portable linux distribution

Group Project Assignment



**Author:** Georgi Butev 3024421

**Senior Lecturer:** Stavros Dimitriou

**Class:** Network Operating Systems

**Course:** MSc Computer Systems and Networking (1024 FT)

**Faculty:** ESBE

**University:** London South Bank University, London, SE1 6LN.

**Email:** [butevg@lsbu.ac.uk](mailto:butevg@lsbu.ac.uk)

**Submission Deadline:** 11/05/2012

**Abstract**:

|  |
| --- |
| This report discusses the construction of a custom Linux based operating system distribution for educational purposes. In addition it details the process of designing (flowchart, UML class diagram, algorithms) and developing an encrypted network chat application which supports both RSA and DES algorithms. The RSA algorithm is used to encrypt line text messages with 2048 bit long public key. The DES algorithm is used to encrypt entire text files which can be sent through the network media. This report also describes some of the problems encountered during the developing and testing stages with the Linux distribution and chat program. |

Table of Contents

[1. Aims and Objectives 2](#_Toc323146662)

[2. Introduction 2](#_Toc323146663)

[3. Technical Approach (Design Considerations and Implementation) 3](#_Toc323146664)

[3.1 Application Layer 3](#_Toc323146665)

[3.2 Media Layer 4](#_Toc323146666)

[3.3 Services 5](#_Toc323146667)

[3.4 OS Layer 6](#_Toc323146668)

[3.5 Kernel and Device Drivers 6](#_Toc323146669)

[3.6 System Applications 8](#_Toc323146670)

[3.7 User Applications 9](#_Toc323146671)

[4. System and User Software Test 10](#_Toc323146672)

[5. RSA Chat Application 10](#_Toc323146673)

[5.1 The RSA algorithm 11](#_Toc323146674)

[5.2 UNIX sockets 12](#_Toc323146675)

[5.3 Application flow chart 13](#_Toc323146676)

[5.4 Application classes diagrams 14](#_Toc323146677)

[5.5 Application Algorithms 15](#_Toc323146678)

[5.6 Application Improvements 16](#_Toc323146679)

[5.6.1 The DES algorithm 17](#_Toc323146680)

[5.6.2 Comparison between RSA, DES, AES, Blowfish algorithms 17](#_Toc323146681)

[5.6.3 Sending and receiving encrypted files flow chart 17](#_Toc323146682)

[5.6.4 Sending and receiving encrypted files classes diagrams 17](#_Toc323146683)

[5.6.5 Sending and receiving encrypted files algorithms 18](#_Toc323146684)

[5.7 Final Design 19](#_Toc323146685)

[6. Results and Discussions 20](#_Toc323146686)

[6.1 Kernel Issues 20](#_Toc323146687)

[6.2 OpenSuse Studio Concerns 20](#_Toc323146688)

[6.3 Absence of GUI 21](#_Toc323146689)

[6.4 The choice between XAMPP and LAMP 21](#_Toc323146690)

[6.5 Chat application problems 21](#_Toc323146691)

[7. Conclusion 22](#_Toc323146692)

[8. Future Development 23](#_Toc323146693)

[List of References 23](#_Toc323146694)

[Appendix A 24](#_Toc323146695)

# 1. Aims and Objectives

The aim of the project is the development of a portable Linux based distribution applying knowledge gained through technical research of the Linux operating system structure and also Network Operating Systems class notes and lectures. The distribution is supposed to be portable because it would be able to boot-up and be functional on most personal computers using nothing more than a USB memory stick. The distribution would be Linux based because its kernel and associated programs are open source. This would make it possible to create and build a custom operating system according to the developer’s needs. Unfortunately this would be impossible using any UNIX variant because of its closed source nature and perplexing licencing options.

This project also aims at creating a useful above all Linux distribution targeted at the university student. This is because building such a Linux distribution would require a lot of effort and time but cost close to nothing. Thus it would be easily distributed at no cost to students interested in working with Linux on daily basis.

Hopefully this project would also bring social change within the student community in addition to technical solution. This is possible because of the open source software model and copyright model used by many Linux distributions. This would encourage the community to learn, experiment, modify, and improve the functionality and software of the operating system – something quite difficult to achieve with Mac OS X, Windows, and corporate UNIX. Also due to the free licencing, students would write off spending hundreds of pounds on software distributed and marketed by big IT companies that have created software monopoly. Such aim is also highly relevant in today’s difficult economic times.

To achieve the technical aims the following objectives would have to been fulfilled:

1. Draft schema design of the building blocks of a Linux distribution.
2. Compile, install, and configure all system applications.
3. Compile, install, and configure all user applications.
4. Test the functional OS.
5. Develop a small chat program which can encrypt / decrypt text message between two local or remote peers using the RSA algorithm.
6. Test the functionality of the script / program on different personal computers.
7. Prepare distribution medium – VMware / Virtual Box.

# 2. Introduction

In order to build an operating system the following features should be included – boot-up procedure, hardware device drivers, processes and thread management, CPU scheduling, memory management, access to IO and peripheral devices, network connectivity and tools, authentication and security, protection against viruses and malware, graphical user interface, system and user software.

Renowned pioneers and IT technology heroes such as Denis Ritchie, Ken Thompson, and Linus Torvalds have written their very own operating systems from scratch. Whether it was because Denis wanted to play Space Travel on the PDP-7 minicomputer or Linus wanted to create something more usable than DEC’s ULTRIX, the benefits for the IT industry and PC user were and still are immense.

This project is far less complicated, has a slightly different approach and target audience. It is less complicated because it will be built upon already existing know how and software solutions, namely the Linux kernel and related applications. The approach is slightly different because the final version of the OS will run from the ubiquitous USB dongle on majority of personal computers available to consumers. One of the keys to success is that the Linux community has developed countless drivers and tools for a plethora of hardware platforms, such as Alpha, Atari, Dreamcast, i386, ia64, Macintosh, PlayStation, Sparc, Sparc64, Sun, VAX, etc. The same cannot be argued about avid competitors such as Apple and Microsoft which operate only on selected hardware.

The target audience also differs; it is primarily for students who are willing to use Linux for their assignments, reports, research, and use of specialised software. There are several benefits to potential users of this project:

* Fully featured advanced OS compared to Microsoft Windows and Mac OS X.
* Enormous catalogue of software ready to be installed and used by the user for free.
* According to analysed information from [1] USB compares to HDD technology as follows:
  + Modern USB2 thumb drives outperform consumer range hard disk drives.
  + USB3 thumb drives outperform expensive server cluster hard disk drives.
  + USB thumb drives are a fraction of the size of HDD.
  + USB thumb drives are similar to HDD hot swap technology.
  + USB thumb drives are robust which means that if dropped they will not break.
  + Long term data retention which means that data can survive for as long as 10 years.

Amongst the top ten distributions (i.e. Mint, Ubuntu, Fedora, OpenSUSE) according to [2] there is none which is optimised for educational purposes and university students. Amongst the somewhat popular higher-education oriented distribution are listed below:

* Alinex developed by Universidade de Évora.
* ALT Linux popular with the Russian student community.
* EDUbuntu developed by Canonical.
* Guadalinex developed by Junta de Andalucía.
* MINIX with its 5000 lines of source code is popular with those who desire to understand how an operating system works looking at C and C++ code.

The above mentioned Linux distributions are distributed via ISO images which can be recorded on CDs and DVDs. In contrast this project is distributed via virtual machine image. This means that the user would simply load the image in VMware or Virtual Box and start using the operating system.

