone ran on the Symbian Operating System, which provided an API for the Bluetooth functionality of the mobile phone. This made development on the mobile phone possible.

The Symbian Operating System is a popular operating system for mobile devices such as PDAs and mobile phones. This means that it would be easy to port our phone code over to these mobile devices easily.

4.2 Access Point

The access point is used to control the ringing volume of the surrounding mobile phones running the ZoneIT software. It acts as intelligent beacon, authenticating itself with the mobile phones using the ZoneIT software.

Minimum requirements for the access point are:

1. Bluetooth capability.
2. Programmable, supports Bluetooth discovery and data communications.
3. Ability to store and process Bluetooth addresses.

As with the mobile phone, a Bluetooth protocol stack is also required.

We have several choices to make regarding the platform to deploy the access point, and the operating system on which the access point would operate.

1. Bluetooth-enabled IPAQ PDA (Personal Digital Assistant).
2. Windows and Casira CSR BT device.
3. Windows and USB Dongle.
4. Linux and USB Dongle.

4.2.1 IPAQ and Microsoft Windows CE

The IPAQ PDA has many appeals. It is small and quite powerful, while running on the Windows CE operating system. The major disadvantage is that it cost a little over \$1000 for a typical PDA, which implies that using the IPAQ as an access point would not be very cost effective when it comes to deployment.

Even so, our team was attracted to its simplicity and elegance. We decided to try developing the access point on the IPAQ. After much research, we found out that the SDK (software development kit) for the Pocket PC is freely distributed. Unfortunately, the SDK requires either Microsoft eMbedded Visual C++ development, or Microsoft eMbedded Visual Basic development to be used [11]. Those programming environments (IDE) will cost thousands of dollars, and hence we decided against using the IPAQ.

4.2.2 Microsoft Windows & Casira



Figure 7: Carisa micro-controller

The Casira micro-controller has a full Bluetooth protocol stack running on the BlueCore Chip. It is a combination of micro-controller and Bluetooth radio device called BlueCore Radio Module [12]. Using Casira is attractive because it can act as a stand-alone device. But its major disadvantage is that there is a limitation in the micro-controller's on-board memory. This could possibly impose a limit on the length of our code. Moreover there were also other students in the Bluetooth laboratory competing with us for the use of these modules. Hence we decided to go for other alternatives.

4.2.3 Microsoft Windows and USB Dongle

The Microsoft Windows platform coupled with a Bluetooth USB device (dongle) is another attractive combination to implement the system with. The university already has many computers running on Microsoft Windows in lecture theatres. By adding Bluetooth dongle and access control software, we can implement the system in our university lecture rooms. After research, we found that the SDK from Microsoft was free [13]. Unfortunately, after testing on two different computers we discovered that the Bluetooth stack in the SDK did not work properly, since we were unable to access the Bluetooth stack functions. This could possibly be because the Bluetooth USB dongle we were using was not supported. Similar remarks regarding this problem were also found on various forums over the internet.

4.2.4 Linux and USB Dongle

Similar to the previous case in section 4.2.3, this option is attractive because computers are widely available in the university. In addition, the Linux operating system is part of GNU, so it is freely available and downloadable from the internet. GNU software means that the public is allowed to distribute, use, and modify the software [14].

After some research, we found a Bluetooth stack for Linux called Bluez. At that time, Bluez had become the official Bluetooth stack for Linux. Just like Linux, Bluez is also GNU software.

4.2.5 The Chosen Platform and Operating System

After having evaluating and testing these options, and weighing the pros and cons of each, we finally decided to implement the access point program on Linux with a USB dongle, as detailed in section 4.2.4.

The advantages of using Linux with the Bluetooth USB device are:

- The cost for a Bluetooth dongle is relatively low (approximately \$60).
- The Linux operating system and Bluez Bluetooth stack are free.
- Uses C programming language that comes with most Linux Operating System. Such as GCC or CC.
- Bluez implements most layers in the stack such as SDP, RFCOMM, L2CAP, HCI layers.
- It supports many Bluetooth devices, including the MSI dongle we are using [15].

We used Mandrake Linux Version 9.1 with Bluez. The installations for Bluez onto Linux will be described in the Appendix.

4.3 Programming Tools and SDKs

This section shall describe the programming tools and software development kits (SDKs) which were used in the development of our software system. Since both mobile phone and access point are different platforms, each has their own set of tools and SDKs. Also discussed are some of the advantages in using these tools and SDKs.

4.3.1 Mobile Phone

4.3.1.1 The Symbian Operating System, and its Bluetooth Stack

The Symbian operating system is the operating system on which the Sony Ericsson P800 runs on. The Symbian operating system comes in several variants. For example, the Series 60 mobile phones run on the Series 60 Symbian operating system, while the Sony Ericsson P800 runs on the UIQ Symbian operating system. All these operating systems are similar, and most of the code implemented on one Symbian operating system and can be ported rather easily to another Symbian operating system.

The Bluetooth stack provided by the Symbian operating system implements all the necessary components that define the Bluetooth specifications v1.1. In addition, it gives the programmer access to RFCOMM, L2CAP, SDP, and to a limited extent, HCI layers. Since our team is programming at the L2CAP layer, this Bluetooth stack is sufficient for our requirements. [16]

4.3.1.2 Programming on Symbian

Programming on Symbian was straightforward, except for the fact that the code had to be compiled and deployed on the mobile phone before any testing of the code could be done. Under such circumstances, an emulator for the phone would be ideal. Sony Ericsson does provide such an emulator, but at a hefty cost. However, even if we are willing to pay for the cost of the emulator, it would be practically impossible to test our Bluetooth code directly on the emulator, since the emulator does not recognise external Bluetooth devices.

Starting out on programming required the study of existing code. The code our team used was the “HelloWorld” example. Our team modified this code, and it formed the application framework of our software system. Since the focus of this project is Bluetooth development, we did not want to spend too much time on the user-interface.

Sony Ericsson provides an online-forum, which only provides technical assistance. Most of the time, it did not help. However there are several websites that have been established by Symbian programmers. Although most of these sites do not focus on UIQ Symbian programming, they were of much help because as mentioned before, most Symbian code can be ported easily across the variants of the Symbian operating systems.

Most of the information we required for the phone was on the documentation, which is packaged together with the Symbian SDK. The information was very useful and was critical to the success of the mobile phone software. However, there were some problems in the documentation. For example, there were errors in the example codes, and several functions were not well-described in the documentation.

4.3.2 Access Point

4.3.2.1 Linux and Bluez

Linux is a well known open source operating system released under the GPL (GNU Public License). It means that it is free for use or distribution. Recently, Bluez has been made the official Linux Bluetooth stack. Thus, a more recent Linux distribution with version 2.4.6 or higher would include Bluez packages. This also suggests that Bluez has been recognized as a reliable stack compared to other Linux Bluetooth stacks, and it will receive more attention and improvements by the open source community in the future. Bluez was originally developed by Qualcomm Incorporated.

Other well-known Bluetooth stacks are OpenBT, and Affix. Both are also open source. OpenBT has been somewhat static in terms of updates and improvements, while Affix is a newer stack that is developed by the mobile phone company Nokia.

The rationale for using Linux and Bluez was initially due to economic reasons, as other combinations of Bluetooth stack and operating systems are expensive. But as the project moved along, we discovered many positive aspects of Bluez.

Bluez implements almost all of the available stacks that is defined by the Bluetooth standard. The lists of implemented stacks taken from the Bluez official website are: [17]

- Bluetooth Core - HCI device and connection manager.
- HCI USB, UART, PCMCIA and VHCI (Virtual HCI) driver.
- L2CAP- Reliable datagram protocol.
- RFCOMM – Serial port emulation. Reliable connection oriented streaming protocol.
- BNEP - Ethernet Emulation.
- SCO – Synchronized connection (voice).

Bluez supports link layer security, and multiple connections. Those are crucial for our project. On top of those, it also allows multiple Bluetooth devices, which may be required for future extensions to this thesis. For example, we can use multiple devices create a single “virtual” access point. This enlarges the coverage area of the service.

4.3.2.2 Programming on Bluez

Bluez is an open source project [17]. Hence there is not much support given to programmers using the stack. The API (Application Programmer Interface) provides references to functions available to a programmer. Unfortunately, an API does not exist in Bluez. Hence, in order for a programmer to program using Bluez stack, they will have to look at the given example files, and also try screening the Bluez mailing list for any topics related to the development of the program.

The files given from the packages were in the form of C (source) files and H (header) files. There was little, or no documentation made regarding the functions used in those files. Thus it was considerably hard for us to start on the programming aspect for the access point. But the mailing list and the FAQ (Frequent Asked Questions) helped in the understanding of the source code, which provided the basis for our access points programs.

Fortunately for us, Bluez L2CAP is a socket based programming that uses standard UNIX C socket programming. Hence we were able to understand the basic requirement easily from other sources.

4.3.3.3 Portability

As this report mentioned earlier, Bluez uses Unix C, and it also uses standard Socket programming. This is mentioned in the Bluez web site that says, “Adding Bluez support to any existing socket based programs is very easy. For example, you would use AF_BLUETOOTH instead of AF_INET (ip) when you make a socket call. You would use sockaddr_l2 instead of sockaddr_in and SOCK_SEQ_PACKET instead of SOCK_STREAM and so on. Only a few new data structures and constants are introduced.” [17].

This means that it is easy to port from other C socket program to Bluetooth C socket. It also implies that it will be relatively easy to port a C socket for Bluez, to another stack or even another platform. After doing the programming for the access point, our team believes that the claims hold true for communication module, which uses L2CAP socket based programming. But the same cannot be said for the scanning module which uses HCI layer that is particular only to Bluetooth.

5. Thesis Proposal

5.1 Problem Statement

Currently, most access control solutions have been done via the use of conventional technologies, such as bar-coded swipe cards, and access pin numbers. With the introduction of newer mobile devices equipped with Bluetooth, it is possible to replace older access control technology with the use of Bluetooth technology. Bluetooth allows the development of wireless access control applications. Our thesis attempts to demonstrate the use of Bluetooth technology in access control applications. Our group has selected one particular application, which is described below.

