

# SECURE DATA COMMUNICATION FRAMEWORK FOR MOBILE DEVICES

## A DISSERTATION

*Submitted in partial fulfilment of the  
requirements for the award of the degree*

*of*

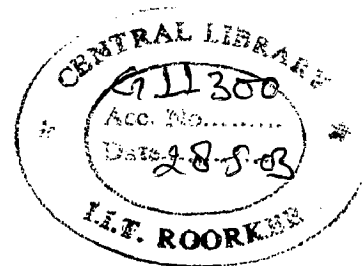
MASTER OF TECHNOLOGY

*in*

INFORMATION TECHNOLOGY

By

**NITIN GUPTA**



**ER & DCI  
NOIDA**

**IIT Roorkee-ER&DCI, Noida  
C-56/1, "Anusandhan Bhawan"  
Sector 62, Noida-201 307**

**FEBRUARY, 2003**

## CANDIDATE'S DECLARATION

---

I hereby declare that the work presented in this dissertation titled "**Secure Data Communication Framework for Mobile Devices**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT, Roorkee – ER&DCI Campus, Noida**, is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Mr. Ajit Saha**, Scientist-C, TDPP Division, National Informatics Center, New Delhi, and Co-guide **Mr. Munish Kumar**, Project Engineer, ER&DCI, Noida.

I have not submitted the matter embodied in this dissertation for the award of any other degree.

Date: 21/02/2003

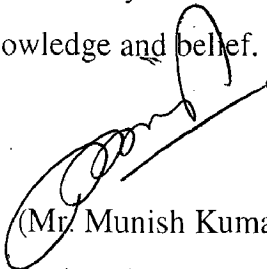
Place:

  
(Nitin Gupta)

## CERTIFICATE

---

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.


  
(Mr. Munish Kumar)

Project Co-Guide

Project Engineer

ER&DCI, Noida.



  
(Mr. Ajit Saha)

Project Guide

Scientist- C, TDPP Division,

N.I.C. New Delhi.

Date: 21-02-03

Place: Noida

Date: 21.02.2003

Place: New Delhi

## ACKNOWLEDGEMENT


---

First of all I would like to pay my humble regards to **Prof. Premvrath**, Director IIT-Roorkee, **Mr. R.K. Verma**, Executive Director ER&DCI-Noida, **Mrs. Rama Nangpal**, Technical Director, TDPP Division, N.I.C. New Delhi, for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Dr. A.K. Awasthi**, Programme Director, M.Tech (I.T) & Dean PG S&R, IIT-Roorkee, **Prof. R.P. Agarwal**, Course-Coordinator M.Tech (I.T.), IIT-Roorkee, **Mr. V.N. Shukla**, Course-Coordinator M.Tech. (I.T.), ER&DCI-Noida for their support and help generously extended to me without which this project could not have been completed in time.

I owe a special sense of gratitude to my honorable guide **Mr. Ajit Saha**, Scientist C, TDPP Division N.I.C., New Delhi and my respected co-guide **Mr. Munish Kumar**, Project Engineer, ER&DCI-Noida for their prodigious guidance, painstaking attitude and giving reformative suggestions throughout my project

My sincere thanks are due to **Dr. (Ms) P.R. Gupta**, Reader, ER&DCI, Noida for the encouragement and valuable suggestions she provided me during the course of my project work.

Above all I would like to thank my family. My parents provided me a perfect environment for my studies and supported me throughout. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

  
(Nitin Gupta)  
Enrollment no: 019030

# CONTENTS

---

<b>CANDIDATE'S DECLARATION</b>	(i)
<b>ACKNOWLEDGEMENT</b>	(ii)
<b>ABSTRACT</b>	1
<b>1. INTRODUCTION</b>	3
1.1 Objective of the dissertation	3
1.2 Scope of the work	4
1.3 Overview	4
1.4 Organization of the thesis	6
<b>2. LITERATURE SURVEY</b>	7
2.1 Java 2 Micro Edition	9
2.2 Kilo Virtual Machine (KVM)	10
2.3 Configurations and CLDC	11
2.3.1 Overview	11
2.3.2 CLDC Libraries	14
2.3.3 CLDC limitations	14
2.4 The Mobile Information Device Profile (MIDP)	15
2.4.1 Overview	15
2.4.2 The MIDP Libraries	16
2.5 MIDlets	17
2.6 Advantages of J2ME	18
2.7 J2ME Vs WAP	19
2.8 Cryptography	20
<b>3. ANALYSIS &amp; DESIGN</b>	
3.1 Analysis of Problem	23
3.1.1 Authentication Mechanism	23
3.1.2 Encryption and Decryption Mechanism	25
3.2 Design	27
3.2.1 Authentication of thin Client	27
3.2.2 Data Encryption	28
3.3 Algorithms Used	
3.3.1 Secure Hash Algorithm	30
3.3.2 RC4 Encryption	33
<b>4. IMPLEMENTATION ASPECTS</b>	
4.1 Main MIDlet Class	35
4.1.1 SecureMIDlet	35
4.1.2 HexadecimalCode	36
4.1.3 OutputString	37

4.2 Authentication of client using SHA1	37
4.3 Encryption and Decryption using RC4	38
4.4 Servlet	38
<b>5. RESULTS AND DISCUSSION</b>	<b>39</b>
<b>6. CONCLUSIONS AND FUTURE WORK</b>	
6.1 Conclusion	43
6.2 Future work	43
<b>REFERENCES</b>	<b>45</b>
<b>GLOSSARY</b>	<b>47</b>
<b>Appendix A – Tools Used</b>	

## ABSTRACT

---

The use of wireless devices such as cellular phones and two-way pagers has undergone tremendous growth over the past few years. As the wireless market matures, people will demand more advanced reliable applications. Sun Micro system and a group of wireless industry leaders such as Nokia, Motorola, and Palm defined a new set of standards called Java 2 Micro Edition (J2ME) to help in developing and deploying the next generation wireless applications, but the transaction security is becoming an important concern for mobile users and wireless application developers alike. The overall security of the network is not so strong and the weakest link is the client-side device. The interceptable nature of wireless signals and the limited memory and computing power of most handheld devices leaves wireless systems dangerously vulnerable to data theft. Thus I have proposed and implemented a platform independent Authentication and an Encryption & Decryption system that will help the thin client to prove its identity to the server and will secure the communication between the client and the server. I have used Java 2 Micro Edition Wireless Tool Kit device simulators. The application developed for wireless devices is named as SecureMIDlet. The class files of this MIDlet applications is packaged in to a JAR file and then this JAR file can be installed in the mobile device via serial cable connected to a PC or via a wireless network. This approach gives mobile user a secure and unified model. For example, a customer making an on line query should not be impacted by whether they are using a mobile phone or a PDA, as long as each device can securely express the proper identity. The application is developed as one of the modules of "Mobile-Commerce Security", a research project going on at TDPP Division of National Informatics Center, New Delhi.



## **1.2 Scope of the work**

The application developed as one of the modules of “Mobile-Commerce Security”, a research project going on at National Informatics Center, New Delhi, will be used for client authentication and data encryption. Unfortunately, encryption is not a standard part of either the Connected Device Configuration (CDC) or the Connected Limited Device Configuration (CLDC) i.e. they do not support the java.security package, so currently CLDC/MIDP does not provide any cryptography APIs. As a result, crucial security APIs such as encryption/decryption ciphers are missing from all these standard profiles. Thus we have to write our own code for data security. Thus the application developed fulfills all the above requirements and provides a mean for authentication and data encryption that can be used by the mobile users for the secret transmission of their confidential data.

## **1.3 Overview**

Although Java-based wireless Web services have a bright future in the world of pervasive mobile commerce, the current technology is not yet mature. Security is among the remaining issues yet to be resolved. The lack of data security on current mobile application platforms is one of the biggest hindrances to mobile commerce adoption, especially in the corporate world.

Wireless communications are easy targets for airwave interception, and wireless devices rarely have the computing power to support strong encryption of all communication data. Moreover, on the back end, Web services run outside corporate firewalls and interact with each other using open messaging protocols. Wireless Web services are likewise vulnerable targets for various cracking attacks. Well developed point-to-point security technologies such as SSL/TLS and HTTPS are not suitable for the multiple vendor, multiple intermediary Web services network topography the focus needs to be on securing the contents themselves rather than the connections over which they travel. Despite the new challenges, Web services themselves, however, can be used to enhance mobile commerce security. Emerging Web services security specifications enable you to use Web services as security utilities.[1]

The biggest benefit of using the Java platform for wireless device development is that we are able to produce portable code that can run on multiple platforms. But even with this



advantage, wireless devices offer a vast range of capabilities in terms of memory, processing power, battery life, display size, and network bandwidth. It would be impossible to port the complete functionalities of an application running on a sophisticated set-top box to a cell phone. Even for similar devices such as PDAs and advanced smart phones, establishing portability between the two often poses a strain to one device and underutilization of the other. Real portability can only be achieved among groups of similar devices. Recognizing that one size does not fit all, J2ME has been carefully designed to strike a balance between portability and usability.

J2ME is divided into several different configurations and profiles. Configurations contain Java language core libraries for a range of devices. Currently there are two configurations: Connected Device Configuration (CDC) is designed for relatively big and powerful devices such as high-end PDAs, set-top boxes, and network appliances; Connected Limited Device Configuration (CLDC) is designed for small, resource-constrained devices such as cell phones and low-end PDAs. CDC has far more advanced security, mathematical, and I/O functions than does CLDC.

On top of each configuration rest several profiles. Profiles define more advanced, device-specific API libraries, including GUI, networking, and persistent-storage APIs. Each profile has its own runtime environment and is suited for a range of similar devices. Java applications written for a specific profile can be ported across all the hardware/OS platforms supported by that profile. The Mobile Information Device Profile (MIDP) and the PDA Profile are two of the more significant profiles for the CLDC. The Foundation Profile and the Personal Profile are two important profiles for the CDC.

The Personal Profile is built on top of the Foundation Profile to run on high-end PDAs. The Personal Profile is equipped with a complete Java 2-compatible virtual machine implementation. Personal Profile applications can leverage all the Java 2, Standard Edition (J2SE) domain-based security managers, as well as the extensive set of cryptography and security libraries available for J2SE applications. Overall, the Personal Profile offers mature security solutions that are similar to those for J2SE applications.

Implementing secure MIDP applications is much harder, due to the CLDC configuration's limited mathematical functionalities and the scant processing power of many of the underlying devices. MIDP devices are, however, the most widely used wireless devices, so enabling secure applications on those devices is very important. [1][2]

## **1.4 Organization of the thesis**

Chapter 2 includes the literature survey of the thesis that gives an idea about what Java 2 Micro Edition is and what are the Configurations and Profiles related to the J2ME devices. More over it also compares J2ME with WAP and tells the advantages of using J2ME in the mobile devices.