# 3. Technical Approach (Design Considerations and Implementation)

The system architecture of this portable Linux distribution has to follow the established convention and structure. The original UNIX developed by Denis Ritchie and Ken Thompson at AT&T would be a good starting point for quality reference. However it would be better to create a diagram of the operating system based on more modern version of UNIX or Linux. The OS components have been divided into several layers discussed in this report.

## 3.1 Application Layer

This is the top most layer of the structure. It exclusively deals with the user-computer interaction through the graphical user interface applications which use the popular WIMP (Windows Icons Menus Pointer) model.

Figure 1: Application Layer

## 3.2 Media Layer

This layer is necessary in order to decode, encode, and play various media type files. These include image formats (e.g. JPEG, PNG, EPS, BMP), audio formats (e.g. OGG, MPEG-3, MPEG-4, AAC), video formats (e.g. AVI, MPEG-1, MPEG-2, H.264), and web streaming formats (e.g. HTTP, RSTP, RTP). The layer would also have to support management of fonts, colours, printing (e.g. traditional UNIX CUPS, GIMP graphics editor files, PostScript files, Latex documents, Adobe PDF), gaming (e.g. OpenGL, OpenAl, OpenCL, CUDA), burning CD/DVD optical media, and playback of VCD/SVCD/DVD media.

Figure 2: Media Layer

## 3.3 Services

Services have little direct impact on the user interaction with the operating system. However their usage in UNIX and Linux like environments is common. Also known as background processes in Microsoft Windows, Linux distributions refer to them as daemons. According to [3] daemons are forked children of the init( ) process which runs as long as the system is up and perform low level tasks which output or resources are needed by high level user applications. For example the Apache web server which runs as a daemon on most Linux server setups, parsing PHP, Perl, and Python source code suitable for viewing by the remote user via his / her web browser.

**Network** daemons are essential part of any interconnected UNIX system which primarily deals with the TCP/IP stack. Most notable server services are Telnet, FTP, and electronic mail. According to [4] some of these daemons are – bootpd, fingerd, ftpd, gated (routing), httpd, imapd (mail), lpd (printing), sendmail, and smbd (connect to Windows network domain machines).

**Security** is usually the reason for big companies to use UNIX for their servers rather than what many consider as an insecure network operating system – Windows Server. This is accomplished by a combination of user privileges, access lists and rights, advanced firewall, and secure methods of remote users’ authentication. Furthermore very little viruses, worms, and spam bots target Linux users unlike Microsoft Windows. Another strong security feature is the use of encrypted shell for remote connections via SSH and SSL protocol unlike Microsoft Windows which still uses Telnet and Remote Desktop Connection.

**Task Scheduler** daemon performs regular mundane and repetitive tasks also known as cron jobs. This is an essential feature used both by professional system administrators and GUI users. Configuring cron jobs could be done using an application or by editing the configuration files located at /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly.

Figure 3: Services Layer

## 3.4 OS Layer

The operating system layer is the basis of the previous three layers. This layer includes essential system software to all modern UNIX systems without which services and user applications simply would not be able to operate. Some of the more important components are the **shell**, **package manager**, **curses**, and **X windowing server**.

**Bourne Shell** developed not surprisingly by Stephen Bourne for the UNIX system while he was working for AT&T is a low-level interaction between the user and the system’s kernel itself. The user can enter commands which will be translated into system calls (e.g. open( ), close( ), socket( ) ). The output of those commands would be displayed on the screen or printer. According to [9] there are several shells developed for UNIX operating systems that are still very popular and in use today – Bourn Shell (AT&T), Tenex C Shell (BSD), Korn Shell, and Bourne Again Shell (GNU). It is recommended to include all of these historic and modern shells in this project. According to [9] the user should be able to configure his/her profile, create command aliases, create task scripts, use commands completion, use commands history, use input and output redirections, run third-party applications (e.g. wget, ffmpeg, emacs).

It is important to note that shells were developed at a time when there were no personal computers and human-computer interaction was achieved via a TTY (teletype) terminal connected to a main frame. The terminals had no processing power and were connected to a printer. The terminals had buttons which required physical force unlike modern day keyboards. Thus the necessity of cryptic looking commands such cp (copy), mv (move), cat (concatenate).

Despite the popularity of UNIX curses and the improved version ncurses, the advent of **Xorg** and **XFree86** offered a lot more possibility for further improvement and development according to [10]. The Xorg server is so popular amongst the UNIX and Linux community that virtually all distributions use X server components except for Mac OS X which uses proprietary solution. Currently there are four major complete GUI suites implemented by BSD and Linux – Gnome (more popular for its special 3D effects engine Compiz), KDE (more popular for its Microsoft Windows like interface), XFCE (more popular for its minimal resources requirements), Java Desktop System (more popular for its futuristic Project Looking Glass). However these packages are very costly in terms of hard drive storage and in order to save disk space it would be better to install and configure only one of them.

## 3.5 Kernel and Device Drivers

Linux was written by the Finish student Linus Torvalds as an alternative to the industry standard UNIX. The project aims were to offer free to use and distribute software product with source code accessible by everyone (unlike AT&T’s closed source version). Even though it is a variant of the System V UNIX it follows prescribed rules and regulation. Thus is it very similar to other UNIX variants such as the BSD and Sun OS. These rules are part of two standards that form the UNIX specifications – POSIX.1-2008 and ISO/IEC 9899:1999.

The developer could read, analyse, and implement the standards in his / her operating system. Afterwards the organisation would evaluate the source code compliance with the following standards and potentially award UNIX certificate (Mac OS X Lion received UNIX 03 certificate).

* POSIX
* UNIX 03
* UNIX 98

The most vital segments of the kernel are listed below along with short description.

|  |  |
| --- | --- |
| Specification Name | Brief Description |
| User Management | Root and advanced user privileges defined in /etc/passwd and /etc/groups in the format:  account : password : user ID : group ID : home directory : user shell |
| Directory Hierarchy | / - > Root file system.  /bin - > Binary executable files.  /boot - > Kernel executable.  /dev - > Detected hardware devices.  /etc - > Host configuration settings for daemons, X, init( ).  /home - > Existing users’ ($USER) home ($HOME) directories.  /lib - > C programming language libraries and APIs.  /lost+found - > Recovered files if the system crashed.  /media - > Only mounting point / reference for CDs and DVDs.  /mnt - > Automatic system mounting point for devices.  /opt - > Software that does not comply with the GNU specs.  /proc - > Special virtual system for system processes.  /root - > The super user Root’s home directory.  /sbin - > System administration executables (e.g. ifconfig).  /srv - > Collection of service data (e.g. generated by CGI).  /tmp - > Storage for temporary files remove upon restart.  /usr - > User executables, libraries, data, etc.  /var - > Program’s variables such as logs and spools. |
| Regular Expressions | Evaluate and control data IO according to a selected text chunk pattern. Shell expansions use regex for mundane tasks such show all .c file (i.e ls \*.c). |
| File Access and Permissions | Security feature which uses three types of permissions (read, write, execute) to decide whether a user (owner, group, others) should manipulate a file. |
| ANSI C99 | AT&T UNIX, Linux Kernel, and many more system applications were written in the C programming language. C99 is the standardised version of AT&T C. |

Furthermore the Linux system should support the following functionality:

|  |  |
| --- | --- |
| Name | Brief Description |
| Protected Memory | Avoiding the possibility of applications overwriting each other’s data in the RAM. |
| Pre-emptive Multitasking | Achieve maximum CPU utilisation with tasks prioritisation, allowing kernel interrupts from IO or external devices. |
| Advance Virtual Memory | Mapping, un-mapping, paging, memory allocation, and addressing of the physical and virtual memory. |
| Real-time | Multitasking, multithreading, and low-latency during processing data. |