The ringing tones of mobile phones often interrupt a meeting, such as a lecture, which is being carried out. This is often due to the mobile phone users forgetting to turn off their mobile phones prior to the meeting. This not only annoys the other parties in the meeting, but also interrupts the meeting that is taking place.

5.2 Proposed Solution

A practical solution to this problem would be to turn off these mobile phones automatically if possible, as they enter a designated “silent” zone, such as the meeting area. In that way, even if the owners of the mobile phones forget to turn them off manually, the mobile phone ringing tones will not interrupt the meeting when there is an incoming call.

This concept could then be extended to other access control applications such as using a Bluetooth-enabled phone to gain access to a room, which is secured by a Bluetooth-enabled lock. Another example would be to create an application that could provide a guided tour of a museum, for example, with text, audio and images being provided for the location the visitor is in.

5.3 Requirements

Our project team would like to propose, design and implement a system which solves the above-mentioned problem. The following gives a top-level description of how the system should behave:

- A user with a mobile phone enters a designated “silent” zone. The coverage area of an “access point” defines the zone.
- When the user enters the “silent” zone, the user’s mobile phone would be able to detect the presence of the access point.
- The mobile phone should be able to tell from the access point that it is indeed in a “silent” zone, and should turn down its ringer volume as required.
- The mobile phone keeps its ringing tone volume down as long as it is within the coverage area of the access point. If it exits that coverage area, the volume on the mobile phone will be restored to its previous state.
- The system would have to be resilient to two forms of denial-of-service attacks.
 - A malicious Bluetooth device should not be able to turn off the ringing tone of a mobile phone.
 - The access point should be robust, such that an attacker would not be able to bring down the services provided by it.

The requirements of the system are defined loosely so as to allow for different approaches to the problem. The next section shows the possible approaches which should meet the requirements.

5.4 Possible Approaches

The following are three possible approaches our team came out with during the formulation of a solution to the problem. Each solution has its own advantages and disadvantages.

5.4.1 Approach 1: Passive Scanning

There would be no actual data communications between the mobile phone and the access point. The Bluetooth addresses of the access points of the system would be either hard-coded into the mobile phone software, or accessible via a database in the mobile phone. When the mobile phone does a periodic inquiry process and detects nearby Bluetooth devices, it would compare the Bluetooth addresses of these devices with those in its database. If there is a match, the mobile phone would know that it is in a “silent” zone and would shut off its ringing tone.

The advantage of such a solution is that it is simple to implement, since we would not have to do any form of socket programming, and we would not have to do any form of programming on the access point. Another advantage is the scalability of the system, since each mobile phone is responsible for shutting itself down. The access point merely acts as a beacon with no processing requirements.

The disadvantage of such an approach is that there is no way to implement any form of application layer authentication, meaning that the system would be susceptible to denial-of-service attacks (see section 6.5.6). This may occur if a malicious user spoofs the Bluetooth address of a genuine access point, allowing him to illegally shut off the ringing volume of a mobile phone. Moreover the mobile phone would need to know before-hand the Bluetooth addresses of all access points, meaning that the databases on the mobile phone may need to be updated frequently. This makes the deployment and maintenance of such a system difficult.

5.4.2 Approach 2: Data Communications and Passive Scanning

The mobile phone would run a server and the access point would run a client. The access point would periodically scan for nearby Bluetooth devices. Once it finds a Bluetooth device (phone), it attempts to set up a communication channel (via a socket) with the phone, and authenticates with it. Once authentication is complete, the phone shuts off its ringing tone and records the Bluetooth address of the access point. It then continues to inquiry for nearby Bluetooth devices, checking if the access point is still in the coverage area using the recorded Bluetooth address.

The advantage of this approach is that it is possible to authenticate the access point, regardless of its Bluetooth address. Hence there is no need for hard-coding the Bluetooth address of all possible access points or for storing all those addresses in a database. This approach also enjoys scalability. The access point only needs to authenticate each mobile phone once, and ignore it for the rest of the time. The mobile phone would determine if it is still in the coverage area of the access point.

The disadvantage is that it is still possible to spoof the Bluetooth address of the access point, allowing a malicious user to fool a phone using the software into thinking that it is still within the coverage area of an access point, when it is actually not, in the inquiry phase (i.e. after the authentication phase).

5.4.3 Approach 3: Regular Polling

This approach is similar to second approach (5.4.2). However, the access point would attempt to establish a communication channel with the phone at every predefined period (requires a timer), instead of just once. The phone would not have to do any inquiry of nearby Bluetooth devices. As long as the access point manages to communicate with the phone at that time interval, the phone ringing tone remains turned off. Otherwise, the phone would assume that it is outside the “silent” zone and restore its ringing tone.

The advantage of this approach is that it is more secure than the other two approaches. This is because the access point is being authenticated at regular intervals. A malicious user may still spoof the Bluetooth address of the access point, but would not be able to authenticate itself with the mobile phone. Moreover, in such an approach, all the processing load is transferred from the mobile phone to the access point. Since a mobile phone has significantly less processing power than the access point (which is powered by a PC), this may be an important factor worth considering.

One significant disadvantage of such a approach is the scalability of the system. If too many users attempt to use the system simultaneously, the access point may have trouble coping with these users as it tries to authenticate all of them at regular time intervals. It is possible that the access point may be in the midst of authenticating a list of users, when the next list of users arrives, leading to an accumulation of backlog of users it has to authenticate. Another disadvantage is the energy requirements. Such regular polling will require regular data communications (active mode), and this will consume a lot of battery power from the mobile phone, as described in section 2.7.

5.5 Choice of Approach

Our team has decided to use the second approach, as described in section 5.4.2. This is because of the possibility of our system being deployed in a situation where there are lots of users, such as in a lecture theatre. This means that the system has to scale to a large number of users, and hence the third approach described in section 5.4.3 cannot be taken. Also, because we want our system to be secure, we cannot use the approach as described in section 5.4.1. Although the second approach is susceptible to spoofing, our team feels that it is a good compromise between approach 1 and 3, and hence we have decided to take this approach.

It is possible to allow the user to be able to select between the three approaches in the mobile phone software system, since these three approaches can be programmed as independent components of the system. However our team has decided not to do so, due to the lack of time, and have left it as a possible extension for future work.

5.6 Specifications

From the above requirements and approach, our team has developed a set of specifications which our system would have to adhere to, in order to provide the required functionality.

5.6.1 Mobile Phone / Access Point Interface

- Both the mobile phone and access point would communicate via the Bluetooth v1.1 specifications.
- Using the client-server relationship model of most communication systems, the mobile phone would act as a server, while the access point would act as a client. The access point would access the mobile phone.

5.6.2 Mobile Phone

- Upon starting up, the software shall run a server module, which listens for incoming Bluetooth connections.
- If an incoming connection is initiated, the mobile phone would have to ensure that the owner of the incoming connection belongs to a trusted access point. If not, it would close the connection and continue listening for new connections.
- After authentication, the mobile phone deduces that it is in a “silent” zone, and automatically turns down its ringing tone volume. The Bluetooth connection is then closed.
- During this period, the mobile phone does a Bluetooth “inquiry” on the access point periodically, in order to ensure that it is still within the coverage area of that access point.

- Because Bluetooth is a wireless communications standard, and it experiences frequent dropouts, this “inquiry” procedure would have to fail several times before the mobile phone decides that it is not in the coverage area of the access point. If it fails too many times, the software stops the “inquiry” procedure, and restores the volume of the ringing tone.
- Once out of the coverage zone, the software restarts the server module to listen for new incoming connections.

5.6.3 Access Point

- The access point scans for all nearby Bluetooth devices inside its coverage area. The access point will keep a list of all the devices in the form of Bluetooth addresses, which is unique in all Bluetooth devices.
- The access point will then try to communicate with all the mobile devices, and disable the ringing tones of those mobile phones which have the ZoneIT software installed.
- During communications, the access point will be required to authenticate itself with the mobile device. This authentication is used to make sure that the access point is authorised to disable the ringing tone of the mobile phone.
- The access point must be efficient, possibly connecting to several mobile phones simultaneously.
- Finally, the access point should not reconnect to those mobile phones it has connected to before, since it would be redundant to do so.

6. System Design

6.1 System Design Considerations

Before starting on the design procedure, we need to enhance our basic understanding of Bluetooth communications, so that we can make the right decisions when carrying out the design.

The first issue is the connection initiation. Connections can be initiated either by the access point or the mobile device. After careful consideration, our team believes that if the mobile phone were to initiate the connection, they would have to screen through all the available devices in the vicinity. This is power consuming for the phones, and we do not want to burden these mobile devices in such a manner. Moreover, making the phones initiate the connections means that the access point will be a slave in the piconet. This will require the access point to attempt to track the frequency hops of only one of mobile device, but not the others. The consequence of this is that the access point can make at most one connection at a time.

In addition, it is more practical if the access point coordinates the connections, because if every mobile phone tries to initiate a connection to the access point, the access point will be overwhelmed by a large number of connections. This causes a scalability issue. Hence it was decided that the access point would initiate the connections, while being the master, during all communication and thus it will be the device that inquires (scans) all other surrounding devices, while the mobile phone listens for incoming access point connection requests.

The second consideration is that as the state of the device cannot be in connection and inquiry mode at the same time. This means that the device cannot scan for other devices while handling a communication. This implies that they have to be done consecutively. This is reflected in our design later on, when that part of the code is executed sequentially.

6.2 Mobile Phone

6.2.1 Software Modules

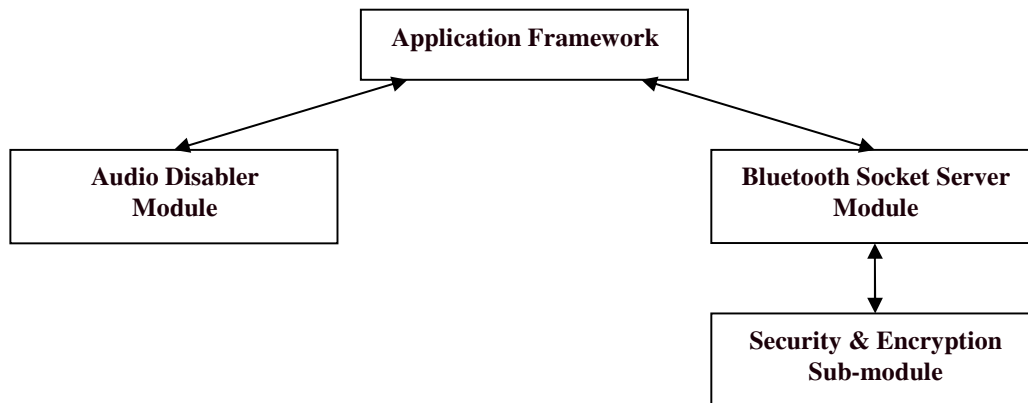


Figure 8: Software modules of mobile phone, and their independencies.