Chapter 3 gives the Analysis and Designing part of the project where the problem is analyzed with different approaches and finally a design is proposed to implement security feature in the CLDC/MIDP devices. It also discusses the standard algorithms SHA1 and RC4 that are used in the project.

Chapter 4 is the implementation part that discusses the implementation of the algorithms discussed above.

Chapter 5 includes Results and their discussion.

Chapter 6 includes conclusions and the future work

## LITERATURE SURVEY

---

A recent report by Zelos Group provides projections for regional and global shipments of mobile handsets that support wireless Java. Zelos Group contends that Java will be the dominant terminal platform in the wireless sector. Support for the technology will be found in over 450 million handsets sold in 2007, corresponding to 74% of all wireless phones that ship that year. [3]

Java has already secured substantial adoption in the wireless sector and about 11% of handsets that ship in 2002 incorporate some iteration of J2ME. Based on relatively conservative assumptions regarding carrier and OEM commitments to the platform it is expected that about one third of handsets that ship in 2004 will incorporate support for Java, a fraction that will grow to almost three quarters of all handsets in 2007. The forecasting model is based on a variety of assumptions including:

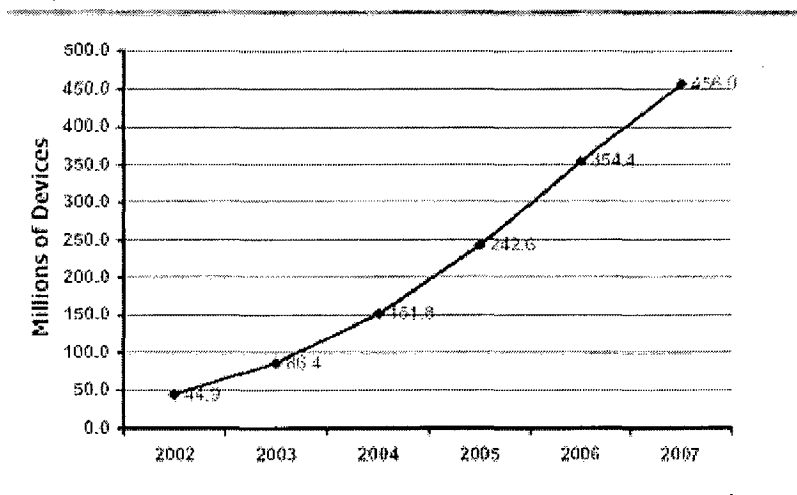
- Adoption will be most pronounced in the Japanese and Korean markets. Java will be supported in almost two thirds of handsets that ship in Japan in 2003 as all carriers are committed to the platform. Korea's largest carrier, SK Telecom, which had relied on technology from Sinjisoft called GVM, is migrating to support J2ME technology.
- Nokia is strategically committed to Java. About one quarter of Nokia handsets that ship in Europe in 2003 will support J2ME, a figure that will rise to about 80% in 2006. The proportion of Nokia handsets that support Java will be lower in markets like North America as introduction of new handset designs for CDMA and TDMA networks will lag.
- Second tier OEMs will adopt Java at a varied pace. For example, Motorola has committed to Java and the majority of iDEN handsets will support the platform. However, Motorola will initially lag Nokia in incorporating J2ME support on GSM handsets. Siemens is fully committed to the platform and will support Java in the

majority of its handsets in 2004. Samsung will hedge its bets and will support a variety of platforms over the next few years.

- Nextel and Sprint will drive initial adoption in the North American market. Shipments of Java handsets by these carriers, which collectively account for about 19% of cellular subscribers in the US, will account for about 50% of the North American market in 2003.
- Neither Microsoft nor Qualcomm will succeed in undermining the market momentum of wireless Java. Handsets from many vendors that support Windows CE or BREW will also support Java. Several VM solutions are already available for each platform and carriers will want to ensure support for Java on most high-end and mid-range hands.

### Wireless Java Market Projections

Figure 3 Global Shipments Java Handsets (2002 - 2007)



	2002	2003	2004	2005	2006	2007
Total Handsets (Mil.)	401.8	450.8	509.1	550.4	584.7	613.7
Java Handsets (Mil.)	44.9	86.4	151.8	242.6	354.4	456.0
Java Handset Percent	11%	19%	30%	44%	61%	74%

Source: Zolot Group

Figure: 2.1 Global Shipment Java Handsets (2002-2007) [3]

## 2.1 Java 2 Micro Edition

Over the last six years Java has grown into a complete and mature object-oriented development platform for applications in a vast and heterogeneous computing environment. These applications range from enterprise-level server applications for small devices. The current release of the Java 2 platform is defined in three editions, with each edition targeting a particular group of applications [4]. These three Java editions are:

- Java 2 Enterprise Edition (J2EE) – Designed for heavyweight and scalable business server applications.
- Java 2 Standard Edition (J2SE) – Designed for traditional and well-established desktop applications.
- Java 2 Micro Edition (J2ME) – Designed for the new generation of applications that target consumer electronics and embedded devices.

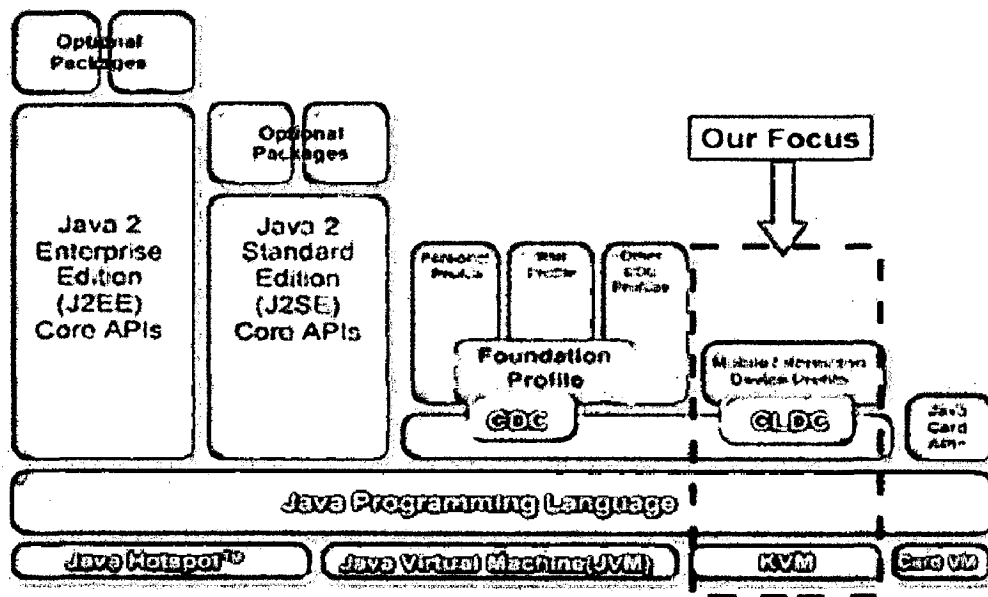


Figure: 2.2 JAVA™ Platform

As the wireless mobile devices have less computing power and smaller screens than their desktop counterparts, it stands to reason that J2ME represents a simplified version of J2SE with a reduced feature set. Thus J2ME is actually a subset of J2SE that supports a minimal set of features that are applicable to mobile devices, both wireless and wired.

The J2ME is a very small Java application environment. Built in to this core platform is the capability to receive not just application code, but libraries that form part of the Java 2 platform itself. In this way a J2ME environment can be dynamically configured to provide the environment the consumer needs to run an application, regardless of whether all the Java technology based libraries necessary to run the application are present on the device. The latest version of the KVM is packed with the J2ME, Connected, Limited Device Configuration (CLDC).

J2ME is composed of several building blocks -configurations and profiles-- that can be combined in different ways to match the memory and CPU requirements of different categories of mobile devices. The CLDC and MID profile combination is the application environment targeted at medium to low-end devices.

A portion of J2ME is fixed and applies to all devices, while another portion is defined specifically for a certain kind of device such as a mobile phone or a PDA.

## **2.2 Kilo Virtual Machine (KVM)**

The KVM provides the runtime environment for Java applications. It is a small Java virtual machine (JVM) specifically designed for small devices with lack of resources. It is made as small as possible but it still supports all main features of Java programming language. Letter “K” in KVM stands for “kilo” –amount of memory used by the machine is measured in kilobytes while in desktop system – in megabytes. KVM comes with a total memory budget less than 128 kilobytes (most compact version). This version is used in cellular phones, pagers and organizers. These 128 kilobytes contain virtual machine itself, some Java class libraries and some space for running Java applications. KVM can use both read-only and volatile memory [4][5]. If system classes are preloaded into the device and stored in read-only memory it requires less volatile memory.

- KVM is currently 40 KB of object code in its standard configuration.
- It requires only a few tens of KB of dynamic memory to run efficiently.
- It can run effectively on 16 bit processors clocked as low as 25 MHz to 32 bit processors.

In latest version of KVM, a new feature was added to improve performance and security of the KVM byte code. The CLDC of J2ME has a preverifier. The pre-verifier pre-checks your code for completeness and safety prior to loading on target J2ME device.

The features that are not supported by KVM are:

1. **No Java Native Interface (JNI)** – KVM doesn't support JNI for two reasons. First, according to the security model of the CLDC the application programmer can't download any new libraries containing native functionality, or access any native functions that are not part of the Java libraries. Second, implementing JNI is considered too memory intensive on CLDC devices.
2. **No User-Defined Class Loaders** – The KVM doesn't support user-defined class loaders due to security concerns. The built-in class loader in KVM cannot be overridden, replaced or reconfigured by the user.
3. **No Reflection, RMI, or Object Serialization** – There is no reflection feature in KVM, which means that CLDC programs cannot inspect the contents of classes, objects, methods, and so on. Consequently, all the features that depend on reflection are also not supported; they include Remote Method Invocation (RMI) and object serialization.
4. **No Thread Groups or Daemon Threads** – KVM fully supports multi-threaded applications, but does not support thread groups or daemon threads. We must use explicit collection objects to store the thread objects for performance.
5. **Weak reference** – Weak reference allows a program to be notified when the collector has determined that an object has become eligible for reclamation. The KVM does not support this feature.