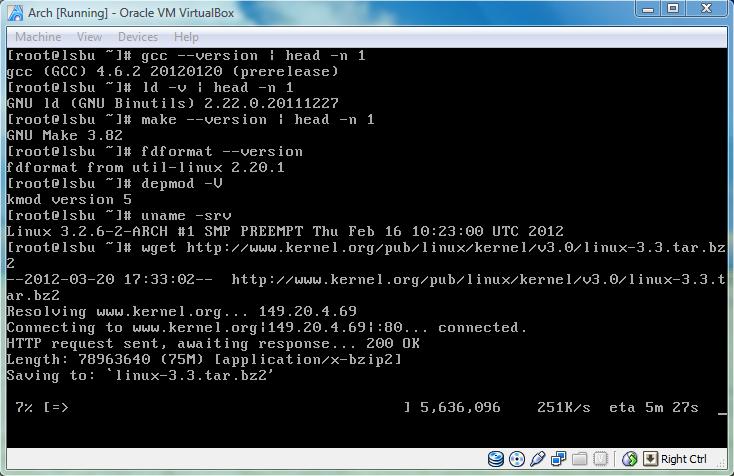
In addition the kernel would have to manage low level networking operations:

|  |  |
| --- | --- |
| Name | Brief Description |
| DHCP | The client can reques an IP address from the server pool of addresses. |
| DNS | Can translate human readable URI addresses into computer usable IP addresses. |
| FTP/SFTP/TFP | File transfer protocol for data exchange between an FTP server and client. |
| HTTP | Protocol used for browsing the Internet. |
| LDAP | Network resources, groups, and users locator. |
| SSH | The client can login into a remote system securely. |
| TCP/UDP/IP | Reliable connection-oriented and unreliable connectionless protocol using the IPv4 and IPv6 addresses to uniquely identify nodes on the network. |

In addition the kernel would have to manage the hard disk drive using a reliable file system. While Microsoft Windows still prefers the not so much advanced FAT32 and NTFS, Mac OS X advanced HFS, Oracle Sun’s extremely reliable ZFS, Linux prefers the journal file system Ext3 and Ext4. According to [3] ZFS used by FreeBSD, Sun Solaris, Mac OS X amongst other has better performance and superior journaling and recovery options.

## 3.6 System Applications

The system applications were installed on an already existing Linux distribution. Naturally the collection of these programs, tools, and kernel are the latest available unlike the ones supplied with the distribution. The process was straight forward. Establish the requirements, update the system kernel, update the shell and related applications, install and configure Gnome. Afterwards all user level programs were automatically compiled and configured by a packet manager such as synaptic, .deb (Debian and Ubuntu), .rpm (Red Hat and Fedora),and pacman (Arch Linux and Frugalware).



|  |
| --- |
| [root@lsbu linux]# tar xjf linux-3.3.tar.bz2  [root@lsbu linux]# ls –ld linux-3.3.tar.bz2  drwxrwx-w 23 root root 4096 Mar 19 05:15 linux-3.3  [root@lsbu linux]# cd linux-3.3  [root@lsbu linux-3.3]# make defconfig  [root@lsbu linux-3.3]# make  [root@lsbu linux-3.3]# make module\_install  [root@lsbu linux-3.3]# make install  Sh /root/linux /linux-3.3/arch/x86/boot/install.sh 3.3.0 arch/x86/boot/bzImage  System.map “/boot”  Cannot find LILO  [root@lsbu linux-3.3]# make kernelversion  [root@lsbu linux-3.3]# cp arch/i386/boot/bzImage /boot/vmlinuz-linux  [root@lsbu linux-3.3]# cp System.map /boot/System.map  [root@lsbu linux-3.3]# ls /etc/lilo.conf  ls: cannot access /etc/lilo.conf: No such file or directory  [root@lsbu linux-3.3]# ls –F /boot | grep grub  grub/ |

The 3.3 version of the Linux kernel was extracted from the bz2 archive using the tar command. Using the make command the kernel was configured (in this case with default settings). Another make command without an argument compiles the source code. Yet, another make command installs the compiled modules in /lib/modules. Due to an error that prevented make install, the new 3.3 kernel had to be copied to the /boot directory with make kernel version and cp commands. Finally the boot manager had to be configured with the absolute path to the kernel image, like such:

|  |
| --- |
| [root@lsbu grub]# nano /boot/grub/menu.lst  title LSBU Linux  root (hd0, 0)  kernel /vmlinuz-linux root=/dev/sda1 |

Alternatively the Syslinux boot manager could be used rather than GRUB (shown above). The advantage of Syslinux is that it is more advanced. The administrator can modify the boot menu (e.g. text based, graphical interface) and boot from media formatted in Ext2, Ext3, Ext4, and FAT32. The configuration file for Syslinux is located in /boot/syslinux/syslinux.cfg which was modified as follows:

|  |
| --- |
| [root@lsbu syslinux]# nano /boot/syslinux/syslinux.cfg  #UI menu.c32  UI vesamenu.c32  MENU TITLE LSBU Linux  MENU BACKGROUND splash.png  LABEL lsbu  MENU LABEL LSBU Linux  Linux ../vmlinuz/linux  APPEND root=/dev/sda1 ro  INITRD ../initramfs-linux.img  LABEL lsbu rescue  MENU LABEL LSBU Linux Rescue  Linux ../vmlinuz/linux  APPEND root=/dev/sda1 ro  INITRD ../initramfs-linux-fallback.img  LABEL off  MENU LABEL Shut Down  COMBOOT poweroff.com |

After the successful installation and default configuration of the kernel, system applications were installed. Thankfully downloading them, compiling, configuring, and installing them manually was not necessary. This is because Arch Linux uses a packet manager which automatically updates all system software (e.g. bash, nano, gcc). This included the Xorg window system and desktop environment, such as Gnome, KDE, and XFCE4. These are not monolithic applications but rather vast collection of programs and widgets that build on top of the X system. That is necessary because X does not offer files manager, access to system tools, start menu, and other tools most users are familiar with in Microsoft Windows. The resulting portable Linux was copied to the selected USB disk drive, using the following commands.

|  |
| --- |
| [root@lsbu ~]# mkdir /mnt/usbdrive  [root@lsbu ~]# mount /dev/disk/by-lable/IMATION /mnt/usbdrive  [root@lsbu ~]# mount –o loop /root/archlinux-2012.03.20\_14-30-01-core-i686 /mnt/archlinux  [root@lsbu ~]# cp –r /mnt/archlinux /mnt/usbdrive  [root@lsbu ~]# extlinux --install /mnt/usbdrive/arch/boot/syslinux/  [root@lsbu ~]# nano /mnt/usbdrive/arch/boot/syslinux/syslinux.cfg  archisodevice=/dev/disk/by-label/IMATION  [root@lsbu ~]# fdisk -l  Device Boot Start End Blocks Id System  /dev/sdb1 \* 62 212597 106268 83 Linux  /dev/sdb2 212598 740155 263779 82 Linux swap / Solaris |

The procedure described above is straight forward. First the USB thumb drive was mounted at the specified mount point. Next the Arch Linux distribution image was mounted to the specified mount point. Afterwards the contents of the ISO image were copied to the USB disk drive and the Syslinux boot manager was installed. The Syslinux configuration was modified to boot from the USB drive. Finally fdisk checked whether the file system was successfully created and it is bootable.