In order to simplify the problem, we broke the program down into several modules. Each module can be designed and implemented independently. Finally they can be integrated.

The mobile phone software consists of three main modules, and one sub-module.

The *Application Framework* is the driver of the software. It consists of methods to construct the graphical user interface (GUI). From there, the other modules can be easily added, and the GUI modified to incorporate the additional functionalities these modules add. The application framework also provides an interface for communications between each module. If, for example, the *Bluetooth Server Socket Module* needs to access a method in the *Audio Disabler*, it will call it via this framework.

The *Audio Disabler Module* provides a simple interface to the Beatnik audio libraries (see section 7.1.2). It simplifies the audio disabling procedures by hiding these procedures from the calling application framework.

The *Bluetooth Socket Server Module* runs a Bluetooth socket server process in the software. This is the module that allows the mobile phone to exchange data with the access point. The data consists of mainly authentication related data, which is processed by the *Security & Encryption Sub-module*. After authentication, this module continues to inquire the access point. This allows the mobile phone to know if it is still within the coverage area of the access point. As long as it is, this module will continue running. Once outside the coverage area, this module is restarted.

6.2.2 Top-Level State Diagram

A state diagram shows the various possible states of the program during operation, and how the program changes from one state to another. It is a tool to design the flow of a software system. Figure 9 shows a state diagram of the Bluetooth Server Socket module.

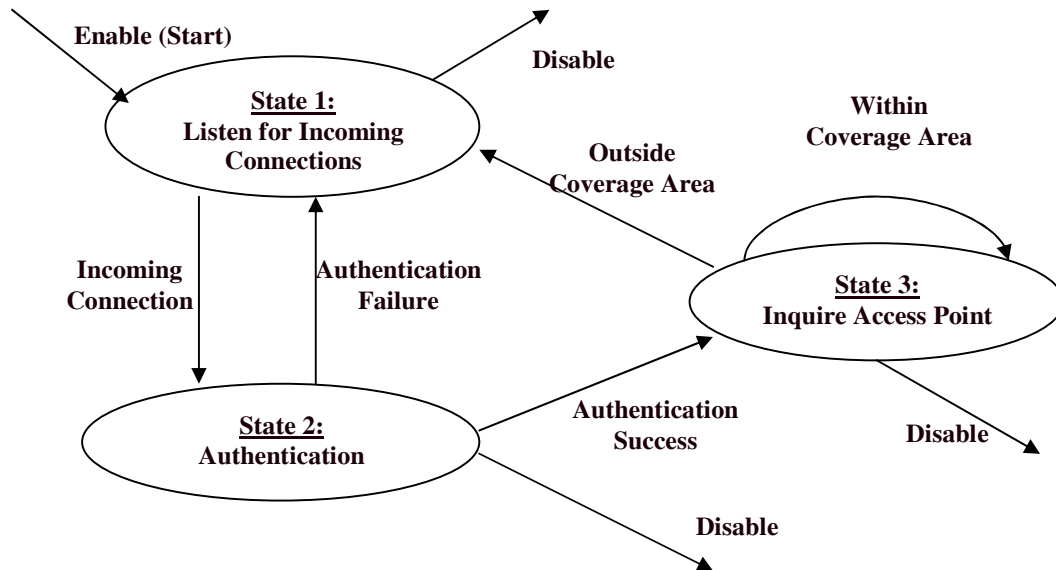


Figure 9: State diagram of the Bluetooth Server Socket module.

1. The ZoneIT software is enabled when the software is first run. This allows the software to listen for incoming Bluetooth connections to the phone (State 1).
2. If there is an incoming connection, the system moves on to the authentication phase (State 2), which is detailed in section 6.4. If authentication fails, the state machine is reset back to state 1, and the software continues to listen for new connections.
3. If authentication is successful, the state machine moves to state 3, where it will periodically inquiry for the access point, to determine if it is still within the coverage area of the access point. If it is, the state machine will remain in state 3. If not, the state machine is reset back to state 1, whereby it listens for a new connection.

4. The state machine can be terminated at any time if the user “disables” the software by shutting it down.

6.2.3 Inquiry State Diagram

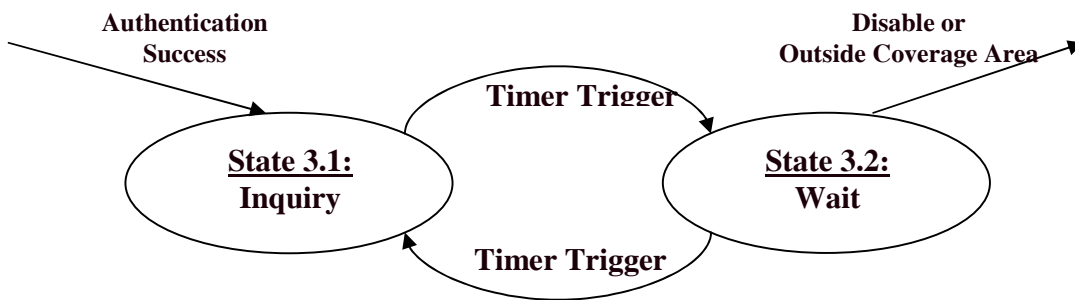


Figure 10: State diagram illustrating how the inquiry procedure is carried out.

- A timer trigger switches the state machine between the Inquiry and Wait states. In the Inquiry state, the software inquires the access point. In the Wait state, the software waits for a short moment, allowing other processes on the mobile phone to be carried out, so that the software does not hog all the resources of the system.

6.3 Access Point

6.3.1 Software Modules

The design of the access point starts with simple diagrams of how it interacts with other Bluetooth devices, and defining how it supposed to behave to satisfy the requirement. Then we proceed to define the required modules for the access point to function as intended. The diagram below shows the basic communication behaviour. For example, access point attempting to connect to devices which are not mobile phones will result in a “connection refused” error.

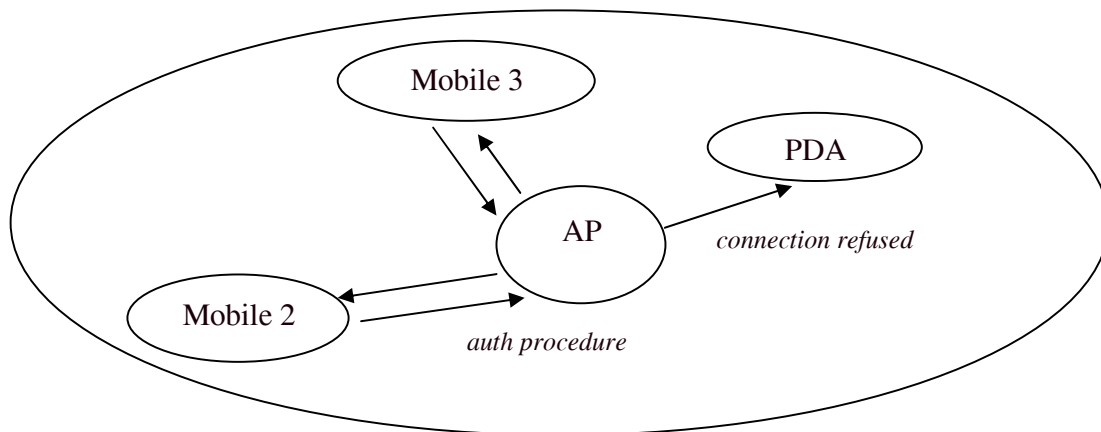


Figure 11: Relationship between the access point and mobile devices.

To reduce complexity and increase reusability, we introduce abstraction by use of modular design. That is we break the main problem into many smaller problems, and having each modules solving different problems.

As the design matures, it emphasizes more on security and efficiency, because we realize that the two factors are an essential aspect of any access control application.

The access point program is broken into the following modules:

1. ***Directory module*** acts as the record keeper of which phones has been deactivated, and when it was last seen.
2. ***Scanning module*** will discover surrounding Bluetooth devices.
3. ***Communication module*** provides Bluetooth communication to mobile phones.
4. ***Security module*** will encrypt and decrypt messages.
5. ***Main module*** will use all the other modules working together to provide the access point. This can be seen as the driver of the program controlling all the other modules in terms of synchronization of data, and program flow.

6.3.2 Data Flow Diagram

The modules interact by sending and receiving data from one module to other modules. A data flow diagram (DFD) will illustrate how the modules interact with one another.