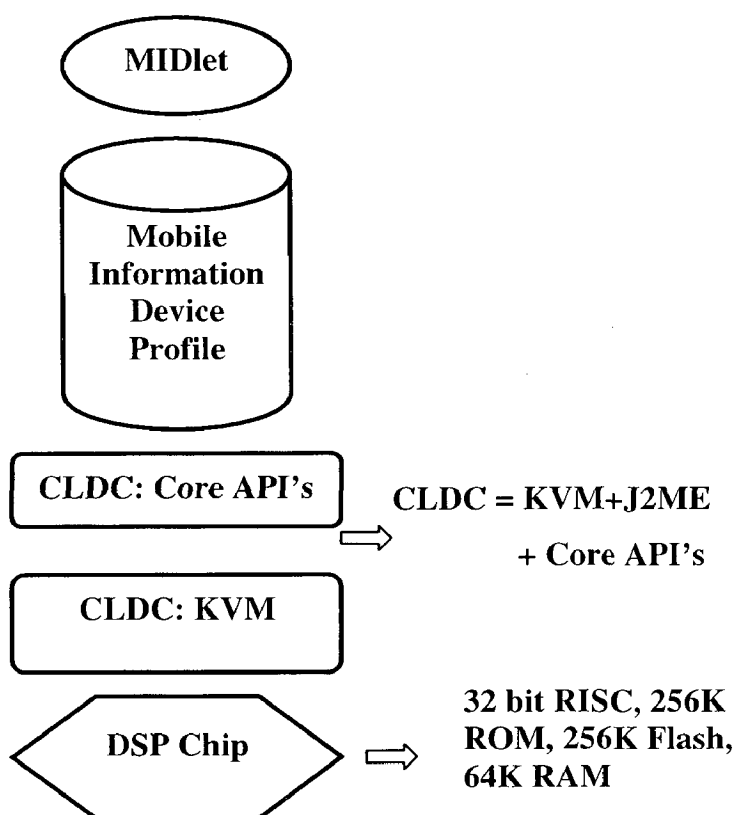
## 2.3 Configurations and the CLDC

### 2.3.1 Overview

Configuration is a minimum set of API's that is useful for developing applications to run on a range of devices. Configurations are very important because they describe the core functionality required of a range of devices. A standard configuration for wireless device is called as the Connected Limited Device Configuration, or CLDC. The CLDC takes in

to consideration the factors such as the amount of memory available to such devices along with their processing power. Thus CLDC can be thought as the set of essential classes and interfaces that are necessary for building applications for wireless mobile devices[4][5].

Note: The CLDC is the only configuration defined for J2ME. However, it is expected that additional configurations will be developed as Sun targets other types of devices for J2ME development.



**Figure: 2.3 J2ME™ Technology wireless Device Stack**

The CLDC clearly outlines the following pieces of information with respect to wireless mobile devices:

1. The subset of Java programming language features.
2. The subset of functionality of the Java virtual machine.
3. The core API's required for wireless mobile application development.

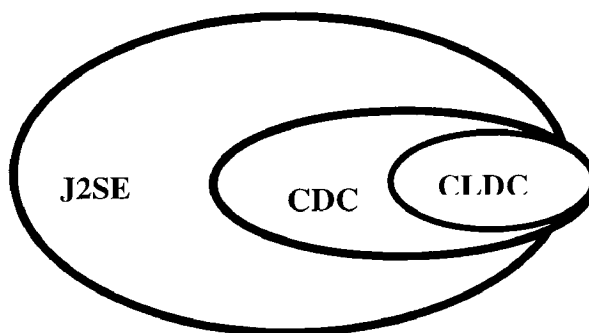
The hardware requirements of the wireless mobile devices targeted by the CLDC.



The final piece of the CLDC puzzle is the actual hardware requirements of a wireless mobile device. These requirements are specified as minimums, meaning that a device must have at least the following hardware attributes to qualify as a CLDC device:

- 160KB to 512KB of total memory budget available for the Java platform.
- A 16-bit or 32-bit processor with 25MHz speed.
- Low power consumption, often operating with battery power.
- Connectivity to some kind of network, often with a wireless, intermittent connection and with limited (often 9600bps or less) bandwidth.
- 128 Kilobytes of non-volatile memory available for the Java virtual machine and CLDC libraries.
- 32 Kilobytes of volatile memory available for the Java runtime and object memory.

The CLDC is not the only profile available for J2ME – for such devices as Internet TV’s, automobile navigation systems, screen phones and others there is the Connected Device Configuration (CDC). CDC is a superset of CLDC, this is made to make configuration compatible. Relations between CDC, CDLC and J2SE are described on the following picture



**Figure: 2.4 J2SE and J2ME classes.**

CDC and CLDC include some features that are not available in J2SE and are designed specifically for small devices. Such classes do not start with java. \*.

### 2.3.2 CLDC Libraries

J2EE and J2SE provide a very rich set of libraries for the development of applications for desktop and server machines. But these libraries require several megabytes of memory to run, and are therefore unsuitable for small devices with limited resources.

Java libraries for CLDC were designed to provide a minimum useful set of libraries for application development and to define profile for a group of small devices. The majority of the libraries included in CLDC are a subset of the larger Java editions (J2SE and J2EE) to ensure upward compatibility of applications. To provide small size of the package some libraries were redesigned networking and I/O.

The CLDC libraries can be divided into two categories:

1. Classes that are a subset of standard J2SE libraries,
2. Classes those are specific to CLDC.

☞ Classes, which are subsets of J2SE classes, are located in packages

- java.lang. \*,
- java.util. \*, and
- java.io. \*.

These classes have been derived from Java 2 Standard Edition version 1.3.

☞ Classes specific to CLDC are located in package

- javax.microedition.\*

These classes are CLDC specific and are not upward compatible with the J2SE libraries.

### 2.3.3 CLDC Limitations

1. **No Floating Point Support** – CLDC doesn't include floating-point support. As a result, the primitive data types float and double are not allowed in J2ME programs. The same thing is true of two type wrapper classes' java.lang.Float and java.lang.Double.
2. **No Finalization** – Finalization is a process in which the garbage collector gives an object an opportunity to clean itself up before being garbage collected. More specifically, the cleanup code is put in the finalize() method, which is called by the garbage collector. CLDC libraries do not include the method finalize() from the

object class. The reason that finalization is not supported in CLDC is to simplify the garbage collection of KVM.

3. **Limited Internationalization Support** – CLDC provides only very basic internationalization support with its `java.io.InputStreamReader` and `java.io.OutputStreamWriter` classes. These classes allow applications to convert a byte stream to a Unicode character stream and back. Other localization-related classes such as `Locale`, `ResourceBundle`, and `DateFormat` are not defined in J2ME.
4. **Error-Handling Limitations** – Only limited support is available for error handling in CLDC. There are only two error classes defined in CLDC: `java.lang.VirtualMachineError`, `Java.lang.OutOfMemoryError`. Most of the error classes are removed, for the two reasons:
  - In embedded systems, recovery from error conditions is usually highly device-specific. Application programmers should not be expected to handle these device-specific errors.
  - The Error exceptions are usually unrecoverable. Implementing full error-handling capabilities is rather expensive, and also imposes significant overhead on the resource-constrained CLDC devices.

## **2.4 The Mobile Information Device Profile (MIDP)**

### **2.4.1 Overview**

On top of a configuration sits a profile, which is more specific set of API's that further targets a particular type of device. The Mobile Information Device Profile (MIDP) is a set of Java APIs, which together with the Connected Limited Device Configuration (CLDC), provides a complete J2ME application runtime environment targeted at mobile information devices, such as cellular phones and two-way pagers. The MIDP defines the application architecture for these devices and addresses issues such as user interface, persistence storage and networking.

The MID Profile runtime environment allows to dynamically deploy new applications and services on the end user devices. It is designed to work on top of CLDC and the software and hardware requirements of Mobile Information Devices are in addition to

those for the broader range of Connected Limited Devices. The APIs defined by MIDP allows an open application development for Mobile Information Devices.

The primary goals of MIDP are to keep the implementation size minimal – must fit in small “footprint”, and efficiency – must run on low-end microprocessors with limited heap size and with minimal creation of garbage[4]

According to the MIDP Specification 1.0, a MID should have these minimum hardware characteristics:

#### **1. Display**

- Screen-size of 96x54 pixel
- Display depth of 1-bit
- Pixel shape (aspect ratio) of approximately 1:1

#### **2. Input**

- One-handed keyboard (ITU-T phone keypad), two-handed keyboard, (QWERTY keyboard), or touch screen.

#### **3. Memory**

- 128 kilobytes of non-volatile memory for the MIDP components.
- 8 kilobytes of non-volatile memory for application-created persistent data.
- 32 kilobytes of volatile memory for the Java™ runtime environment.

#### **4. Networking**

- Two-way
- Wireless
- Possibly intermittent
- Limited bandwidth

#### **2.4.2 The MIDP Libraries**

Where as the CLDC provides device-independent functionality, the MIDP libraries provide device specific functionality. This functionality includes

**1. Application Management Classes** - The classes associated with on device application management are defined in `javax.microedition.midlet` package. All MIDP applications must extend the `MIDlet` class in this package and implements its three abstract methods: `startApp()`, `PauseApp()`, `destroyApp()`.

**2. The GUI Classes** – The classes associated with the GUI and event handling are defined in the `javax.microedition.lcdui` package.

**3. Persistent Storage Classes** – The classes defined in the `javax.microedition.rms` package offers a persistent storage mechanism called `RecordStore` that allows applications to add, delete and update data records to the persistent storage on a device.

**4. Network Classes** – The Generic Connection framework defined in the CLDC contains a set of connection interface, but CLDC doesn't implement the actual protocols behind the connection interfaces. The implementation is left to the MIDP. Among all the connection interfaces, the `HttpConnection` interface is mandatory for all MIDP implementations. As a result, http communication is guaranteed to be available on all MIDP devices. The implemented classes of these interfaces can be found in the `javax.microedition.io` package.

## 2.5 MIDlets

MIDlets are programs that are used in a mobile computing environment using the J2ME API's. More specifically, the CLDC and MIDP API's must be used to develop MIDlets. A MIDlet cannot use classes or interfaces that do not appear in the CLDC and MIDP API's. Unlike an applet, a MIDlet isn't designed to run in a Web-browser. MIDlets don't have `main()` method and are not executed in Java interpreter. In this regard MIDlets are closer to applets because a special runtime environment is required for MIDlets to run. This environment primarily consists of an application manager that provides a means for selecting and launching MIDlet on a mobile device. Also similar to applets, the application manager for a MIDlet is responsible for establishing a frame window for a MIDlet.

MIDlet development is somewhat different from traditional Java applet and application development, we need to have Standard Java SDK and a suitable J2ME development

toolkit installed to build and test MIDlets. The development steps involved in taking a MIDlet from concept to reality are:

1. Develop the source code file.
2. Compile the source code files in to byte code classes.
3. Pre-verify the byte code classes.
4. Package the byte code classes in to a JAR file with any additional resources and a manifest file.
5. Develop an application descriptor file (JAD file) to accompany the JAR file.
6. Test and debug the MIDlet.

## 2.5 Advantages of J2ME

Java 2 Micro edition provides several benefits for wireless application development.