## 3.7 User Applications

There are 23338 FreeBSD ports (programs) and 9388 Arch Linux packages available for free. It would be unwise to search through, select, and install all of these user applications on the portable Linux project. That is why the project would offer only the bare minimum system applications and several user applications specific to web development and programming. This is because students enrolled in the Computer Systems and Networks academic programme at LSBU need such tools the most for their course work and software projects. This means the student would rely on:

|  |  |
| --- | --- |
| Text Editor | Nano, VI / VIM, Emacs, Diakonos. |
| Scripting | Python, Ruby, Perl, PHP. |
| Programming | GCC and Java OpenJDK. |
| Framework | Django (Python), Ruby on Rails (Ruby), Catalyst (Perl), Zend (PHP), Apache and MySQL, LaTex. |

# 4. System and User Software Test

Testing the system and user applications was a project milestone. In order to continue with the development of the RSA chat application, it was important to check whether the portable Linux system was properly configured.

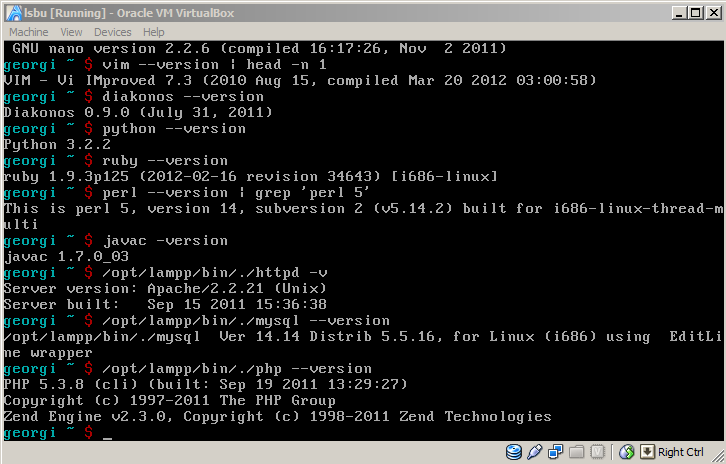


Figure 4: User applications and server daemons.

The first problem encountered was that the system was inaccessible because the network interface was behind NAT (Network Address Translation). To fix this issue, Virtual Box was configured to allow direct access to the NIC (Network Interface Card). The system remained out of reach because no IP address / network mask was configured. To fix this issue, two commands were executed – the first to install network utilities (e.g. ifconfig, ip, netstat) and the second dhcpcd which automatically assigned network address to the machine. Afterwards Internet Explorer successfully established a connection with the Apache web server which successfully served the already existing HTML web page. The second problem was that the system’s SSH and FTP servers denied access to PyTTY. To fix this issue, new accounts had to be created with user privileges for both.

# 5. RSA Chat Application

This application aims to demonstrate the purpose and effects of using RSA encryption. To do so, two local or remote peers exchange text messages. There are known issues when transmitting plain text across 802.3 (Ethernet) and 802.11 (Wireless) networks. Data can be intercepted by malicious users on the network. To avoid such case we could use public key cryptography using the RSA algorithm.

The message will be encrypted using public key pair of public exponent and modulus, while at the same time the remote peer can decode the message using a private key pair of private exponent and modulus. The two users can exchange plain text messages and encrypted messages using the keyword “secret” as a prefix.

The message will be encrypted using RSA algorithm with p (1024 bits long), q (1024 bits long), and modulus (2048 bits long). Such high bit rate is comparable to DES and AES encryption algorithms. According to latest research primes in the range of 512 bits are not secure at all and could be cracked by recalculation for a reasonable amount of time.

The user will not be prompted to type in p and q prime numbers and guess e exponents because such simplicity will produce RSA key as “strong” as the Caesar cipher. That is why the primes will be randomly generated using Java pseudo random class.

### 5.1 The RSA algorithm

It is public key cryptography method which uses two keys in asymmetric mode. The mode is asymmetric because the two users cannot use the same key to encrypt and decrypt the secret message. The public key is used to encrypt a numeric or plain text message with a public key pair. The private key is used to decrypt the numeric or plain text message with a private key pair. Such method is useful because it is not feasible to calculate either private or public key. Even fast performing central processing units in cluster configuration require unreasonable time span to recreate the exact key appropriate for decrypting the message. According to [12] randomly generated keys longer than 512 bits is enough for protecting sensible information. Also according to [12] keys of 1024 / 2048 are comparable in terms for cryptanalysis to AES and DES.

The brief mathematical algorithm follows the steps listed below:

1. Generate **p** and **q** which are two distinctly different random prime numbers.
   1. and
   2. and
2. Calculate their modulus or the product of their multiplication **N**.
   1. and
3. Select an integer **e** that satisfies the following conditions:
4. Calculate **d** that satisfies the following conditions:
5. Encrypt the **message** using public key pair to create **cipher text** as follows:
6. Decrypt the **message** using private key pair to create **plain text** as follows:

Selecting an appropriate **e** could be a serious challenge for the computer’s processing power. That is why protocols (e.g. SSL) and programming languages crypto libraries recommend the usage of the Chinese Remainder theorem which is much more efficient compared to.

1. Similar to the original p and q are primes.
   1. and.
   2. .
2. To calculate the secret the following steps are taken:
   1. and .
   2. .
   3. .

Another practical issue is that the message can only be a string of numbers. In order to transmit meaningful text information, the algorithm requires PKCS scheme implementation. The Public Key Cryptography Standard offers a solution to encoding plain text into digits which can be encrypted and afterwards be decrypted and decoded from digits to ASCII code which can be translated into human readable letters from the English alphabet. According to the JDK’s documentation and [13] there are several padding schemes in use, notably PKCS#5 and PKCS#7. The first one uses 8 bytes which is sufficient for the 128 characters (including non-printable) ASCII table reference. Whereas the second one is used by extended block ciphers (more than 8 bytes).

It is worth to note that **e** (public exponent)can be 3, 17, or 65537 calculated by also known as Fermat’s prime numbers. Using these instead of generating new ones each time the algorithm is processed, saves time and system resources. In addition the Euler totient function can be substituted with the Charmichael’s function as recommended by [13].

RSA is considered highly secure asymmetric public key cryptographic method. With a key size of 32 bits it would take approximately 35 minutes to break the code. However it would take approximately 1142 years to break a key that is 56 bits long. For example the following procedure demonstrates how to refactor the public modulus N and find the private key:

1. N = 119 which yields 7 and 17.
2. Thus the private key is KR =

Other common cracking techniques that have proved quite successful in compromising RSA encrypted data are listed below:

* Dictionary attack is useful if there is a correlation between the cipher text and the plain text.
* Common modulus attack – the modulus is used by more than one user.
* Blinding attack exploits RSA signatures.
* Wiener’s attack which can recover **d** exponent if and

### 5.2 UNIX sockets

The UNIX BSD API is the main tool for network programming. Unlike the C programming language (which makes UNIX/Linux system calls) Java and C# have their own libraries to interact with the TCP/IP model. The developer need only be concerned with the application layer even though a socket constitutes a pair of Transport Control Protocol (Transport Layer), Internet Protocol version 4/6(Network Layer), and port number which identifies different processes from a single identifying IP address. The following table lists the network stack as outlined by the BSD documentation:

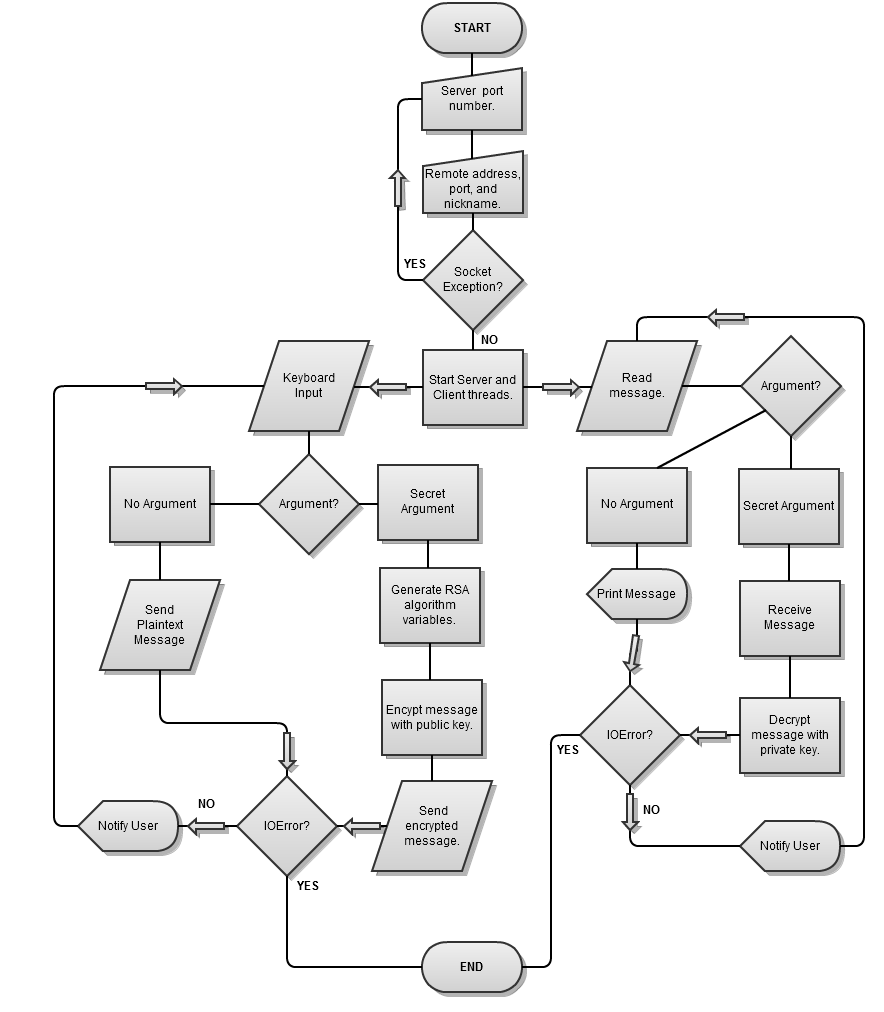
|  |  |
| --- | --- |
| **Name** | **Brief Description** |
| Socket | Manage open/close sockets in kernel space. |
| Domain | IPv4 (PF\_INET), IPv6 (PF\_INET6), and local (PF\_LOCAL, PF\_UNIX). |
| Protocols | TCP, UDP, ICMP, IGMP, RAW. |
| Interface | ifconfig command lists all available network interfaces such as en0, eth0, eth1, lo0. |

There are several Java libraries that can bind to this API on Windows, UNIX, and Linux listed below.

|  |  |
| --- | --- |
| **Name** | **Brief Description** |
| java.net.Socket | Endpoint communication between two local or remote peers on a computer network. |
| java.net.Inet4Address and java.net.Inet6Address | IPv4 and IPv6 address range. |
| java.net.NetworkInterface | Manages hardware network interfaces (eth0). |
| java.net.ServerSocket | Responsible for accepting client socket requests. |
| java.net.SocketException  java.net.PortUnreachableException | Useful try and catch block exceptions for determining network connectivity problems. |

### 5.3 Application flow chart

The figure below illustrates the general principal of operation for this application.



### 5.4 Application classes diagrams

The following figures illustrate the Java source code UML class diagrams.

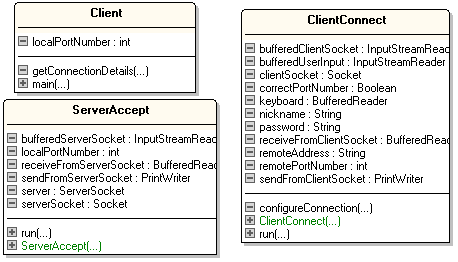


Figure 6: Client, ClientConnect, and ServerAccept classes UML diagrams.

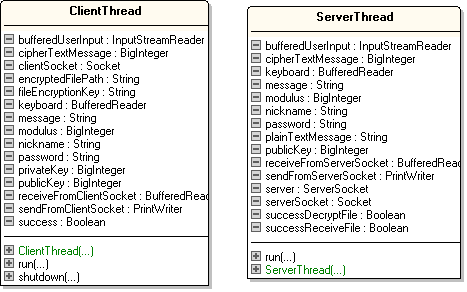


Figure 7: ClientThread and ServerThread classes UML diagrams.

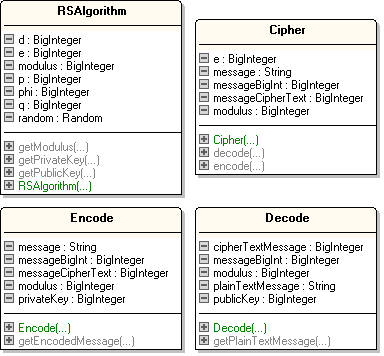


Figure 8: RSAlgorithm, Cipher, Encode, and Decode classes UML diagrams.

### 5.5 Application Algorithms

The following code samples demonstrate the most important routines performed by the application.

The code snippet below follows the already outlined RSA algorithm. Prime numbers are generated with a random seed of 1024 bits. Their product is derived from the “multiply” method rather than \* because BigInteger was used which is necessary for mathematical operations (e.g. GCD, mod, modPow) and precision. To represent the number 1, the valueOf( ) method was used because BigInteger arithmetic does not work with plain integers. A loop is necessary to find public exponent e which satisfies the condition, which in some cases may take plenty of time even when the Chinese Remainder theorem is used. Finally the private exponent is calculate and the “if” condition makes sure that it is correct. These values are used by the Encode and Decode classes to manipulate the plain text message and cipher text message.

|  |
| --- |
| random = new Random();  p = p.probablePrime(1024, random);  q = q.probablePrime(1024, random);  modulus = p.multiply(q);  phi = (p.subtract(BigInteger.valueOf(1)).multiply(q.subtract(BigInteger.valueOf(1))));  do{  e = e.probablePrime(2048, random);  } while((BigInteger.valueOf(1).compareTo(e) < 0)  && (e.compareTo(phi) < 0)  && (e.gcd(phi).equals(BigInteger.valueOf(1))));  d = e.modInverse(phi);  if((BigInteger.valueOf(1).compareTo(d) < 0) && (d.compareTo(phi) < 0)){} else{System.out.println("Incorrect public exponent!");} |

The code snippet below is responsible for encoding the text message with the supplied cipher. Since the algorithm encrypts digits rather than text, the message is represented by BigInteger using the getBytes( ) method. The program then checks whether the conversation satisfies the condition. Finally the BigInteger is encrypted using the modPow function from the java.math library using the already generated / calculated public key pair (i.e. public exponent and modulus).

|  |
| --- |
| messageBigInt = new BigInteger(message.getBytes());  if((BigInteger.valueOf(1).compareTo(messageBigInt) < 0)  && (messageBigInt.compareTo(modulus) < 0)){} else {  System.out.println("Incorrect message conversation!");  }  messageCipherText = messageBigInt.modPow(publicKey, modulus);  return messageCipherText; |

The code snippet below is responsible for decoding the text message with the supplied cipher. The message is decrypted using the supplied private key pair (i.e. private exponent and modulus). Since this is only BigInteger it has to be converted into readable text using the toByteArray( ) method.

|  |
| --- |
| messageBigInt = cipherTextMessage.modPow(privateKey, modulus);  plainTextMessage = new String(messageBigInt.toByteArray());  return plainTextMessage; |