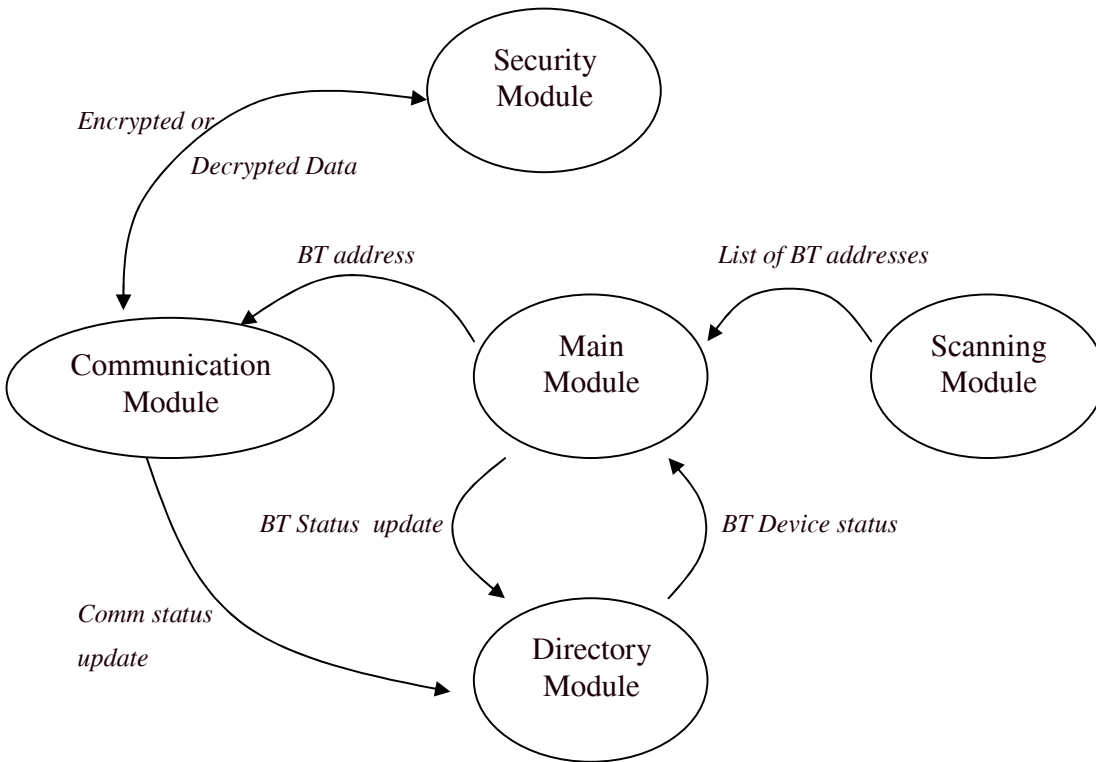


Figure 12: Data Flow Diagram of the access point modules.

The Scanning module and the Security module are dependant only on one other module. Hence, there is not much efficiency gained in terms of reusability, but it is desirable to introduce abstraction when coding for reasons mentioned earlier.

6.3.3 State Diagram

The following state diagram illustrates the various possible states the access point can have.

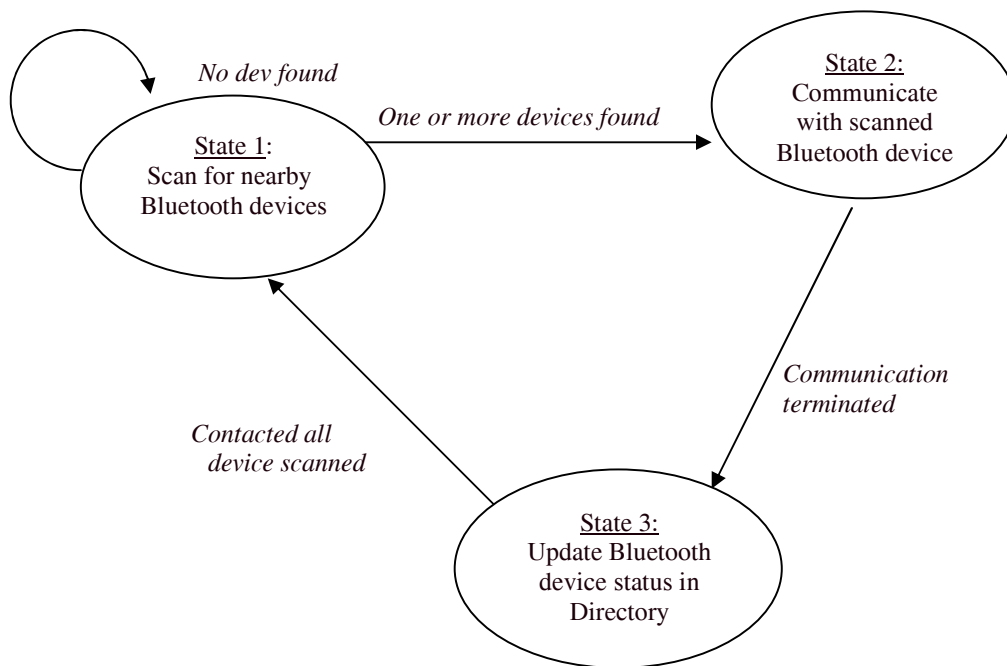


Figure 13: State diagram of the Access Point.

The diagram above only describes the high level states of the main module for the access point. In lower level, state 2 becomes more complex as parallelism is introduced by multithreading. Multi-threading will be discussed more in section 7.2.2.1.

The three states above can be seen as the lower layer of the main module, where the states are the procedure that the main module has to execute.

6.4 Authentication Protocol

The authentication protocol was designed to make the communications between the mobile phone and access point as secure as possible. The following diagram shows how a “nonce”, in a form of a random number, is used together with an encryption algorithm to secure the communications. The introduction of the “nonce” in the communications makes a playback extremely hard, if a sufficiently large “nonce” is employed.

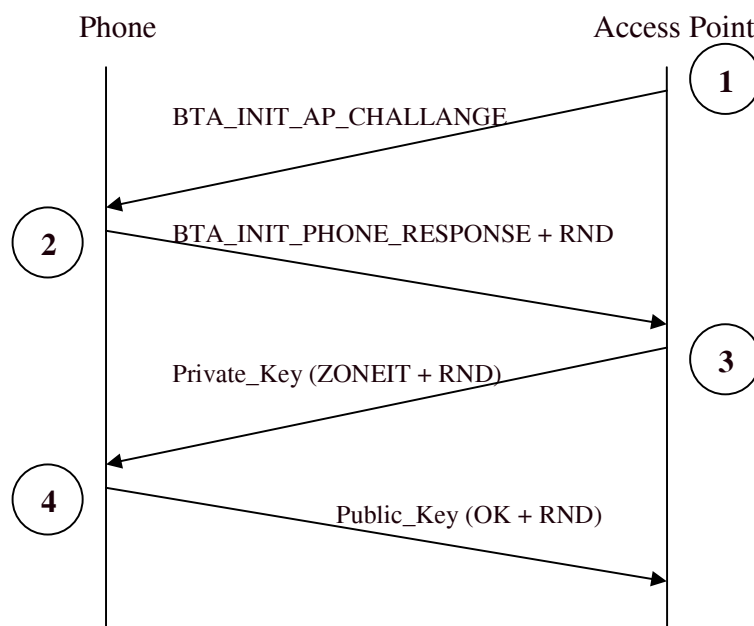


Figure 14: The figure shows the messages exchanged between the phone and the access point after a connection has been established between the two.

After the access point has initiated a connection to the phone, the following procedures are carried out in order for the mobile phone to authenticate the access point.

1. The access point issues a challenge (*BTA_INIT_AP_CHALLENGE*) to the mobile phone.
2. The mobile phone verifies the challenge. It generates an 8-bit random number (*RND*), and appends it to its response (*BTA_INIT_PHONE_RESPONSE*) to the access point, before sending the message to the access point. (See section 6.5.7 for more information about this random number.)
3. The access point will then request the mobile phone to turn off its ringing tone by appending the same random number (*RND*) to its request (*ZONEIT*), and then encrypting the whole message with its private key, before sending the message to the mobile phone. The encryption algorithm used is RSA, as described in section 6.5.2.
4. The mobile phone, on receiving the encrypted request, decrypts the request with the access point's public key. If the request is successfully decrypted, and the random number is correct, the phone then shuts off its ringing tone. It then generates an acknowledgement (*OK*) with the random number (*RND*) appended at the end, and encrypts this entire message with the access point's public key, and sends it back to the access point.

After these procedures, the mobile phone's ringing tone would be shut off, and the access point would note that that specific mobile phone has had its ringing tone shut off.

6.5 Security and Encryption

6.5.1 Public Key and Shared Key Cryptography

Public key cryptography is one whereby a user generates two pairs of keys for himself. One of these keys is termed a “public key”, which is made known to the public. The other is termed “private key” which the user keeps secret. These two keys are functional inverses of each other, meaning that if you use one to encrypt information, you will need the other to decrypt it. If public key cryptography is used in this system, the access point will possess the private key, while the phones will possess the public key.

Shared-key cryptography is generally used between two trusted parties only. These two parties share the same encryption key, which is also used as a decryption key. There are only two approaches to using this system in our system.

1. The access point only has one key. Everyone shares the same shared-key.
2. The access point stores a shared key for each user.

The first approach makes the system insecure. Since everyone knows what the encrypted key is, an attacker can simply use that key to encrypt and decrypt data moving across the network, allowing him to perform a denial-of-service attack easily.

The second approach makes the system not scalable to a large number of users. As the number of users in the system increase, the number of shared keys will increase. Moreover, introducing a new user to the system would require some form of “registration”, whereby the user is issued a new shared key. This makes the deployment of such a system cumbersome.

On the other hand, a public key cryptography solution will allow the public key of the access point to be hard-coded into the phone software, thus avoiding the scalability issue. The security issue is also avoided, since a malicious user only knows the public key of the access point.

6.5.2 Introduction to RSA

RSA is a public key encryption algorithm. The key length of the RSA algorithm is variable. Using a long key will provide a higher level of security, but requires more computation. Likewise, using a short key provides less security, but improves the efficiency of the algorithm. The most commonly used key length is 512 bits. [18]

There is no way to prove that RSA is secure. However, there is no evidence that anyone has managed to break the algorithm yet. RSA works on the fact that it is hard to factor a large number. Using the best known technique today to crack a 512-bit number would take literally thousands of years to complete. [18]

The reasons why our team chose RSA over other public-key algorithms were because RSA has been well-tested, and the source code is readily available (ease of implementation).

6.5.3 RSA Key Generation

The following steps are used to generate a private and public key pair [18].

1. Generate two large prime numbers, p and q .
2. Let $n = pq$
3. Let $\phi = (p-1)(q-1)$
4. Choose a small number, e , which is relatively prime to ϕ .
5. Find d , such that $de \% \phi = 1$
6. The public key is (e, n) . The private key is (d, n) .

Note that the operator ‘ $\%$ ’ in ‘ $x \% y$ ’ stands for ‘the remainder of x divided by y ’.

Both p and q should remain secret. Since p and q are large primes, the result key values will be large, and very hard to factorise.

6.5.4 Encryption and Decryption

The encryption and decryption of a message occurs as follows [18].