1. **Platform Independence:** J2Me extends the original “Write Once, Run Anywhere” design philosophy to the wireless world. Wireless applications developed using Java can be run on different devices from different vendors.
2. **Simple Programming Language:** Java technology can save development time and cost and thus considerably improve productivity. This factor is particularly critical in today’s fast-paced, highly competitive market.
3. **Rich Network Functionality:** Wireless applications by nature are network oriented and provide the user with constant communication to the outside world anywhere, any time. Java is designed with network capabilities in mind; it provides rich set of network libraries that makes writing network programs much easier.
4. **Dynamic Application Deployment:** Most of the existing applications on wireless devices are built-in, fixed-feature applications. It is very difficult to upgrade and install new applications without getting the manufacturer involved. J2ME provides a dynamic deployment mechanism that allows applications to be downloaded and installed onto devices over the wireless network. This mechanism is similar to executing a Java applet from a Web browser.

5. **Distributed Computing:** Java is especially popular when the Internet is main platform for an application. Java is the first choice for building Web applications. Applications developed using J2ME can be easily integrated with J2EE, which provides backend support to deliver enterprise wireless applications. Wireless applications by nature are thin clients. The strong XML support in Java makes client/server or transaction-based applications feasible on the wireless device.
6. **Graphical User interface:** Just like support in J2SE, J2ME comes with a rich set of user interface and event handling class libraries that make the most of the limited display space on wireless devices. This user interface support makes sophisticated video games and complex entertainment applications feasible on wireless devices.

## 2.6 J2ME Vs WAP

WAP currently uses the Wireless Transport Layer Security (WTLS) specification, which some believes has notable holes. WTLS was specifically designed to conduct secure transactions over a low bandwidth environment without requiring PC- level processing power or memory in the hand set. For this to work, the WAP gateway acts as a translator between WTLs encryption and the Web's standard, more robust SSL (Secured Socket Layer) security protocol. The problem occurs when data is handed over from WTLS to SSL, a process in which the data is decrypted and then re-encrypted meaning that for a split of second the data is not secure. Even though the data translation occurs within a secure data center, it's still valuable for that split second[4].

There is where J2ME, or more specifically java, really shines, thanks to the fact that all java applications, regardless of implementation, are restricted by a simple principle that untrusted code be placed in a sandbox, where it can play safely without doing any damage to the real world. When an applet or other piece of untrusted piece of code is running in a sandbox, there are a number of restrictions on what it can do. The most obvious is that it has no access whatsoever to the local file system or system resources.

J2ME has other benefits as well because it can support other Internet protocols. An example Sun gives is that you might download an Internet Message Access protocol-

based mail application and can communicate directly with your back-end mail server, not only speeding the rate of data exchange but also doing so without using WAP.

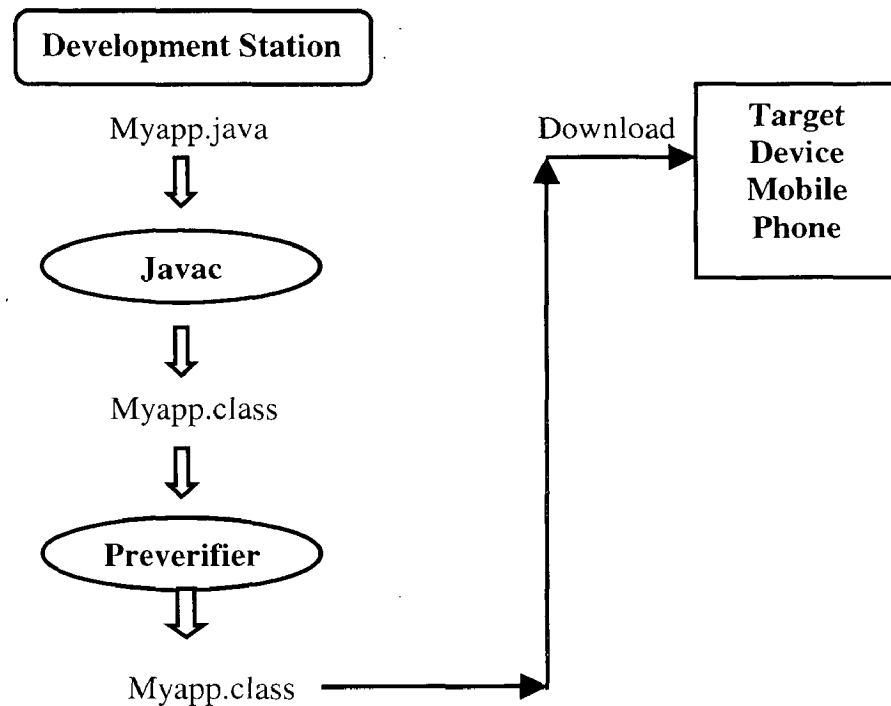


Figure: 2.5 Class File Verification in CLDC

## 2.8 Cryptography

Cryptography is a branch of mathematics that has powerful implications for data security. The basic principle of cryptography is that some mathematical problems are computationally expensive, which is a fancy way of saying they take a long time to solve. Cryptography relies on the use of keys. A key is a number that makes it easy to solve a math problem. Data integrity is just one aspect of securing communication. Cryptography can provide solutions to other aspects as well [6]. Generally, secure network communications must meet the following criteria:

- **Authentication:** Parties have to identify themselves. The digital signature on a public key certificate can validate the authenticity of the public key and therefore the party who holds it.



- **Data integrity:** The parties must make sure that the contents are not altered during transmission. Message digest is the most commonly used technology to guarantee data integrity.
- **Data confidentiality:** Sometimes, the communication data is sensitive and has to be kept secret. Digital signature does not provide data confidentiality. We have to use data encryption.
- **Non-repudiation:** After a message is sent, the sender or receiver should not be able deny it later.

The Various security threats are:

- **Eavesdropping:** Information remains intact, but its privacy is compromised. For example, someone could learn your credit card number, record a sensitive conversation, or intercept classified information.
- **Tampering:** Information in transit is changed or replaced and then sent on to the recipient. For example, someone could alter an order for goods or change a person's resume.
- **Impersonation:** Information passes to a person who poses as the intended recipient.

Impersonation can take two forms:

1. **Spoofing:** A person can pretend to be someone else. For example, a person can pretend to have the email address [nitin\\_roorkee@rediffmail.com](mailto:nitin_roorkee@rediffmail.com), or a computer can identify itself as a site called [www.iitr.ac.in](http://www.iitr.ac.in) when it is not. This type of impersonation is known as spoofing.
2. **Misrepresentation:** A person or organization can misrepresent itself. For example, suppose the site [www.mozilla.com](http://www.mozilla.com) pretends to be a furniture store when it is really just a site that takes credit-card payments but never sends any goods.



## ANALYSIS AND DESIGN

---

### 3.1 Analysis of the Problem

In mobile commerce the data travels over some infrastructure we know nothing about (provided by our mobile carrier) and probably over the Internet, as well. The Internet is not a secure network, for sure, and our carrier's mobile infrastructure probably isn't either. If we are passing sensitive data around, it's very possible that eavesdroppers at various points in the network can listen in on the data. They may even be able to change parts of it. If our MIDP application involves passing around credit card numbers or sensitive corporate data, we should be concerned.

#### 3.1.1 Authentication Mechanism

Devices that communicate over an insecure network like the Internet need to prove their identity to each other i.e. the client has to authenticate itself to the server. The trustworthiest methods of authentication are based on secrets, little pieces of information that are not widely known. A password is one example of a secret; a private cryptographic key is another. Presenting the password or proving that you possess a private key are two methods of authentication.

##### 1. Password Authentication System

The password authentication technique includes that the user supplies a user name and a secret value, a password. The server checks the user's password against a database; if the password is correct, the user is authenticated. The simple password scheme has two problems.

- First, people don't handle passwords well. They tend to pick bad passwords, and they tend to write passwords down in convenient but insecure places, like on a sticky note pasted to the front of the computer monitor. A bad password is one that is easy to guess. It may be vulnerable to a *dictionary attack*, guessing a user's password simply by using a dictionary of common passwords. This problem can be lessened by imposing restrictions on the password a user may select: it must be X characters long, it must contain both

letters and numbers, and so forth. It's nearly impossible to convince users that they shouldn't write down their passwords anywhere, although education about computer security may help. Requiring frequent changes of passwords may also help, although it makes a system less usable and actually encourages users to write passwords down.

- The second problem with simple password authentication is that the password itself, the client's secret value, is sent in cleartext between the client and server. There are two ways to fix this problem. You can either send the authentication information over an encrypted connection (probably using HTTPS) or send proof of the password instead of the password itself i.e. the client can send a message digest value of the authentication secret (and some other data) to prove to the server that it possesses the secret without actually sending the secret over the network connection. This approach is necessary wherever the communication between client and server is not encrypted.

## **2. Client Authentication Using Certificates**

If we are going to embed authentication information in the client application, we might as well go the distance and use X.509 certificates for authentication. One way is to create a root key pair and a root certificate representing the application itself, then create client key pairs and certificates signed by the root key. Each copy of the client software can be packaged with a unique client key and certificate. When the client connects to the server, it can use its private key to sign or encrypt some data, then send the signature and its certificate to the server, as shown below. The server uses the application root certificate to verify the client's identity and uses the client's certificate to verify the signature.

There are some difficulties with this approach too.

- The real difficulty of this approach is generating the pair of keys. It can take several minutes to generate a pair of keys on the mobile device due to the less processing power of the mobile device. One solution to this problem is to pre-generate the keys on Desktop P.C. and transfer the key to the mobile device, but this is not at all a good idea to leave the keys in a file stored in the device itself as mobile devices are easy to steal or lose. For example a companies financial data

financial data or private keys should not be recovered from a stolen mobile phone.

2. Distributing private keys securely is a challenge in itself. When the private keys are distributed as part of MIDlet suites, the challenge widens to include secure client provisioning[6].

### 3.1.2 Encryption and Decryption mechanism

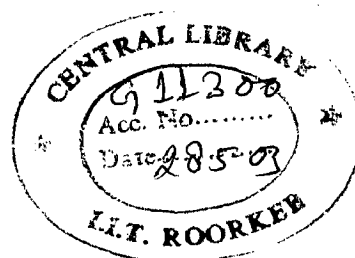
Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of transforming encrypted information so that it is intelligible again. A **cryptographic algorithm**, also called a **cipher**, is a mathematical function used for encryption or decryption. In most cases, two related functions are employed, one for encryption and the other for decryption.