The code snippet below is responsible for receiving and deciding what to do with the message. This is part of the ServerThread class which only receives data from the remote host. If the message is empty, this means that the remote host is no longer connected to the network. If the message starts with the keyword “secret”, the nickname is collected first, then the decode class is called with three arguments – modulus, private key, and the encrypted text. The decrypted text is printed out.

|  |
| --- |
| message = receiveFromServerSocket.readLine();  if(message == null){  System.out.println("The connection was terminated by the remote peer.");  System.exit(0);  } else if(message.startsWith("secret")){  nickname = message.substring(7);  cipherTextMessage = new BigInteger(receiveFromServerSocket.readLine());  Decode decode = new Decode(modulus, privateKey, cipherTextMessage);  plainTextMessage = decode.getPlainTextMessage();  System.out.println("\n" + nickname + ": " + plainTextMessage);} |

The code snippet below is responsible for sending the message typed by the user. If the user input matches the keyword “exit”, the application will be shut down and the remote host notified. If the user input begins with the keyword “secret”, a preamble message is sent along with the nickname. The message is passed as an argument in the Encode class along with the public key pair (i.e. public exponent and modulus). Finally the message is sent in encrypted form.

|  |
| --- |
| if(message.matches("exit")){  shutdown();  System.exit(0);  } else if(message.startsWith("secret")){  sendFromClientSocket.println("secret" + nickname);  String msg = message.substring(7);  Encode encode = new Encode(modulus, publicKey, msg);  cipherTextMessage = encode.getEncodedMessage();  sendFromClientSocket.println(cipherTextMessage); |

### 5.6 Application Improvements

The length of the cipher key pair is of great importance. If the key is too short it will be easy to crack the code in very little time. On the other hand if the key is too long it will be impractical to memorise. According to JDK 1.4.2 documentation, p and q prime numbers have to be at least 512 bits long (1024/2048 recommended). The product N will be too long to write down, let alone remember. That is why the functionality of the application was modified as follows. First both peers can exchange plain text messages. Second they can use secure RSA cryptography, which is implemented only as demonstration of the algorithm. Third they can send / receive text files encrypted with DES, CBC, PKCS5 padding scheme which requires easy to memorise 8 character long password. This might be much more convenient because the text file size is unlimited while at the same time the password is only 8 bytes long.

### 5.6.1 The DES algorithm

Unlike the RSA algorithm the Data Encryption Standard algorithm is fairly complex to be briefly explained in few consecutive steps. In fact DES necessitates 16 permutations in total. At each repetition the algorithm takes in 64 bit blocks of text which can be either substituted or transported. According to [14] DES makes sure that the output bits are obfuscated so that the input bits bear no resemblance because “substitution provides the confusion, and transportation provides the diffusion”. This algorithm and its derivatives are part of javax.crypto library which makes it easy to implement into any application.

### 5.6.2 Comparison between RSA, DES, AES, Blowfish algorithms

It is worth to note that RSA, DES, and Triple DES are similar in terms of the security they provide to plain text data. However AES-256 bit length is only comparable to 15360 bits RSA key length, which confirms how inefficient the RSA algorithm is. In addition the Advanced Encryption Standard (aka Rijndael algorithm) requires less CPU and memory utilisation, it is easier to implement, and it is approved for usage by the USA government for confidential documents. AES was developed to replace DES and overcome some of its limitations and security issues. Blowfish is similar to AES but more apt for software developers because it is an open source project and many programming languages have bindings for it.

### 5.6.3 Sending and receiving encrypted files flow chart

The figure below illustrates the general principal of operation for this application improvement.

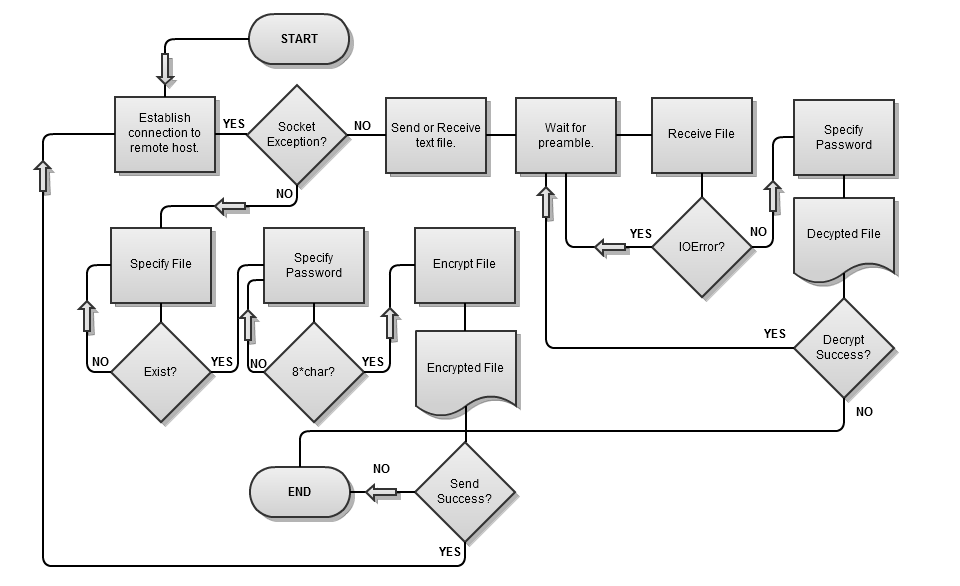


Figure 9: Send or Receive encrypted files flow chart.

### 5.6.4 Sending and receiving encrypted files classes diagrams

The following figures illustrate the Java source code UML class diagrams.

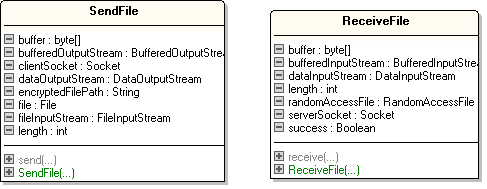


Figure 10: SendFile and ReceiveFile classes UML diagrams.

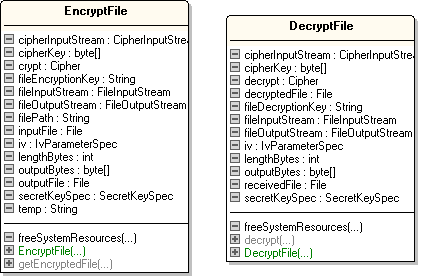


Figure 11: EncyptFile and DecryptFile classes UML diagrams.

### 5.6.5 Sending and receiving encrypted files algorithms

The following code samples demonstrate the most important routines performed by the application.

The code snippet below is responsible for sending the already DES encrypted file to the remote host. The file object is manipulated with the DataOutputStream (i.e. writing) and FileInputStream (i.e. reading) classes. The file is sent using a buffer of 1024 bytes for efficiency until EndOfFile (i.e. -1). The buffer is flushed at every other iteration of 1024 bytes.

|  |
| --- |
| file = new File(encryptedFilePath);  bufferedOutputStream = new BufferedOutputStream(clientSocket.getOutputStream());  dataOutputStream = new DataOutputStream(bufferedOutputStream);  fileInputStream = new FileInputStream(file);  buffer = new byte[1024];  length = 0;  do{  dataOutputStream.write(buffer, 0, length);  dataOutputStream.flush();  } while((length = fileInputStream.read(buffer)) != -1); |

The code snippet below is responsible for receiving the encrypted file from the remote host. The file object is manipulated with the DataInputStream (i.e. reading) and RandomAccessFile (i.e. writing) classes. The benefit of the latter class is that read, write, and index pointer are supported. At each iteration 1024 bytes of data are written to the user’s hard disk drive. This is still an encrypted file.