1. Let m be the message (plain-text), and $m < n$
2. The encrypted message (cipher-text) will be

$$c = m^e \% n$$

3. The decrypted message (plain-text) will be

$$m = c^d \% n$$

6.5.5 Using RSA in ZoneIT

This software system uses the RSA public encryption algorithm to encrypt its communications. However, implementing an algorithm with a large key strength is complicated. Available source codes on the web could not be ported over to the Symbian OS. Hence, we have settled for an RSA algorithm with a weaker key strength of 32-bits. Our focus in this project is not to implement the RSA algorithm, and hence a 32-bits algorithm would suffice to demonstrate our software system. Due to the modular structure of our design, switching to a stronger RSA key-strength would simply be a matter of replacing the encryption modules of the system.

6.5.6 Protecting the System - Denial-of-Service Attacks

A denial-of-service attack is a type of attack on the services of a resource, causing a user or organization to be deprived of the services of that resource they would normally expect to have [19].

In the case of our project, there are two services. One of the services is the ringing tone of the mobile phone. A malicious user can deny the user access to his ringing tone by using a Bluetooth device to imitate the functionality of an access point, forcing users of the phone software to shut off their ringing tones unnecessarily. Unfortunately, encryption the communications between the mobile phone and the access point is insufficient to defend against such an attack, because of the possibility of a playback attack, as detailed in the next section.

The other service is the service provided by the access point. The access point has to be robust, and resist attacks to take its service down. However this robustness will not rely on the authentication procedure but rather on the implementation of the access point itself, and hence will not be discussed under this section.

6.5.7 Protecting the System - Playback Attacks

A playback attack is one in which a malicious user monitors and records the traffic of a user, and then later “plays back” the same activity, possibly trying to trick the system into believing that the attacker is a valid user.

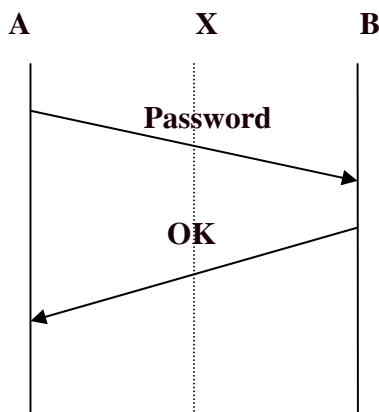


Figure 15: Shows an attacker X listening to the traffic between users A and B.

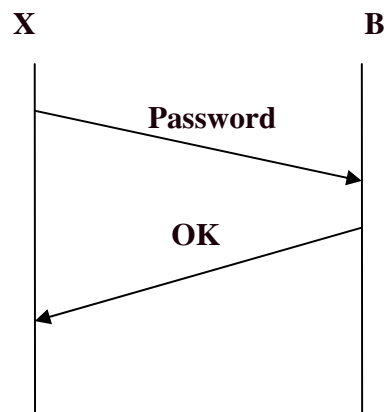


Figure 16: Shows the attacker X pretending to be user A by imitating the traffic sent from A at a later time.

As can be seen from figures 15 and 16, encrypting the messages between users A and B would not work, because the attacker X can just playback the encrypted messages and still be authenticated.

The only solution to this problem is to add some form of timestamp to the communications between A and B. For example, user A can append a timestamp at the end of the password, and then encrypt the message. User B, on receiving the encrypted message, can decrypt it and examine the timestamp. If the timestamp does not reflect the current time, then user B will know that the message originated from an attacker who is carrying out a playback attack. Because attacker X has no idea how to encrypt and

decrypt the message, there is no way he can ever carry out a playback attack on the system.

In this system, the timestamp can take two forms.

1. System time
2. Session ID / “nonce”.

Using the system time to create a timestamp is not difficult. However there is the issue of synchronisation of times between both parties. The success of such a timestamp would depend on the expiry time of the timestamp. If the timestamp is programmed to last for too long a period, a malicious user can take advantage by performing a denial-of-service attack before the timestamp expires. If the timestamp is programmed to last for too short a period, non-malicious traffic might be assumed to be malicious, and the authentication procedure cut short. Figures 16 and 17 illustrate the short-coming of using such a timestamp. The complexity of determining the expiry of the timestamp made our team decide against using such a timestamp.

The other alternative is to use a “nonce”, which is a random number. Provided the random number is sufficiently large, it would be hard for an attacker to attack the system. Moreover, we would not be faced with the difficulties of determining an expiry time for the timestamp.

Our system merely uses an 8-bit “nonce”, which is insufficient to defend against an attack, since an attacker can easily make a “guess” of the random number issued by the mobile phone. However, 8-bits are sufficient for demonstrating this software system. It would also not be difficult to add more bits to the “nonce” at a future date.

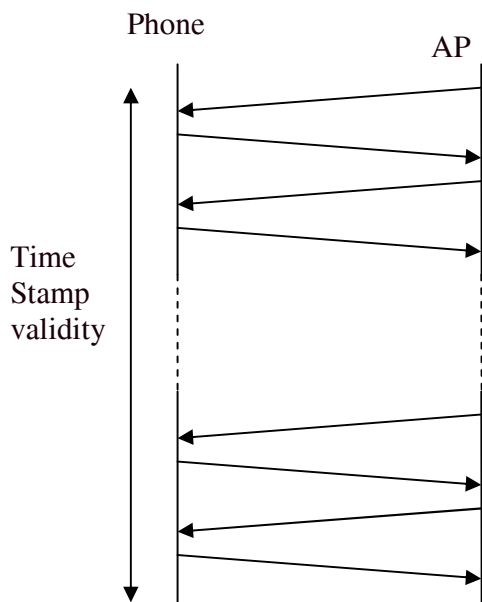


Figure 17: Time stamp expiring beyond the duration of a session. An attacker can use this excessive time to establish another session.

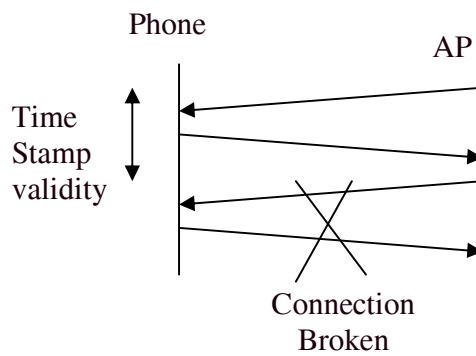


Figure 18: Time stamp expiring before the end of a session. Authentication will be forced to end pre-maturely.

7. Implementation

This section will first discuss the implementation of the mobile phone, followed by the access point. The difficulties faced during the implementation of each will also be briefly discussed.

7.1 Mobile Phone

7.1.1 Application Framework

The application framework is the minimal set of classes that is needed to write a functioning program on the Symbian Operating System. It usually includes classes that extend a number of parent classes. These classes are usually present in almost all programs for this system.

To speed up our development work, we utilised one of the basic 'HelloWorld' examples in the example codes provided by the SDK. This example merely provides a basic GUI with a basic menu, and displays the words 'Hello World' on the main screen. However, what was valuable to us was the application framework that was already in place. Basically, the entire 'HelloWorld' program was modified to suit our purposes, including the integration of the other modules into this framework.

7.1.2 Audio Disabler Module

This particular module is implemented as a bit of a kludge, although it is a critical module of the system. This is due to the fact that Sony Ericsson does not provide the required APIs to turn off the ringing tone volume of the mobile phone. However, our team managed to design a workaround to implement this functionality.

The mobile phone uses an audio driver to playback its ringing tones. By forcing the audio driver to playback an audio file we choose, and then muting the volume, the audio driver will be preoccupied with the audio file and hence cannot playback the ringing tone. Our team suspects that this could be a possible bug with the mobile phone, but unfortunately, this is all we can do to emulate the required functionality.

This workaround uses the Beatnik Audio Engine (BAE). BAE is the engine driving the audio capabilities of the Sony Ericsson P800, allowing the phone to playback polyphonic ringing tones [18]. BAE also provides an API for the programmer to playback polyphonic audio tones in their programs, called the miniBAE API. This API allows us to disable the ringing tone using the workaround in a simple manner.

However, the workaround also gave rise to another problem in the system. If the program on the phone is not running in the foreground, the audio volume of the mobile phone will not be shut off. However, it is sufficient to note that if Sony Ericsson had provided an API for us to disable the ringing tone of the mobile phone, we would not have to use this workaround in the first place, and hence we would not experience this problem.

7.1.3 Bluetooth Server Socket Module

This module runs the server program of the mobile phone, allowing incoming Bluetooth connections to the mobile phone. Writing up this module was not an easy task because of two things.

1. Setting up the connection was not straightforward.
2. The use of active objects.

Setting up the connection was not straightforward because of the high coding overhead in setting up the server sockets, including initialising the Bluetooth Security Manager of the phone. Another difficulty was the use of active objects in the code. Active objects are a type of encapsulation for multithreading in the Symbian Operating System, allowing the programmer to easily access multithreading functions without much complexity. The use of active objects was necessary because the graphical user interface (GUI) of the phone had to continue to function, even though the processor on the phone was busy with the Bluetooth program. If active objects were not used, the GUI would freeze. The complexity involving active objects was the understanding of how active objects work. This was not made any easier by some errors in the documentation of the API.

Moreover, the original plan was to include this module as two smaller modules. One module would listen for an incoming connection, while the other would do the device inquiry. However the use of active objects caused a race condition that forced us to integrate these two modules into a single module.

However, once these issues were solved, programming this module was rather easy. Basically, the module was designed as a state machine, and every aspect of it was modelled to the state diagrams in figures 9 and 10.

Implementing the inquiry procedure was easy. In fact, it was discovered that the procedure would involve inquiring the Bluetooth Manager on the mobile phone directly, since the Bluetooth manager does an automatic inquiry occasionally. The Bluetooth

manager caches inquired devices in its memory for approximately two minutes, to ensure that when devices are indeed out of range when they do not respond to an inquiry, and not because of an unstable link. This means that our software would not have to consider devices that do not respond due to an unstable link, since the Bluetooth manager already implements that.