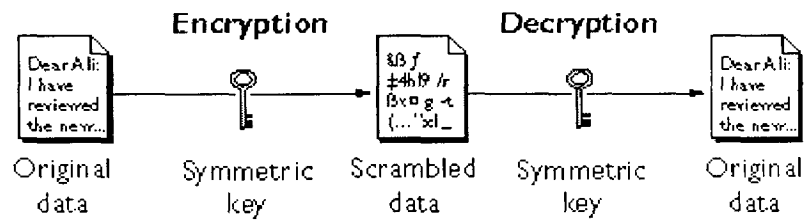
With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a **key** that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, and in some cases impossible for all practical purposes.

#### Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption[7]

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.





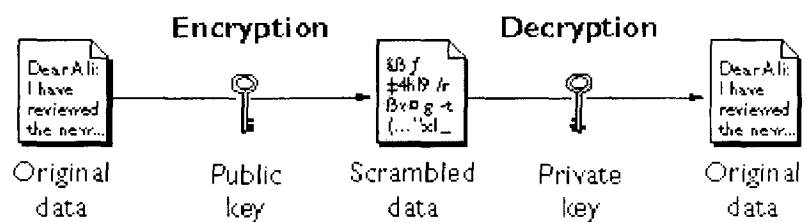
**Figure: 3.1 Symmetric key Encryption**

Symmetric-key encryption is effective only if the two parties involved keep the symmetric key secret. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but also can encrypt new messages and send them as if they came from one of the two parties who were originally using the key.

Symmetric-key encryption plays an important role in the SSL protocol, which is widely used for authentication; tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption, which is described later.

### Asymmetric-Key Encryption

Public-key encryption (also called **asymmetric encryption**) involves a pair of keys--a **public key** and a **private key**--associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret. Data encrypted with your public key can be decrypted only with your private key.



**Figure 3.2 Public-key Encryption**

The scheme shown in above figure where one freely distributes a public key, and only he will be able to read data encrypted using this key. In general, to send encrypted data to someone, we encrypt the data with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more computation and is therefore not always appropriate for large amounts of data. However, it's possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL protocol.

If the query request for data is generated by MIDlet program running on the thin client, this request is encrypted and sends over HTTP to the Server. This request is received on the server side, and decrypted to get the requested information. This required data is retrieved from the database. Data received from the database is encrypted send over HTTP to client. At the client side this response is decrypted and retrieved data is passed to MIDlet application.

## **3.2 Design**

After analyzing the merits and demerits of both the authentication mechanism I have decided to go for Password authentication system as it is faster than certificate authentication and there is no headache of distributing the private key. Message digests provides a way to solve the problem of user authentication. Instead of sending a password as plaintext, we create a message digest value from the password and send that instead. An attacker could just steal the digest value, but he cannot trace the password from it as digest is calculated using one-way hash function.

### **3.2.1 Authentication of Thin Client**

**(At Client Side)**

**Step1:** Enter the username and password from the user.

**Step2:** Calculate the time stamp.

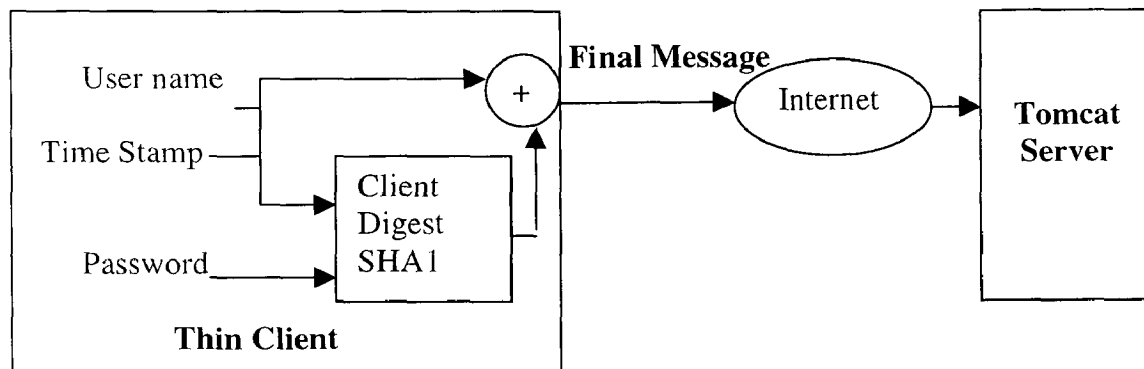
**Step3:** Send the username, Password and the timestamp to the Message digest

Block that uses SHA1.

**Step4:** Calculate the 160-bit digest using the Username, Password and the time stamp.

**Step5:** The digest calculated is converted to hexadecimal numbers.

**Step6:** Finally the username, timestamp and the digest so obtained is send to the server.



**Figure 3.3 Login Request from the server**

**Final Message=** Username + Time stamp + Client Digest.

**(At the server side)**

**Step1:** Receive the message from the client.

**Step2:** Separate the username; timestamp and the client digest from the received message.

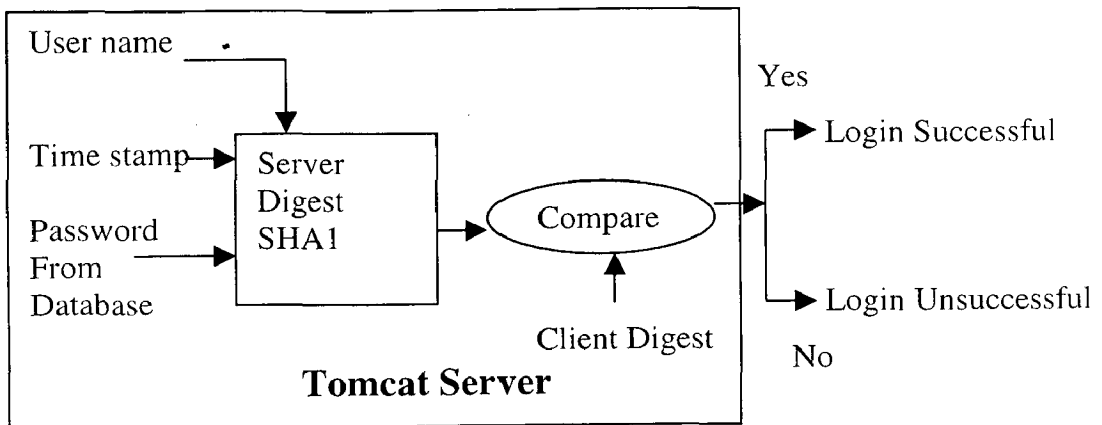
**Step3:** Lookup for the Database for the username received.

**Step4:** Calculate the Server digest using username, timestamp received and the Password just received from the database.

**Step5:** Compare the Client digest and the Server digest.

**Step6:** Login is Successful if the two digests matches and fails if the two digests do not match each other.





**Figure: 3.4 Response from the Server**

### 3.2.2 Data Encryption

#### (For Client Side)

**Step1:** Enter the data (Registration number in this case) in the text field from the user.

**Step2:** Encrypt the data using RC4 algorithm and a Symmetric key (Shared by both Client and Server).

**Step3:** Send the encrypted message to the Server.

#### (For Server side)

**Step1:** Receive the message from the server side.

**Step2:** Decrypt it using RC4 and the key already present in the server side.

**Step3:** Retrieve the requested record from the Database.

**Step4:** Encrypt the record using the encryption key and sent it back to the Client.

### 3.3 Algorithms Used

The standard algorithm SHA1 is used for generation of 160-bit digest value and RC4 is used for Encryption and Decryption.

#### 3.3.1 Secure Hash Algorithm

The algorithm takes as input a message with a maximum length of less than  $2^{64}$  bits and produces as output a 160 – bit message digest. The input is processed in 512-bit blocks. The overall processing of a message follows the structure shown for MD5 in Figure 3.5 With a block length of 512 and a hash length and chaining variable length of 160 bits. The processing consists o the following steps:[8]

- **Step 1: Append padding bits.** The message is padded so that its length is congruent to 448 modulo 512 ( $\text{length}=448\text{mod } 512$ ). Padding is always added even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512, the padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2:Append length.** A block of 64 bits is appended to the message. This block is treated as an unsigned 64 –bit integer (most significant byte first) and contains the length of the original message (before the padding).
- **Step 3: Initialize MD buffer.** A 160 –bit buffer is used t o hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A, B, C, D, E). These registers are initialized to the following 32-bit integers (hexadecimal values):

A= 67452301

B=EFCDA89

C=98BADCFE

D=10325476

E=C3D2E1F0

Note that first four values are the same as those used in MD5. However in the case of SHA-1, these values are stored in big – endian format, which is the more significant byte of a word in the low- address byte position. As 32-bit string s, the initialization values (in hexadecimal) appear as follows:

**Word A:** 67 45 23 01

**Word B:** EF CD AB 89

**Word C:** 98 BA DC FE

**Word D:** 10 32 54 76

**Word E:** C3 D2 E1 F0

- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each. The logic is illustrated in Figure ?? The four rounds have a similar structure, but each uses a different primitive logical function, which we refer to as  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f_4$ . Each round takes as input the current 512-bit block being processed ( $Y_q$ ) and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of an adaptive constant  $K_i$ , where  $0 \leq i \leq 79$  indicates one of the 80 steps across our rounds. In fact, only four constants are used. The values, in hexadecimal and decimal, are as follows:

The output of the fourth round (eightieth step) is added to the input to the first round ( $CV_q$ ) to produce ( $CV_{q+1}$ ). The addition is done to the independently on each of five words in the buffer with each of the corresponding words in ( $CV_q$ ), using addition modulo  $2^{32}$ .

- **Step 5: Output.** After all 512-bit blocks have been processed, the output from the  $L$ th stage is the 160-bit message digest.

We can summarize the behavior of SHA1 as follows.

$$CV_0 = IV$$

$$CV_{q+1} = \text{SUM}_{32}(CV_q, ABCDE_q)$$

$$MD = CV_L$$

**Where**

IV = initial value of the ABCDE buffer, defined in step 3

$ABCDE_q$  = the output of the last round of processing of the  $q$ th message block.

$L$  = the number of the blocks in the message (including padding and length fields).

$\text{SUM}_{32}$  = addition modulo  $2^{32}$  performed separately on each word of the pair of inputs.