|  |
| --- |
| bufferedInputStream = new BufferedInputStream(serverSocket.getInputStream());  dataInputStream = new DataInputStream(bufferedInputStream);  randomAccessFile = new RandomAccessFile("received.des", "rw");  buffer = new byte[1024];  length = 0;  do{  randomAccessFile.write(buffer, 0, length);  randomAccessFile.skipBytes(length);  } while((length = dataInputStream.read(buffer)) != -1); |

The code snippet below is responsible for encrypting a user specified text file. The file object is manipulated by FileInputStream (i.e. reading), FileOutputStream (i.e. writing), and CipherInputStream (reading with cipher processing). The cipher constructor requires crypto scheme such as DES, Cipher Block Chaining or Electronic Code Book, and appropriate padding – PKCS5. This scheme was recommended by [14]. Afterwards the ENCRYPT MODE is selected with the 8 character long password key and initialising vectors. Similarly to previous code samples, 8 bytes of buffer are written to the file.

|  |
| --- |
| crypt = Cipher.getInstance("DES/CBC/PKCS5Padding");  crypt.init(Cipher.ENCRYPT\_MODE, secretKeySpec, iv);  fileInputStream = new FileInputStream(inputFile);  cipherInputStream = new CipherInputStream(fileInputStream, crypt);  fileOutputStream = new FileOutputStream(outputFile);  outputBytes = new byte[8];  lengthBytes = cipherInputStream.read(outputBytes);  do {  fileOutputStream.write(outputBytes, 0, lengthBytes);  lengthBytes = cipherInputStream.read(outputBytes);  } while (lengthBytes != -1); |

The code snippet below is responsible for decrypting the received text file. Similarly to the encrypt algorithm the cipher is constructed and the mode is set to DECRYPT. It is important to note that the initialising vectors have to be exactly the same. As a workaround this project constructs IVs with the password. Afterwards the file is stored as a plain text file using a loop with 8 bytes of buffer.

|  |
| --- |
| decrypt = Cipher.getInstance("DES/CBC/PKCS5Padding");  decrypt.init(Cipher.DECRYPT\_MODE, secretKeySpec, iv);  fileInputStream = new FileInputStream(receivedFile);  cipherInputStream = new CipherInputStream(fileInputStream, decrypt);  decryptedFile = new File("decrypted.txt");  fileOutputStream = new FileOutputStream(decryptedFile);  outputBytes = new byte[8];  lengthBytes = cipherInputStream.read(outputBytes);  do{  fileOutputStream.write(outputBytes, 0, lengthBytes);  lengthBytes = cipherInputStream.read(outputBytes);  } while(lengthBytes != -1); |

### 5.7 Final Design

The final version of the system architecture diagram has been modified to reflect the changes made to the OS due to problems during development.

* Software development frameworks were added – Ruby on Rails, Django, Zend, and LAMP.
* Gnome and KDE were removed because there was no free space left on the USB drive.
* SUSE Studio was not used because of its heavy branding and no options for installation on USB devices. Instead Arch Linux was used which is fully customisable, has low impact of disk drive space, memory, CPU, and can be installed on a USB thumb drive.
* The distribution model was changed to archived Virtual Machine image files.
* The secure chat application was modified to support sending and receiving of text files encrypted using DES algorithm and decrypted using an easy to remember 8 character long password. As a result the chat user can exchange large amounts of confidential information.

# 6. Results and Discussions

## 6.1 Kernel Issues

The initial idea for the project was based on compiling the latest version of the Linux Kernel using GCC, append as many hardware drivers as possible (for portability), and install all necessary user applications using tar.gz distribution model. However after fair amount of research it turned out that only highly experienced expert in the field of UNIX and Linux could configure and compile the Linux Kernel in its tar archive file format. According to [6], [7], [8] it is recommended that the developer is to select an already existing Linux distribution and work around a custom solution. After further research it became clear that even the most popular Linux distributions (e.g. Mint, Ubuntu, CentOS) were not build from scratch but rather modified. Even though this was an unpleasant surprise and very problematic at the beginning of the project; there was an easy solution. Follow the best and brightest and create a Linux distribution, fine-tuning a slimmer and uncluttered already existing Linux distribution such as Debian(e.g. Ubuntu, BackTrack, MythTV), Slackware(Suse, OpenSuse), and Red Hat(e.g. Mandriva, CentOS, Fedora).

The process of compiling and installing the newer 3.3 Linux kernel took more than 30 minutes in total. In retrospect it would have been a better approach to use the alternative upgrade method. This method is patching, where the user would patch the current version only with the newest features of the kernel and nothing else. Also the size of the bz2 archive was much smaller. It is important to note that even though the configuration and installation of the kernel was successful, the system was corrupted – some of the services failed to initialise at boot time. According to the Arch Linux mail list the latest kernel is not entirely compatible with 3.2 and it is recommended to use the distribution’s packet manager to install the kernel.

During the installation procedure three more issues prevented the system from working properly. First was the choice of a file system. For Linux the three popular extended file systems are a perfect solution. Especially Ext4 which prevents serious damage to Solid State Drives and USB disk drives, unlike other file systems which cause physical wear on the devices if used for prolonged duration of time. Unfortunately close to none PC BIOS chipsets support Ext4. Almost all of them support FAT16, FAT32, NTFS, and ISO 9660 (i.e. Compact Disk File System). The second issue was to overcome this limitation using LILO, GRUB, or Syslinux. The LInuxLOader is the most primitive of the three and offers very few features. GRUB is slightly more advanced but during the testing procedure I was not able to boot the USB using neither. Using Syslinux was the only possible way to start-up the operating system. The third issue was drive identification. Linux can identify a drive using four methods – by ID, label, UUID, and path. The only reliable method to boot from a pre-defined disk drive is UUID (Universally Unique Identifier) which guarantees there would be no duplicates. However for this project the label method seemed more appropriate. That is because new users can rename their USB drive (e.g. LSBU Linux) and afterwards install the system. The real disadvantage of UUID is that it is 128 bit long and difficult to manage across different computers which may generate a different identification.

## 6.2 OpenSuse Studio Concerns

The popular SUSE distribution developed and supported by Red Hat / Novell named after the German computer innovator Konrad Suse offers a very interesting web application which lets users created their own Linux distribution. The user can select a GUI (i.e Gnome, KDE, ICEWM) and add 10,000 system and user applications easily. However there were three major problems with this approach for this project. First of all, the resulting distributions are heavily branded with the logo of OpenSuse during the boot, the desktop, windows manager etc. The second concern was that the resulting distributions ran very well on VirtualBox/VMWare and Live CD/DVD but behaved unpredictably on USB drives (observed during testing). The concern was the concept of a Live CD/DVD itself. After some testing and research it became obvious that such media is only used to demonstrate a half-functioning Linux distribution and give the user a simple GUI tool for OS setup. However the aim of this project was to produce a fully-functional Linux distribution for USB media.

## 6.3 Absence of GUI

One of the aims of this project was to provide the user with an intuitive and easy to use Microsoft Windows like graphical user interface. However after some testing and considerations on the value and benefit of such software, they were removed from the final version. Both KDE and Gnome require massive system resources (i.e CPU, RAM, HDD) to operate properly and give the user that familiar desktop environment. After KDE/Gnome was installed on the system, there was only 20% left free disk space on the USB thumb drive. It is important to note that these environments were installed just after the installation of the basic system software. With the addition of other software the USB thumb drive was completely full. Another reason why the GUI system was omitted is because of Linux’s appalling graphics/video card driver support. All three KDE, Gnome, and XFCE4 require already installed, configured, and operational Xorg. On the other hand Xorg requires driver for the target PC graphics card which may be Nvidia, Intel, ATI, Matrox etc. During the testing of the draft version with KDE, Xorg either could not load the driver or crashed after short amount of time. From personal experience with other Linux distributions namely Ubuntu, graphics always seemed a difficult task to accomplish for Linux. Whether that is the fault of the distribution, misconfiguration by the developer or inaccessibility of proprietary drivers by Nvidia and Intel, the decision was simply to neglect the GUI since TTY is where UNIX performs best.