7.1.4 Security and Encryption Sub-Module

Implementation of encryption is not the main focus of our project, and hence our team decided against spending too much time on implementing the actual code for this module. Our team managed to get hold of a few sources of RSA code. Most of these codes were complex, and spanned many files. We found it hard to install these files onto the access point, and impossible to install it on the phone, since these RSA code were in C, and the Symbian code was in C++. Symbian does not allow the running of C code directly. The way in which Symbian handles strings also made it hard to port these codes over.

Fortunately, our team also found a simple C++ implementation of a 32-bit RSA algorithm. The program included code on key-generations, encryption and decryption. This code was ported relatively easily over to Linux C for the access point, and also Symbian C++ for the mobile phone. Other improvements had to be made so that the encrypted messages could be forwarded to a Bluetooth socket easily.

It was decided that the public and private key pairs for the RSA algorithm would be pre-generated and hard-coded into the mobile phone and access point, so that we would not have to port the key generation algorithm code over to the mobile phone and access point. The only code we ported was the encryption and decryption algorithms.

7.2 Access Point

7.2.1 Scanner Module

The scanner modules will scan for any nearby Bluetooth devices. There is no maximum number imposed as to how many devices can be found, but we imposed a time limit for the scanning procedure. Theoretically, it is possible to scan all nearby devices within 10.24 seconds, provided that there is no error in data communication [26]. In practice, we are able to scan devices within 5-8 seconds most of the time, so we have decided to use 8 seconds as the time limit for scanning, in order to speed up the system.

The scanner module uses the HCI layer. As we have explored in section 2.9, the HCI layer act as the first interface a programmer has to the Bluetooth device. It provides the inquiry method that allows the scanning for other devices.

7.2.2 Communication Module

The communication module is one of the more important parts of the system. It executes communication procedures between the mobile phone and the AP. We also used the security module in the communication module, to allow secure communication. We've broken down the communication module itself into several functions.

Steps that the communication module takes are:

1. Create a socket and its options.
2. Bind the socket to a remote Bluetooth device.
3. Initiate the connection on the socket.
4. Initiate handshaking procedures, handshaking fails if remote device is not mobile phones.
5. Initiate authorization procedure. Fails if an encrypted 'OK' is not received.
6. Update the directory with the status of connection.

If any of the steps fails, then the communication jumps to step 6.

In order to improve the performance, we have tried to improve the efficiency of the AP by maximizing the use of resources available. This is done by introducing two concepts, multi-threading and synchronization.

7.2.2.1 Multi-threading

Threads can be defined as a lightweight process. Each thread is able to run independently from one another, and they can run at the same time (parallelism). Multi-threading happens when a single program uses multiple threads to run different section of the program at the same time [20].

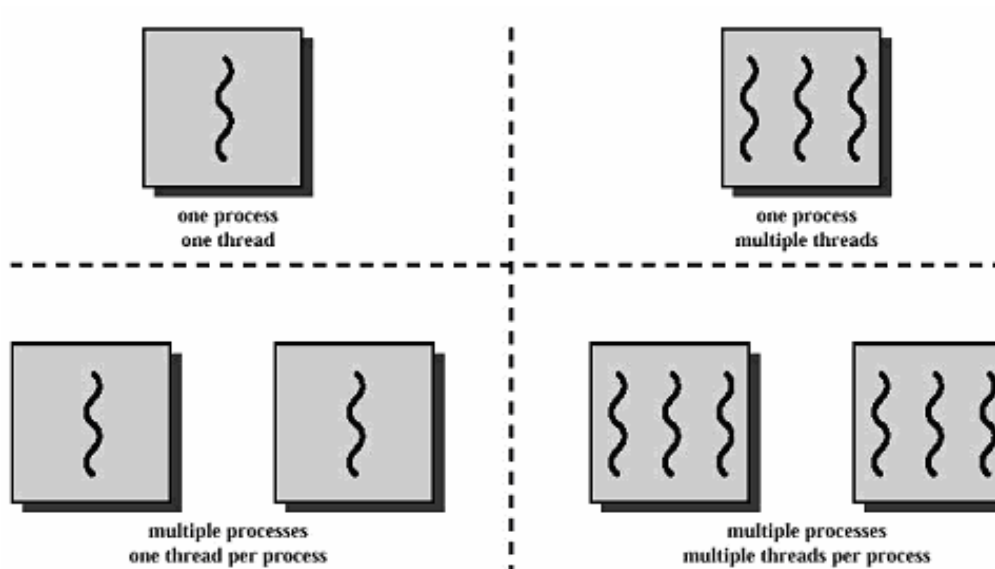


Figure 19: Comparison between threads and processes. (Diagram taken from [20])

Much of the delay in the access point program is caused by communication delays. For example, there are connecting delays and data transfer propagation delays. At these times, a process would be blocked, and processing resources are wasted by waiting. These blocking times can be better utilized.

With multi-threading, as soon as a thread receives a blocking call from awaiting incoming data, another thread runs in its place.

Although Bluetooth uses FH-CDMA (Frequency hopping – Code Division Multiple Access) to separate users, incoming and outgoing data are separated using TDD (Time Division Duplex) [21], which means they alternate with each other. The diagram below shows the data transfer between a master and a slave, when multi threading is not implemented in the software.

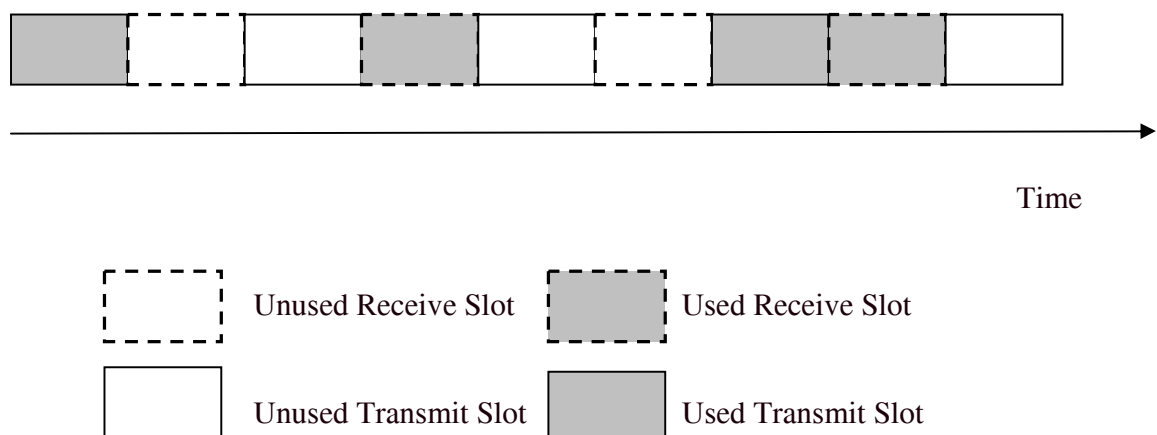


Figure 20: ‘Time-slot’ diagram, when multi-threading is not used.

A typical communications running our authentication protocol (see figure 14) involves two transmit, and two receive time slots, as depicted in figure 20. As we can see, there are unused transmit and receive slots in between used slots. These unused slots would be better utilized if multithreading is used, and these slots would be used by other threads to send and receive messages.

As mentioned in section 2.5, as we are allowed seven simultaneous connections with other Bluetooth devices. Thus, by applying thread programming with multi-threading to the communication modules, we can communicate with seven devices simultaneously, which will effectively maximize the use of the time slots shown in figure 20.

7.2.2.2 Synchronization

Threads in the same process are allowed to share the same data. This introduces complication such as the race conditions and synchronization problems. To avoid these problems, there are several techniques we can use.

Semaphores are effectively flags used to determine which event has occurred in a program. It is called when a thread is required to be synchronized with other threads. The calling thread will block (pause) itself until the synchronizing thread has signalled an event using semaphores.

Mutual exclusion (mutex) is another technique to allow for synchronization. It is similar to semaphores but it is more specific to resources. We use mutex when we want to protect resources from being accessed by two threads at the same time. For example, we can protect a data structure from being accessed by different threads at once. This is done by “locking” and “unlocking” the data structure.

The flow of the access point program can be coordinated by the main program without the use of semaphores. We use the `thread_join` calls to make the main thread wait for a sub-thread to finish before proceeding. This is useful as scanning has to be done in sequence with communication.

The status of addresses in the directory can be accessed by all threads. Our team chose to use mutex, and apply it to the directory such that only one thread is able to access the

directory at one time. No priority is given to any threads, as it is on a first come first serve basis.

Using mutex is important to protect the integrity of the data structures, namely the hash-table and the linked lists. For example, as a node is deleted, it is possible that a sleeping thread is still accessing that node. This causes a segmentation fault (crash).

7.2.3 Directory Module

The purpose of the directory is to increase the efficiency of the access point by making sure that it only communicates with devices that need to be contacted. This reduces redundant communication. An example of devices which do not need to be connected to are those devices that have been successfully connected to before.

To implement this, there is a need to include information such as the status of communication, and the time of last detection. Our team decided to implement a counter instead of timestamp, where each increment of the counter signifies one full iteration of scanning and communicating to mobile devices. To implement the directory, it was decided to use a hash-table that stores Bluetooth addresses and status of the addresses.

The directory also has a 'clean up' function, in which addresses are deleted when they are not used or seen during the scanning (inquiry) procedure.

A hash-table is a form of data structure that uses a lookup table which has fast access to an array of data. The index to the array is found from computing each Bluetooth address using a hash function.

The diagram below shall illustrate a perfect hashing, where no collisions occurred.

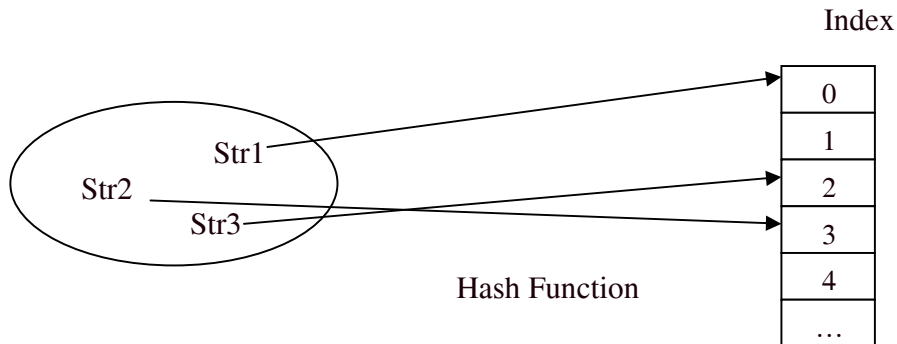


Figure 21: A hash table without any collision. Every string maps to its own index.