MD = final message digest value

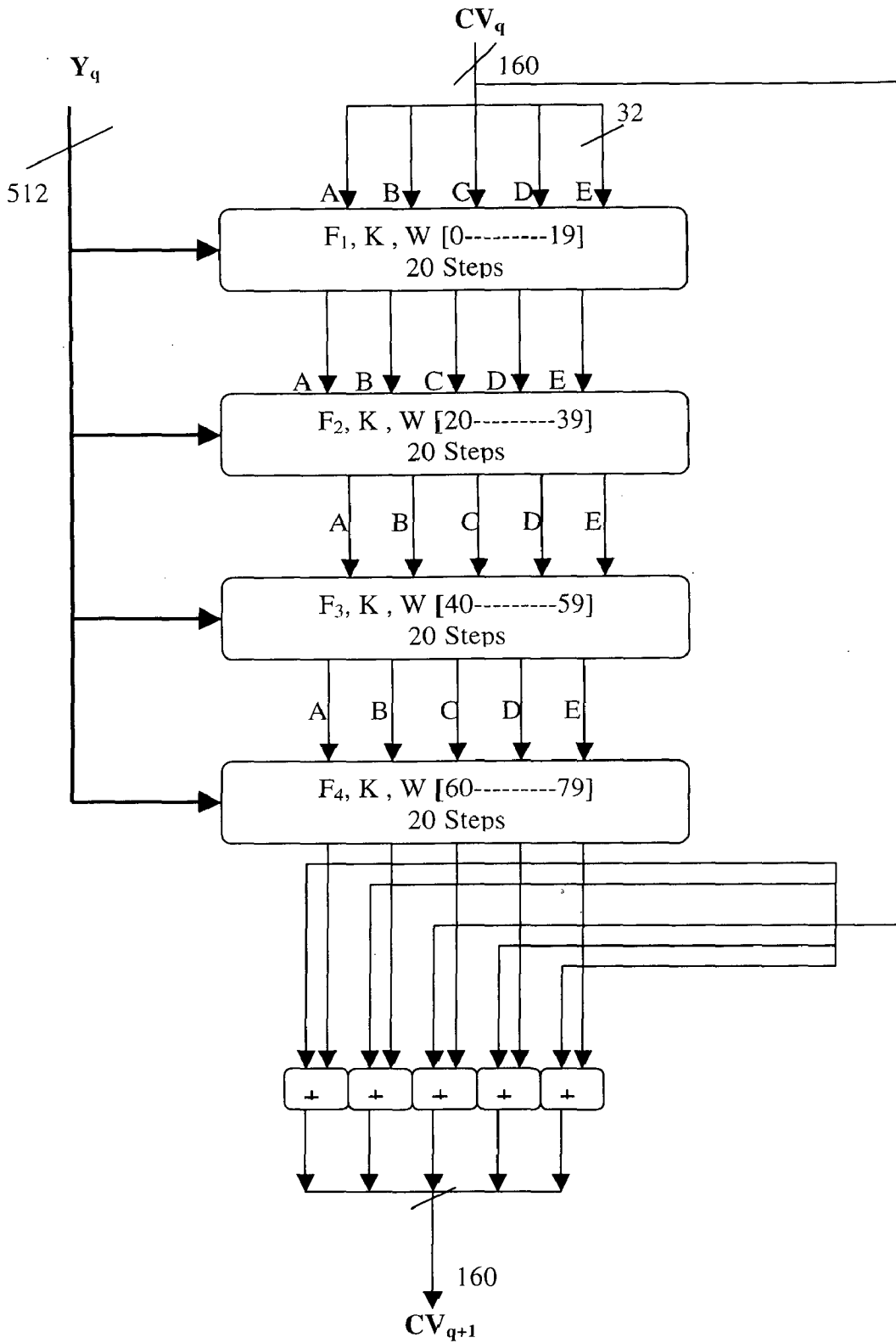


Figure 3.5 SHA1 Algorithm

### 3.3.2 RC4 Encryption

RC4 (Ron's code # 4 or Rivest) is a stream cipher symmetric key algorithm. It was developed in 1987 by Ronald Rivest and kept as a trade secret by RSA Data Security. On September 9, 1994, the RC4 algorithm was anonymously posted on the Internet on the Cyperpunks' "anonymous remailers" list. RC4 uses a variable length key from 1 to 256 bytes to initialize a 256-byte state table. The state table is used for subsequent generation of pseudo-random bytes and then to generate a pseudo-random stream, which is XORed with the plaintext to give the ciphertext. Each element in the state table is swapped at least once.

The RC4 key is often limited to 40 bits, because of export restrictions but it is sometimes used as a 128-bit key. It has the capability of using keys between 1 and 2048 bits. RC4 is used in many commercial software packages such as Lotus Notes and Oracle Secure SQL. It is also part of the Cellular Specification.

In the algorithm the key stream is completely independent of the plaintext used. An 8 \* 8 S-Box ( $S_0$   $S_{255}$ ), each of the entries is a permutation of the numbers 0 to 255, and the permutation is a function of the variable length key. There are two counters  $i$ , and  $j$ , both initialized to 0 used in the algorithm.

The S-Box is easily generated, using the following:

Fill  $S_1$  to  $S_{255}$  linearly (i.e.  $S_0 = 0$ ;  $S_1 = 1$  ...  $S_{255} = 255$ )

Another 256-byte array is then filled with the key; the key is repeated as necessary to fill the entire array.

The index  $j$  is then set to 0

for ( $i = 0$  to  $i = 255$ )

$$j = (j + S_i + k_i) \text{ MOD } 256$$

Swap  $S_i$  and  $S_j$

The following is used to generate a random byte:

$i = (i + 1) \text{ MOD } 256$

$j = (j + S_i) \text{ MOD } 256$

Swap  $S_i$  and  $S_j$

$t = (S_i + S_j) \text{ MOD } 256$

$K = S_t$

K is XORed with the plaintext to produce the cipher text, or the cipher text to produce the plaintext.

### Terminology

- **RC4:** Ron's code # 4 or Rivest
- **Cipher:** It is a cryptographic algorithm used for encryption and decryption.
- **Symmetric key algorithm:** It is an algorithm that uses the same key to encrypt and decrypt.
- **Stream cipher:** An algorithm that encrypts data one byte at a time.
- **Anonymous remailer:** A distribution system that strips off all of the sender information and re-mails the message under an anonymous name.
- **Cyberpunk:** Computer users that believe that privacy from government and large business institutions must be protected. These users generally have expertise in cryptography.
- **State table:** It is a table initialized from 1 to 256 bytes. The bytes in the table are used for subsequent generation of Pseudo-Random bytes. The Pseudo-Random stream generated is XORed with the plaintext to give the ciphertext.

The main strengths of RC4 are:

1. The difficulty of knowing where any value is in the table.
2. The difficulty of knowing which location in the table is used to select each value in the sequence.
3. A particular RC4 key can be used only once.
4. Encryption is about 10 times faster than DES.

## IMPLEMENTATION ASPECTS

---

In this project I have implemented two modules

1. Authentication of the user using Standard Hash Algorithm (SHA1).
2. Encryption and Decryption using RC4 algorithm.

The MIDlet and Servlet exchange various byte arrays, such as the timestamp, the random number, and the message digest value. To make this work smoothly in the context of HTTP headers, which are plain text, the byte arrays are exchanged as hexadecimal strings. A helper class, HexadecimalCod, handles the translation between hexadecimal strings and byte arrays. The MIDlet and the Servlet use this same class. At the MIDlet its main screen is a form where the user can enter a user name and a password. When the user invokes the **Login** command, the MIDlet calculates a 160-bit digest by passing the User name, Password and the calculated time stamp to the SHA1. It assembles various parameters into an HTTP request. It then reads the response from the server and displays on the screen. If the digest regenerated at the server side does not match the digest send by the client the authentication failed and the user is again asked to **Relogin**.

### 4.1 Main MIDlet Class

#### 4.1.1 SecureMIDlet

<b>Class</b>	SecureMIDlet.java
<b>Constructor</b>	SecureMIDlet()
<b>Description</b>	This is the main class that contains the calls for both the modules of the project. When a MIDlet is loaded into the device and the constructor is called, it is in the loaded state. This can happen at any time before the program manager starts the application by calling the <b>startApp()</b> method. After <b>startApp()</b> is called, the MIDlet is in the active state until the program manager calls <b>pauseApp()</b> or <b>desroyApp()</b> ; <b>pauseApp()</b> pauses the MIDlet, and <b>desroyApp()</b> terminates the MIDlet.

## Methods

### 1. login()

This method reads the Username and Password entered by the user and convert the username in small case to make the user name case insensitive and them calculates a time stamp and passes these three information to the SHA1 module to calculate the digest value. The username, timestamp and the digest (in hexadecimal code) are then appended together and send to the invokeServlet method.

### 2. Encrypt()

This method takes the entered enrollment number as the input and encrypts it using the RC4 encryption. Then the username and the encrypted message is appended together and send to the server. It also receives the response from the server and decrypts the response and displays them on the screen of the mobile phone.

### 3. invokeServlet(String url)

This method takes the URL to be fired as input and send the appended information to the server and collect the response from the server to display it on the mobile screen.

## 4.1.2 HexadecimalCode

This class uses methods to convert a byte sequence in to hexadecimal code and vice versa

### Methods

#### 1. bytesToHex(byte [] raw)

This method converts the input byte stream in to hexadecimal character stream.



## 2. **hexToBytes(char [] hex)**

This method converts the input hexadecimal character stream in to byte stream.

### 4.1.3 **OutputString**

This class is used to construct the appended string that is send to the server.

#### **Methods**

#### 1. **addParameter(String name, String value)**

This method appends a '&' before 'name' and a '=' sign after that and then it appends the value.

## 4.2 **Authentication Of the client using SHA1**

This is the authentication module of the project that uses the classes SHA1 class. The same class is used at the client side and the server side.

**Class** SHA1.java

**Constructor** SHA1()

**Description** This class is used to calculate the 160-bit Digest value of the information passed to it.

#### **Methods**

#### 1. **update (byte[] in, int inOff, int len)**

This method updates the message digest with a block of bytes.

#### **Parameters:**

in - the byte array containing the data.

inOff - the offset into the byte array where the data starts.

len - the length of the data.

#### 2. **doFinal( byte[] out, int outOff)**

This method closes the digest, producing the final digest value.

The doFinal call leaves the digest reset.

**Parameters:**

out - the array the digest is to be copied into.

outOff - the offset into the out array the digest is to start at

### [4.3] Encryption and Decryption using RC4

RC4 encryption algorithm is implemented for encrypting and decrypting of data. The same class is used both at client side and the server side.

**Class** RC4.java

**Constructor** RC4()

**Description** This class takes the enrollment number and encrypts and decrypts it using the RC4 encryption algorithm.

**Methods**

1. **processBytes**(byte[] in, int inOff, int len, byte[] out, int outOff)

This method processes a block of bytes from in putting the result into out.

**Parameters:**

in - the input byte array.

inOff - the offset into the in array where the data to be processed starts.

len - the number of bytes to be processed.

out - the output buffer the processed bytes go into.

outOff - the offset into the output byte array the processed data stars at.

2. **reset**()

This method resets the cipher.

### [4.4] Servlet

In this I used java servlets to complete the HTTP request. Actually we can use these servlets on any machine.