## 6.4 The choice between XAMPP and LAMP

The final version of the project was reconfigured with XAMPP rather than traditional installation and configuration of Linux Apache MySQL PHP. The latter one has the benefit of being the better performing, most secure, and flexible in terms of configuration variables. However the XAMPP package offers more than 50 tools (phpMyAdmin, OpenLDAP), widgets(Webalizer), and special libraries (libpng, ImageMagic,GD). Problems with user privileges/permissions, user groups, virtual hosting, security configuration were encountered during the manual installation of AMP. On the contrary XAMPP was installed and configured with few minor issues. The main difference between the two and the reason to choose XAMPP is because it is targeted towards web development (e.g CS students) rather than commercial Internet service (i.e. LAMP).

## 6.5 Chat application problems

One of the first problems was synchronisation between sending and receiving messages. It was possible for both clients to be locked in a state where neither can use the System.in keyboard input. To fix this, two distinct threads were used that run in client/server mode – the client (only for sending messages) and the server (only for receiving messages).

Another problem was the implementation of the RSA algorithm. There is no direct correlation between java.math and the arithmetic used in the algorithm. As a result either Java notified me for execution exceptions or the output was not expected. In particular the decrypted text was mangled and did not represent the original encrypted text at all. Furthermore some implementations of finding the exponent **e** loop were very slow at took minutes to produce any result. Similarly other loops took seconds to finish but the output was incorrect and did not satisfy the algorithm’s conditions. Curiously BigInteger prime numbers could not be multiplied or divided by simple integers and the full extent of the java.math.BigInteger library methods had to be used to achieve the desired result.

|  |
| --- |
| phi = (p.subtract(BigInteger.valueOf(1)).multiply(q.subtract(BigInteger.valueOf(1)))); |

Another problem was sending and receiving files by the chat clients. It turned out it is not as straight forward as sending/receiving plain text/cipher messages. To overcome this issue read and write buffers were used to synchronise transport of text files across the network. In addition the commonly used classes such as InputStreamReader, PrintWriter, and BufferedStreamReader were not suitable for sending chucks of data via the network media.

|  |
| --- |
| do{  dataOutputStream.write(buffer, 0, length);  dataOutputStream.flush();  } while((length = fileInputStream.read(buffer)) != -1); |

On a related note choosing the encryption scheme was also problematic. For example DES does not operate by itself. It requires CBC or ECB with character padding such as PKCS#5. The same goes for AES and Blowfish algorithms. Nonetheless the first version of the DES encryption did not work as expected. After some research it turned out that DES with ECB works without initialising vectors but DES with CBC does not. Since the javac debugger did not notify me for any exceptions it was challenging finding out where the problem was.

|  |
| --- |
| try{  crypt = Cipher.getInstance("DES/CBC/PKCS5Padding");  crypt.init(Cipher.ENCRYPT\_MODE, secretKeySpec, iv);  } catch (java.security.NoSuchAlgorithmException noSuchAlgorithm){  System.out.println("The encryption algorithm failed.");  } |

Yet another problem was distinguishing between plain text, cipher text, and encrypted files send from the remote host. To overcome this problem simple console messaging was implemented. The user does not see the exchange of commands (e.g. secret, file, exit), only the meaningful results.

# 7. Conclusion

The portable Linux project was successfully configured and installed on both USB medium and as virtual machine appliance. The system software and kernel was successfully compiled and passed the functionality and user convenience tests. Unfortunately the graphical user interface and related graphical software applications had to be removed due to their heavy system requirements. However that may have been for the best because of the many difficulties experienced during the installation, configuration, and start-up of Xorg, graphics card drivers, and Gnome/KDE. In addition this makes this distribution suitable for older less powerful computers which can benefit from the latest non-graphical Linux/UNIX programs apt for server environment and student software development. One of the most helpful knowledge base resources used during the development of the project was the Arch Linux Wiki. There are many books, tutorials, and FAQs on Linux but they are all plagued by two core UNIX problems – documentation and compatibility as discussed by [11]. Thus having a centralised database of theory, practice, and source code examples such is the case of Arch Linux Wiki is of great benefit for the developer and undoubtedly for the inexperienced user.

# 8. Future Development

Despite this project’s original ideas, it is still much alike other Linux distributions. Perhaps a more advanced version of the developed operating system could be further improved by USB persistent mode. Also since there is no substantial and heavy GUI, Ncurses applications could be developed to partially substitute X / GTK+ / QT4 built applications. It is worth to note that there exists Curses web browser, audio/video player, advanced text editors, and other specialised software.

The live USB feature of this project is a popular spin off live CD or live DVD. Undoubtedly there is gain in terms of performance both boot-up time and hard drive seek time. However the user still cannot permanently change or store information on neither a CD-RW nor USB. Arguably this makes the core OS more secure and reliable under the control of an inexperienced Linux user. Nonetheless he or she would have to update the system on regular time intervals (e.g. annually) to get the latest software and benefit from application patches and updates. Thus a future development could focus on creating a persistent USB Linux distribution with some additional security feature which would prevent destruction and malfunction of the kernel.

# List of References

[1] Toms Hardware. USB Flash Drive: Super Talent Pico C Gold USB 8 GB, 2008. [Online] Available from: <http://www.tomshardware.com/reviews/usb-hard-drive,2015-4.html> [Accessed 13/03/2012].

[2] Distro Watch. Linux Distribution Ranking, 2012. [Online] Available from: <http://distrowatch.com/> [Accessed 15/03/2012].

[3] M. Lucas. Absolute FreeBSD 2nd edition, 2008. San Francisco: No Starch Press.

[4] J. Kemp. Linux System Administration Recipes: A Problem-Solution Approach, 2009. New York: Apress.

[5] The Open Group. The Single UNIX Specification, Version 4, 2010. [Online] Available from: <http://www.unix.org/version4/> [Accessed 18/03/2012].

[6] The Linux Kernel Archives. New to Linux, 2012. [Online] Available from: <http://www.kernel.org/> [Accessed 18/03/2012].

[7] Linux From Scratch. Stable Linux From Scratch Version 7.1, 2012. [Online] Available from: <http://www.linuxfromscratch.org/lfs/view/stable/> [Accessed 18/03/2012].

[8] K. Yaghmour et al. Building Embedded Linux Systems 2nd edition, 2008. Sebastopol: O’Reilly Media.

[9] S. Sarwar et al. Unix: the textbook 2nd edition, 2005. Boston: Pearson/Addison-Wesley.

[10] B. Goodheart. UNIX curses explained, 1991. New York: Prentice Hall.

[11] S. Garfinkel. The UNIX Hater’s Handbook, 1994. San Mateo: IDG Books.

[12] S. Yan. Cryptanalytic Attacks on RSA, 2008. New York: Springer.

[13] B. Kalinski. Some examples of the PKCS Standards, 1999. Bedford: RSA Laboratoties.

[14] C. Pfleeger et al. Security in Computing 3rd edition, 2003. New York: Prentice Hall.

# Appendix A

The following image is a screenshot of the secure chat application. Note the generated prime numbers, calculated modulus and Phi, generated public and private exponents. The application acts as client/server and requires the address and port number of the local/remote chat peer. Messages are distinguished with the specified nickname. The text file is encrypted/decrypted with the user specified 8 character long password. The same password should be entered by the other chat peer.