Collisions occur when there are different Bluetooth addresses that map to the same index [22], as illustrated in figure 22. As the input to the hash function is a string, we have to find a good string-to-integer function that gives rise to few collisions.

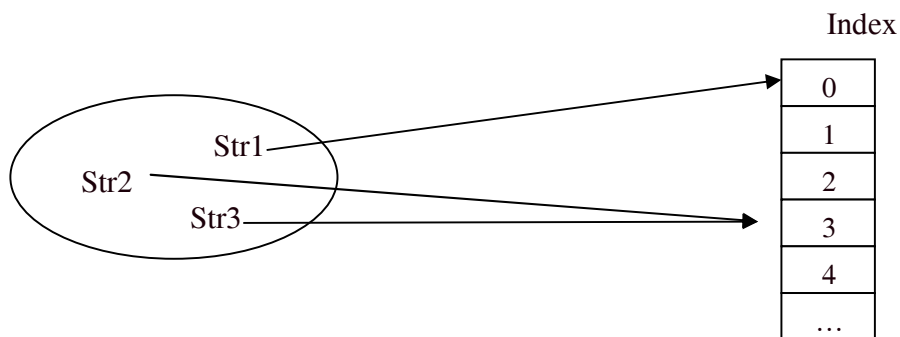


Figure 22: Collision between two strings.

To resolve the problem of collision, which is inevitable, we introduce a concept called chaining [22]. This is done by placing information into a node. A linked list is a type of data structure composed of nodes, which has one node pointing to the next. If there are strings that map to the same index, the node to be added will be chained to the end of the linked list.

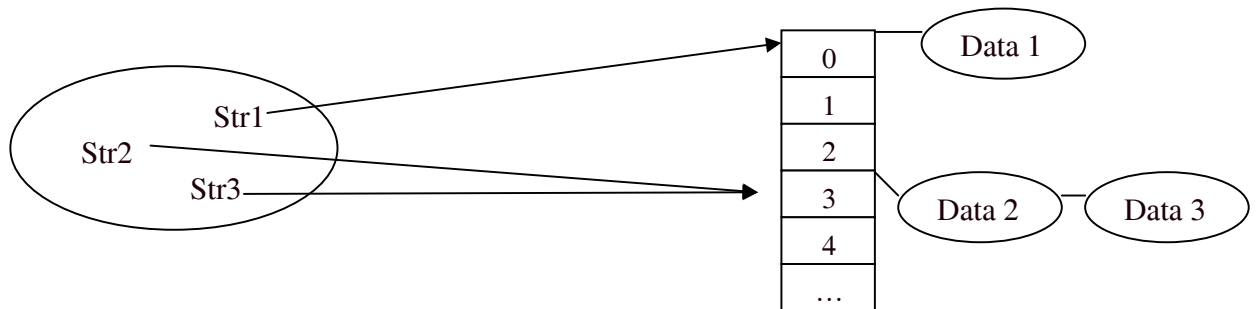


Figure 23: Illustration of the use of a linked list, should a collision occurs.

7.2.3.1 Time Complexity Analysis

In our implementation, we used double linked lists for the simplicity of the program. Time complexity is mostly determined by the hash function itself. If the number of hash-table slots is much greater than the number of entries, then on average, the time for insertion and deletion takes $O(1)$, which is equivalent to one insertion or deletion time with no collision [22].

7.2.3.2 Hash Function

A hash function is a function that maps any arbitrary object into an integer in the range of $[0, N-1]$, where N is the expected capacity of the hash-table [23]. A superior hash function for the directory is one that maps the Bluetooth addresses to the array indices with minimal or no collisions.

The hash function is divided into two parts. A hash code and a compression map. The hash code is a function that maps arbitrary object to a range of integer. The compression map is a function that maps the output of the hash code, which is integer, to the required range of $[0, N-1]$, where N is the size of the directory. This concept is illustrated in figure 24.

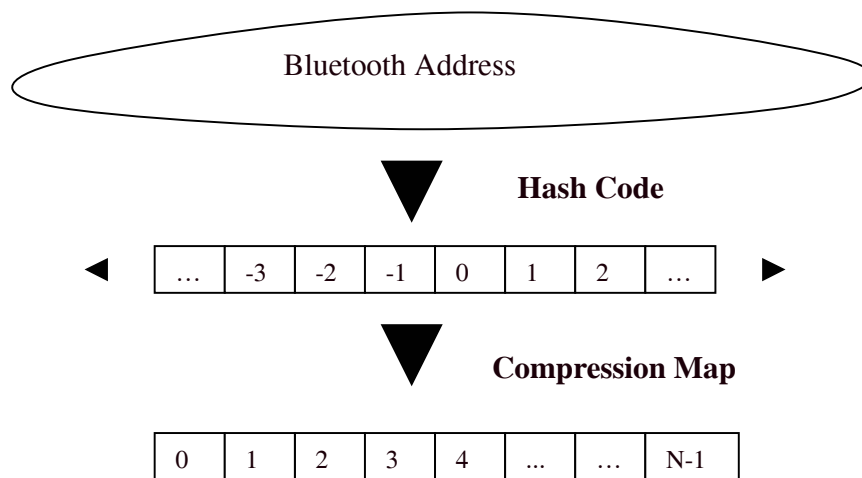


Figure 24: Illustration of a hash function.

The content of the strings are Bluetooth addresses. The addresses are made up of hexadecimal characters so there are 16 possible characters to choose from. This means that inside a Bluetooth address, there will be a lot of repeated characters. Doing a simple summing component hash algorithm to a string with repeated values, will give rise to many collisions.

The approach was to use polynomial hash-codes [23]. It consists of a function that converts the strings into long integers, and then adds some weighting to these integers. It has good immunity to repeated values since the order of the numbers affects the hash result.

7.2.4 Security Module

Just like the mobile phone, the access point is required to have some security measures. The security algorithm we have chosen is an asymmetric key cryptography known as RSA. This module will implement security for part of communications, by providing RSA encryption for the communication during authentication procedure.

7.2.5 Main Module

This is the controller module that coordinates all the other modules together to provide the final ZoneIT access point software. Its main tasks are to create and synchronise threads running these modules. In addition, it also executes the main algorithm that decides which Bluetooth device is to be connected. The overview of the state of the main module can be found in the state diagram of figure 13.

The main functions of this module are:

1. Scan all neighbouring devices using HCI layer commands.
2. Create threads to communicate with scanned devices. Number of threads must not exceed 7 at one time. It will wait for threads to finish before spawning more threads.
3. Consult the directory before creating a thread to make sure that the address were not connected already preciously.
4. Runs the cleaning up function to clean unused and older Bluetooth address in the directory.
5. Repeat steps 1 through 4.

8. Software Testing

8.1 Incremental Testing

Incremental testing is one in which code of a software system is tested as they are being written. Whenever the programmer can establish that his code is stable, he will save a copy of the code, so that he can fall back on that stable code should something go wrong with the code later. This form of testing helps the programmer to quickly identify problems in the code, debug them, and establish stable builds of the code. Our team has employed this technique in order to reduce the time spent on debugging. For example, in the testing of the communication module, our team started off with exchanging simple messages such as “hi” and “ok”, in order to ensure that the Bluetooth sockets of both the mobile phone and access points were configured correctly. After that, we moved on to implementing the protocol step-by-step. As in other programming projects, testing is mostly done by printing out the state of the program with “print” commands, or with a debugging tool. Debugging tools for the access point includes GDB and ‘printf’ statements.

8.2 Modules Testing

Modules testing is one in which a module is tested while being coded or after it is completely implemented, in order to test whether it is providing its intended functionality with no errors. Coinciding with our modular design, this means that we were able to test each module independently, and hence easily isolate problems in the system.

The sequence of tests done by our team is outlined as follows.

1. Inquiry modules: To test if the inquiry modules can detect other neighbouring devices.
2. Communication modules: To test whether communication between the access point and mobile phone is correct.
3. Security modules: To test whether messages can be correctly encrypted and decrypted at both ends.

This sequence corresponds to the order in which we completed each module.

8.2.1 Communication Modules Testing

During the communication testing, it was ideal to have several phones in order to carrying out testing. However, our team only had one phone available to us. Hence, we decided to emulate the phone program with another computer, running on the Linux operating system, equipped with a USB device (same setup as the access point). Of course, in order to save time, our team did not fully implement the features of the phone program, such as the encryption module. Our team merely implemented the challenge procedure, which was sufficient for our testing.

The emulated mobile phone did not work in the beginning because it was not detected by the access point. This problem was fixed by changing the settings of the Bluez configuration file. By modifying this file, we could allow it to be scanned, and be either slave or master.

After fixing the problem, the results turned out to be satisfactory. One of the most important aim of this test was to ensure that multithreading in the access point was working correctly. Our team saw from the print out messages that data was sent and received in an interleaving manner, from the phone and emulated phone, hence demonstrating the capabilities of the access point handling more than one Bluetooth device simultaneously.

8.3 Integration Testing

Integration testing is done by combining the modules and testing them as a whole. It is highly likely that integration testing of certain modules might fail, even though each of these modules passed the modules testing. An incremental approach was used during the integration testing. That is, modules are added to the system in an incremental fashion, so that we can tell which of the modules are giving problems, if one should occur. For example, we tested the system by adding the inquiry and communication modules to the system first. After everything was working fine, we added the security modules.

8.4 Final Testing

Final testing involved testing the entire software system as a whole. This was mainly done to iron out any last minute bugs that we may discover.

During the final testing, our team discovered that on some occasions, the phone could not detect (inquire) about the access point, and vice versa. This occurred when the two device were in inquiry mode at the same time (access point inquiring for other devices, mobile phone checking if access point still in vicinity).

To explain these occurrences, we refer to the section 2.8 above. It was explained that the inquiry procedure is done by increasing the hopping speed of inquirer to make sure it catches up with the other (inquired) devices. Simply put, if both devices are in inquiry mode, the two devices are both in accelerated hopping mode, and their frequency will not likely coincide.