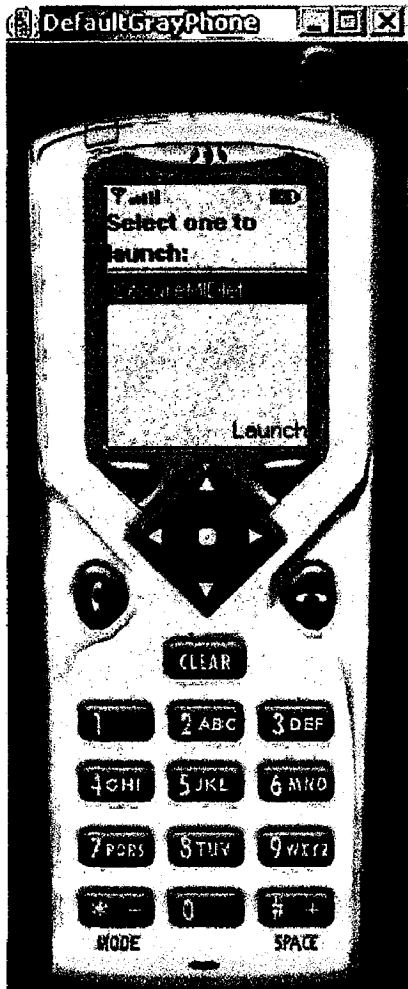
**Class** RequestServlet.java

This class also makes use of SHA1 class, RC4 class, and HexadecimalCode.

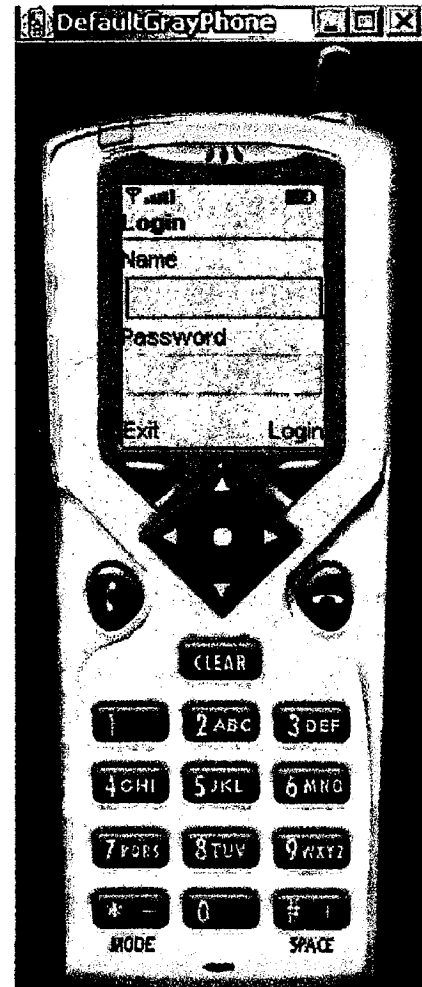
It collects the request from the client side and does the necessary processing and sends the response back to the client.

**RESULTS AND DISCUSSION**

When SecureMIDlet is run following screens occur which are shown below.



**Figure: 5.1** Screen When SecureMIDlet is run.



**Figure: 5.2** Screen to enter Username and Password.

**Data Entered by the user is Username and Password.**

**Username=Nitin**

**Password Entered=Nitin123**

**Time Stamp=00000f356c15499**

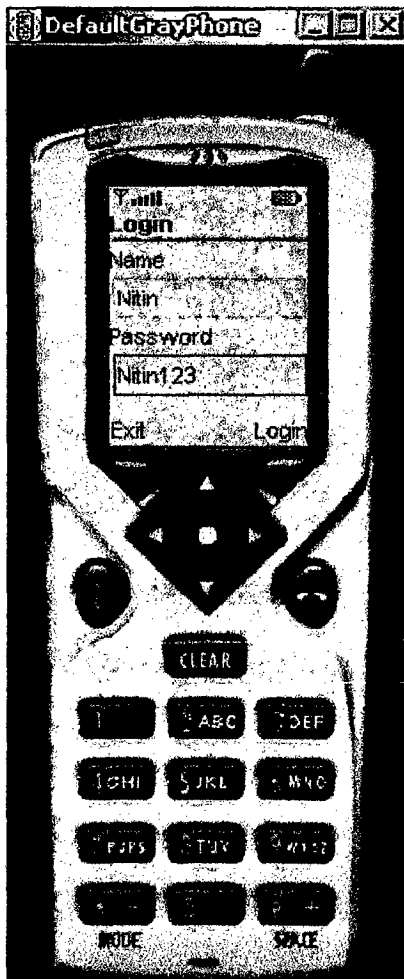
**Client digest (in Hex)=46ea1dbd3e21db32cd68f9e9803c6c18e3d74c36**

**Output stream=**

**?Username=nitin&timestamp=00000f356c15499&digest=46ea1dbd3e21db32cd68f9e9803c6c18e3d74c36**

**Server digest (in Hex)= 46ea1dbd3e21db32cd68f9e9803c6c18e3d74c36**

**Login is Successful!!**



**Figure: 5.3 Screen after entering Username and Password.**



**Figure: 5.4 Screen after pressing Login key.**

**Data Entered by user is the Enrollment number.**

**Input Enrollment Number = 019030**

**Client Encrypted Message=0da8c3453155**

**Output Transmitted=**

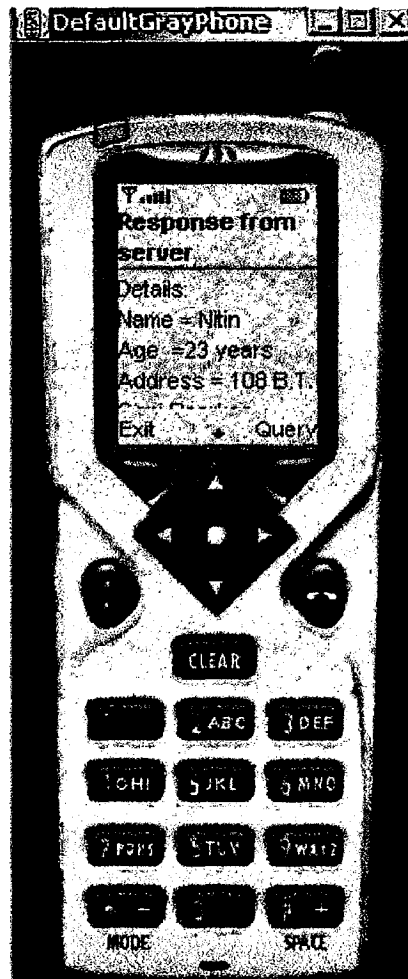
**%?User\_name=Nitin&message=0da8c3453155**

**Response From Server=**

**&eea8164e854d93a463faab7818be3aa8e5766dd2c0184a8340cfc48b1886470c61a7c3f33  
6c1d6d71b83aa10425aa63e878aaea417e24cd047b86d2a937a6f3e23e0ec3d77c986daa39  
81fa566b67d43da551db1684409**



**Figure 5.5 Screen for Entering Enrollment number.**



**Figure 5.6 Screen after pressing Send key.**

## Error Screens

There are two error screens in the project.

- If the enrollment number entered is less than or more than 6 digits.
- If the user enters wrong Username or password.

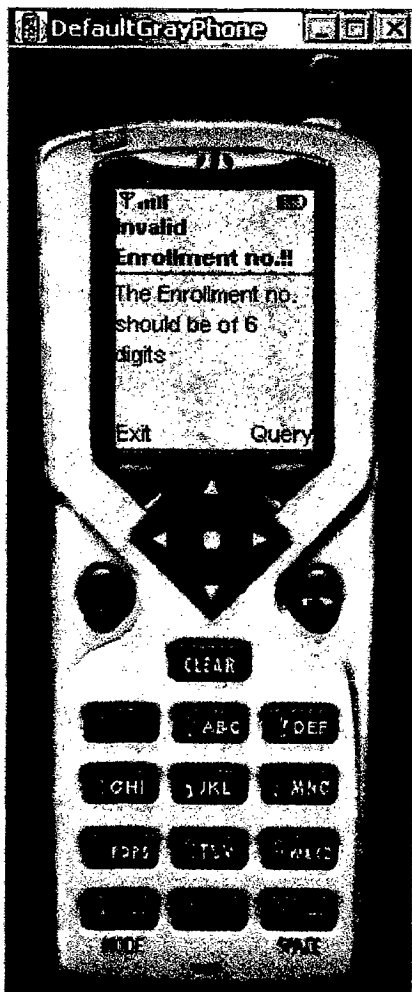


Figure: 5.7 Error Screen for incorrect Enrollment no.

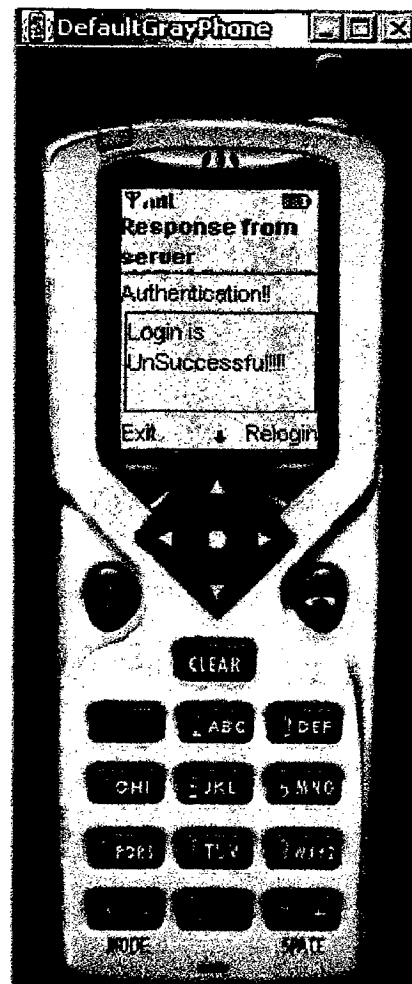


Figure: 5.8 Error Screen for incorrect Username or Password.

## CONCLUSIONS AND FUTURE WORK

---

### 6.1 Conclusions

With the support of J2ME by most of the wireless vendors such as Nokia, Erricson, Motorola e.t.c, transaction security will be of great advantage and thus I have implemented an authentication cum Encryption and Decryption system that can be used by a mobile user for proving its identity to a server as well as sending confidential messages secretly. The mobile users for secure transactions can use this application developed.

Here it is assumed that the thin client users have previously subscribed to a server and thus requesting information or action from the service.

### 6.2 Future Work

As a future enhancement of this project, the communication between the mobile devices can be made through XML based protocol, SOAP (Simple Object Access Protocol). This architecture will be very less resource consumable and data portability will be very high on different platforms. This architecture can avoid the need of WAP Gateway. The J2ME client can directly communicate to any type of server provided the server supports SOAP. As per the implementation, the request for data generated by MIDLET program running on the thin client can be encrypted, marshaled into SOAP envelope and send over HTTP to the Server. This request envelope will be received on the server side, de-marshaled and decrypted to get the request object. This request object will be parsed and required data can be called from the database using JDBC. Data received from the database can be encrypted and marshaled into SOAP response envelope and send over HTTP to client. At the client side this response envelop will be de-marshaled, decrypted, parsed and retrieved data can be passed to MIDLET application.