This problem is more severe in the mobile phone, because in our tests, we had only one mobile phone. Thus the access point would be in the inquiry mode most of the time, and inquiring mobile phone would not be able to see it during those times. The problem is not as severe in the access point because the inquiry for the mobile phone is done by the

Bluetooth manager and it is done at some regular interval unknown to the programmer of the phone. This allows the mobile phone to be detected in between that time, which means that the programmer of the phone would not have to worry that the Bluetooth inquiry call would coincide with the inquiry process of the access point.

To solve this problem, some margin of error for detection of Bluetooth devices was added. In the case of the mobile phone, a counter is added to count how many times the access point has been undetected. It declares the access point to be out of range only after the counter has counted up to eight times consecutively. Similarly, for the access point, a counter (timer) is placed in each entry of its directory. The access point only regards a mobile phone to be out of its range if it goes undetected for five consecutive times. More tolerance is given to the mobile phone (eight times) as compared to the access point (five times), since the problem is more severe with the mobile phones. A sleep time was also added between each scan (in the access point) to give mobile phone more time to detect it. The sleep time is set to five seconds.

9. Conclusion

9.1 Achievements

During the course of the thesis, most of the design goals were met. The goal of demonstrating that Bluetooth can be used for access control applications has been demonstrated. Moreover, we have successfully secured the communications protocol between the mobile phone and access point, so that malicious users cannot abuse the service.

A major setback to this thesis was the fact that Sony Ericsson did not provide an API call to disable the ringing tones of the mobile phone, forcing us to have to design a workaround to achieve the equivalent functionality. In the process of using this workaround, the software will contain a problem which cannot be solved, as described in section 7.1.2.

However it is sufficient to say that the disabling of the ringing tones should not be used as a main criterion to judge the success of this project. The most important part of this project is that access control can be realised using short-range radio technology such as Bluetooth, and that such a concept can be extended to many other applications.

9.2 Commercial Viability

The ZoneIT software system is far from being commercially viable. In fact, the aim of this project was mainly to demonstrate that short-range radio technology, such as Bluetooth, can be used in access control applications.

It is also important to note that this software was developed in a period of less than one year. Most commercial softwares are developed over a period of several years, and undergo vigorous testing before they are released, to ensure that they are free of problems. The main emphasis of this project was to demonstrate the functionality of such a system. Our team has managed to ensure that the product is fully functional. However the standards of which we based on testing on may not be that of commercial standards. In fact, it is likely that we may have missed out on several test case scenarios in which a problem may occur in the software.

In addition to that, making this product commercially viable may require an industry-wide buy in, which may be hard to achieve.

Despite all these short-comings, ZoneIT is an interesting idea that may someday lead to a practical solution. The ability to control the function of a device based on its proximity to a Bluetooth radio could have a number of practical applications, as described in section 5.2.

9.3 Future Improvements

Future improvements to this system may include the addition of more “intelligence” to the software. One possible improvement is that, we could allow the phone to “learn” from previous activities. For example, the Bluetooth mobile phone will function as per our thesis, for the first few weeks. After this initial “learning” period, the mobile phone would be able to tell at which time of the week the user would enter a “silent” zone, and turn off the ringing tone of the mobile phone automatically, without the user having to turn on the Bluetooth radio of his mobile phone. This would allow significant power savings to the mobile phone.

The access point can also be further improved by designing it as a distributed system. Multiple Bluetooth devices can be combined to form a single “virtual” access point. This can be used to increase the coverage area of the system, should such a system be deployed in a large area, such as an exhibition hall. The access points can then be linked to one database (directory).

9.4 Final Conclusion

This thesis has demonstrated that a short-range radio technology such as Bluetooth can be used to implement an access control application. In this thesis, our team had selected one particular access control application, as detailed in the problem statement in section 5.1, to demonstrate this concept. As detailed in section 9.1, this project has been successful, although a commercially viable solution may takes several more years to complete.

Attached with this report is a compact-disc containing all the source code for our software system. These should allow future thesis students to do further work on our thesis, possibly developing a more commercially viable solution to the current system we have developed.

10. References

- [1] Au-System, “Bluetooth Whitepaper”,
[http://www.palowireless.com/infotooth/documents/Bluetooth_Whitepaper -
_AU_System.zip](http://www.palowireless.com/infotooth/documents/Bluetooth_Whitepaper_-_AU_System.zip)
- [2] Clarinet Systems, “Comparison of Infrared, 802.11b RF and Bluetooth”,
[http://www.clarinetsys.com/site/downloads-page/
Comparison_of_IR_802.11b_Bluetooth.pdf](http://www.clarinetsys.com/site/downloads-page/Comparison_of_IR_802.11b_Bluetooth.pdf)
- [3] Jaap C. Haartsen, Ericsson Radio Systems B.V., “The Bluetooth Radio System”,
IEEE Personal Communications, February 2000, pg 28-36
- [4] Per Johansson, Ericsson Research Manthos Kazantzidis, Rohit Kapoor, and Mario Gerla, UCLA, “Bluetooth: An Enabler for Personal Area Networking”, IEEE Network, September/October 2001, pg 28-37
- [5] Charlie Kaufman, Radia Perlman, Mike Speciner, “Network Security: PRIVATE Communication in a PUBLIC World, 2nd Edition”
- [6] Matt Ziegler, “An Overview of Bluetooth: Architecture, Power Consumption and Performance”,
http://www.ece.virginia.edu/~mmz4s/papers/ECE613project_bluetooth.pdf
- [7] John R. Hayes, “Modular Programming in C”,
<http://www.electronicengineering.com/features/software/OEG20011129S0051>
- [8] Crystal Sloan, “State Diagram Tips”,
http://www.ertin.com/pr_state_diagrams.html

- [9] Joe Flynn, Questra Corporation, “Communications Protocols for eAppliances: Bluetooth”
http://www.esconline.com/db_area/00fall/340.pdf
- [10] Wireless Communications Information & Services, “Bluetooth Specification; Bluetooth Technology Overview”,
<http://www.thewirelessdirectory.com/Bluetooth-Overview/Bluetooth-Specification.htm>
- [11] Microsoft Corporation, “Handheld PC 2000 SDK”,
<http://www.microsoft.com/downloads/details.aspx?FamilyID=aeaa7f4c-1331-46f1-bc4f-cc909333261a&DisplayLang=en>
- [12] CSR support, “Casira™ Bluetooth™ Development Kit datasheet”,
<http://www.csrsupport.com/document.php?did=23>
- [13] Microsoft Corporation, “Platform SDK update”,
<http://www.microsoft.com/msdownload/platformsdk/sdkupdate/>
- [14] Free Software Foundation, “The Free Software Definition”
<http://www.gnu.org/philosophy/free-sw.html>
- [15] Marcel Holtmann, “Bluetooth hardware support for BlueZ”,
<http://www.holtmann.org/linux/bluetooth/devices.html>
- [16] “The Symbian Programmers API Reference”
<http://www.symbian.com/developer/development/cppdev.html>
- [17] Bluez Project, “Bluez Frequently Asked Questions”,
<http://bluez.sourceforge.net/faq.html>

- [18] Symbian Community News 2003, "Beatnik and Sony Ericsson duet on the P800",
<http://www.symbian.com/news/2003/pr030212.html>
- [19] Whatis.com, "Denial of Service",
http://whatis.techtarget.com/definition/0,289893,sid9_gci213591,00.html
- [20] A. D. Marshall, "Programming in C: UNIX System Calls and Subroutines using C"
<http://www.cs.cf.ac.uk/Dave/C/node29.html#SECTION00291100000000000000>
- [21] Merry M., Bilsel M. et al, "Bluetooth Scatternet Router"
http://www.mattmerry.com/vlsi_proposal.pdf
- [22] Thomas H. Cormen, et al. "Introduction to Algorithms", 2nd Edition.
- [23] Goodrich M.T., Tamassia R. , "Data Structures and Algorithm in Java", 2nd Edition.
- [24] Bluez Project, "Programming Using Bluez",
<http://bluez.sourceforge.net/howto/node43.html>
- [25] Bluez Project, "Bluez",
<http://bluez.sourceforge.net/howto/index.html>
- [26] Palowireless, "Time Taken to complete Inquiry/Paging Procedures",
<http://www.palowireless.com/infotooth/knowledge/baseband/99.asp>

11. Appendix

11.1 Installing Bluez

Information to install Bluez into the system can be found in the Bluez web site [24, 25]. To install Bluez stack into our Linux Operating System we need to find the right installation files for a particular Linux version. The installation files come in packages, and a basic installation consists of 6 packages. We obtained packages for Linux kernel 2.4.21 (mandrake 9.1), and they are listed below:

```
[ ] bluez-bluefw-0.9-1.i.> 26-Dec-2002 15:30 80k
[ ] bluez-hcidump-1.5-1.> 20-Mar-2003 16:24 29k
[ ] bluez-libs-2.4-1.i38.> 20-Mar-2003 11:39 102k
[ ] bluez-pan-1.1-1.i386.> 20-Mar-2003 15:47 25k
[ ] bluez-sdp-1.1-1.i386.> 20-Mar-2003 14:08 300k
[ ] bluez-utils-2.3-1.i3.> 20-Mar
```

The Unix command to unpack and install the packages is:

```
rpm -Uhv bl*.rpm
```

Then we will add in /etc/modules.conf file the lines below:

```
alias net-pf-31 bluez
alias bt-proto-0 l2cap
alias bt-proto-2 sco
alias bt-proto-3 rfcomm
alias bt-proto-4 bnep
alias tty-ldisc-15 hci_uart
```

This is done to include those modules into the kernel. Finally, to activate the Bluez stack, we run:

```
hcid -f /etc/bluetooth/hcid.conf
```

11.2 Bluez Settings

The setting for the Bluetooth device is written in the file `/etc/bluetooth/hcid.conf`.

It allows the user to set particular options that is required. For example, to enable inquire scan mode, we set:

```
iscan enable;
```

The list of available options are placed as comments in the file, and they are quite self explanatory.

11.3 Source Code

The source code of this thesis can be found on the attached compact-disc.