## REFERENCES

---

1. Michael Juntao Yuan, "Security challenges and solutions for mobile commerce applications", June 2002.  
URL: <http://www-106.ibm.com/developerworks/library/wi-secj2me.html>
2. Ju Long, Michael Juntao Yuan, "Securing your J2ME/MIDP applications", June 2002"  
URL: <http://www-106.ibm.com/developerworks/library/j-midpds.html>
3. Seamus McAteer, "Java will be the Dominant Handset platform", September 2002.  
URL: <http://www.microjava.com/articles/perspective/zelos>
4. Yu Feng and Dr. Jun Zhu, "Wireless Java Programming with J2ME", Techmedia Publication, 2001.
5. Michael Morrison, "Wireless Java with J2ME", Techmedia Publication, 2001.
6. Jonathan Knudsen, "Wireless Java Application Security 1 Design Concerns and Cryptography", September 2002.  
URL: <http://wireless.java.sun.com/midp/articles/security1/>
7. Bruce Schneier, "Applied Cryptography", John Wiley and Sons, inc, 2001.
8. William Stallings, "Cryptography and Network Security", Prentice Hall, 2000.
9. Ronald Rivest, "RC4 Encryption Algorithm".  
URL: [http://www.ncat.edu/~grogans/algorithm\\_history\\_and\\_descriptio.htm](http://www.ncat.edu/~grogans/algorithm_history_and_descriptio.htm)
10. Ju Long, Michael Juntao Yuan, "Securing your J2ME/MIDP applications", June 2002"  
URL: <http://www-106.ibm.com/developerworks/library/j-midpds.html>
11. Michael Juntao Yuan and Ju Long, "Java readies itself for wireless web services", June 21, 2002.  
URL: <http://www.javaworld.com/javaworld/jw-06-2002/>



## **GLOSSARY**

---

<b>CDC</b>	<b>Connected Device Configuration</b>
<b>CLDC</b>	<b>Connected Limited Device Configuration</b>
<b>HTTP</b>	<b>HyperText Transfer Protocol</b>
<b>J2EE</b>	<b>Java 2 Enterprise Edition</b>
<b>J2ME</b>	<b>Java 2 Micro Edition</b>
<b>J2MEWTK</b>	<b>Java 2 Micro Edition Wireless Tool Kit</b>
<b>J2SE</b>	<b>Java 2 Standard Edition</b>
<b>KVM</b>	<b>Kilo Virtual Machine</b>
<b>MIDP</b>	<b>Mobile Information Device Profile</b>
<b>PDA</b>	<b>Personal Digital Assistant</b>
<b>SHA</b>	<b>Secure Hash Algorithm</b>
<b>SOAP</b>	<b>Simple Object Access Protocol</b>
<b>SSL</b>	<b>Secure Socket Layer</b>
<b>TLS</b>	<b>Transport Layer Security</b>

# **APPENDIX - A**

## APPENDIX - A

### Tools Used

#### [A.1] Java 2 Micro Edition Wireless tool Kit (J2MEWTK)

The J2ME Wireless Toolkit supports the development of Java applications that run on devices compliant with the Mobile Information Device Profile (MIDP), such as cellular phones, two-way pagers, and palmtops.

The J2ME Wireless Toolkit supports a number of ways to develop MIDP applications. You can carry out the development process by running the tools from the command line or by using development environments that automate a large part of this process.

The KToolBar, included with the J2ME Wireless Toolkit, is a minimal development environment with a GUI for compiling, packaging, and executing MIDP applications. The only other tools you need are a third-party editor for your Java source files and a debugger.

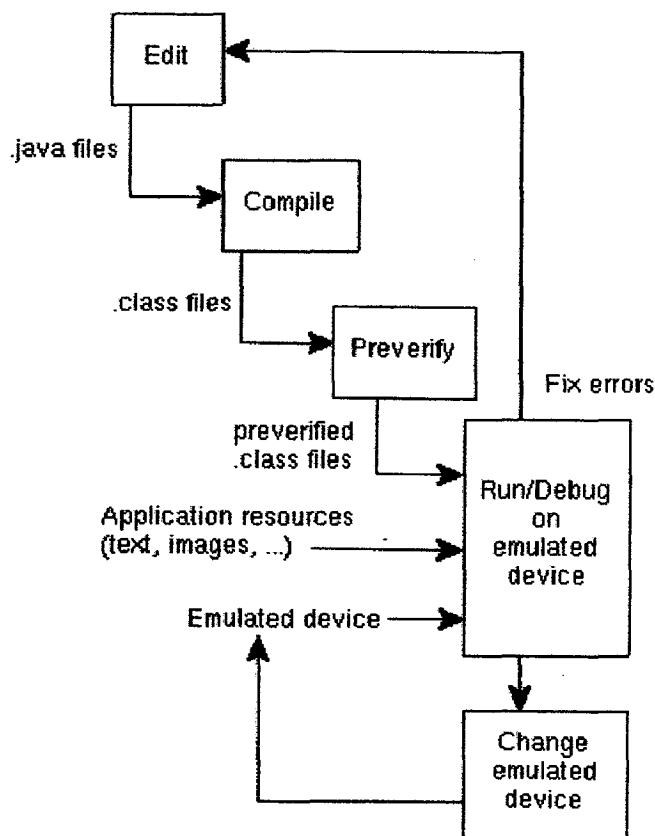
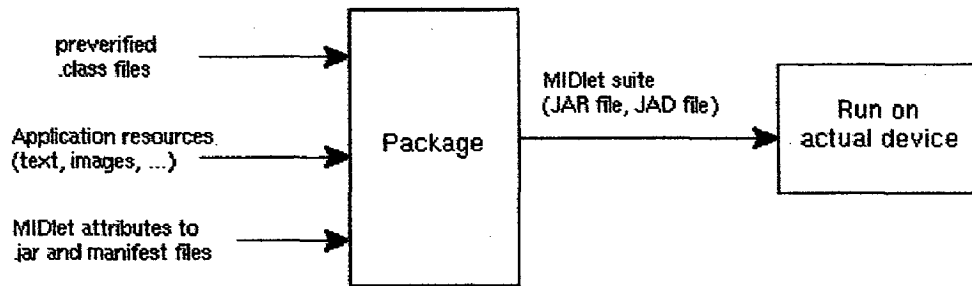


Figure: A.1 Development of a MIDlet



**Figure: A.2 Transfer of MIDlet to a Mobile Device**

### **1. Compilation and Prefabrication**

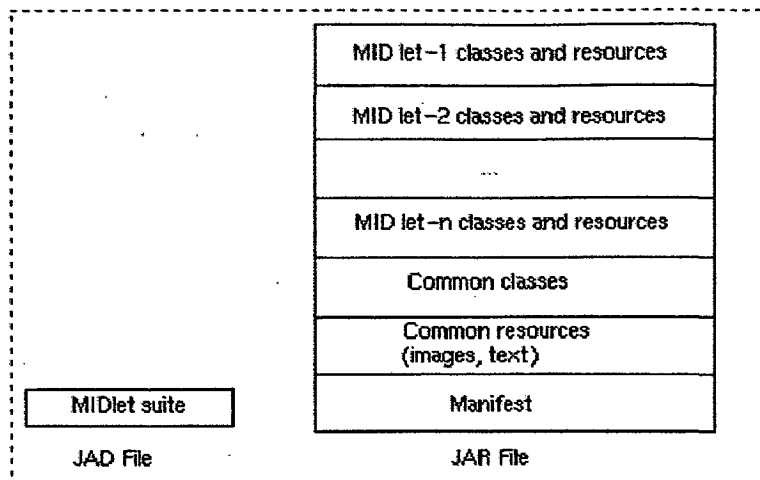
When you use Ktoolbar, the environment compiles your source files for you, using the Java 2 SDK, Standard Edition (J2SE™ SDK) compiler. After compiling the sources, the development environment passes the generated class files to the Preverifier. This tool rearranges bytecodes in the classes to simplify the final stage of bytecode verification on the CLDC virtual machine. It also checks for the use of virtual machine features that are not supported by the CLDC.

### **2. Running and Debugging**

When you use Ktoolbar, you can run and debug applications within the environment using the Emulator, which simulates the execution of the application on different target devices. The Emulator enables you to approximate the experience a user has with an application on a particular device, and to test the portability of the application across different devices.

### **3. Packaging**

MIDP applications, or MIDlets, are packaged into a MIDlet suite, a grouping of MIDlets that can share resources at runtime. The following diagram illustrates how a MIDlet suite is organized.



**Figure: A.3 MIDlet Suite Components**

More formally, a MIDlet suite includes:

- A Java Application Descriptor (JAD) file. This file contains a predefined set of attributes (denoted by names that begin with “MIDlet-”) that allow application management software to identify, retrieve, and install the MIDlets. All attributes appearing in the JAD file are made available to the MIDlets. You can define your own application-specific attributes and add them to the JAD file.
- The JAR file contains:
  1. Java classes for each MIDlet in the suite.
  2. Java classes shared between MIDlets.
  3. Resource files used by the MIDlets (for example, image files).
  4. A manifest file describing the JAR contents and specifying attributes used by application management software to identify and install the MIDlet suite. (For more information on what attributes go into the manifest, see Appendix A, “MIDlet Attributes.”).

#### 4. Packaging Obfuscated Source Code

An additional feature of the J2ME Wireless Toolkit is the ability to build an obfuscated package. You are required to obtain a code obfuscator plug-in to use this feature. The JAR file for the code obfuscator should be placed in the *j2mewtk.dir\bin* directory.

The development of the MIDlet is done using Java 2 Micro Edition Wireless Tool Kit freely available at Sun site. J2MEWTK can be downloaded from <http://java.sun.com/products/j2mewtoolkit/>. Execute the installation file. The installer tries to locate your J2SE SDK; if it's having trouble, make sure you are pointing it to the directory where you installed the J2SE SDK. You will also need to specify whether the J2MEWTK will run by itself (standalone) or be integrated with Forte for Java. This article assumes you will be running the J2MEWTK in standalone mode. The files for the J2MEWTK will go into *c:\J2MEWTK* unless you specify a different directory, and the installer creates shortcuts for various parts of the toolkit. (Case-sensitivity vagaries in Windows may cause the directory name to appear as *J2mewtk*; this bit of caprice will not cause problems.) To run the toolkit itself, select the **KToolbar** shortcut. You should see the following screen.

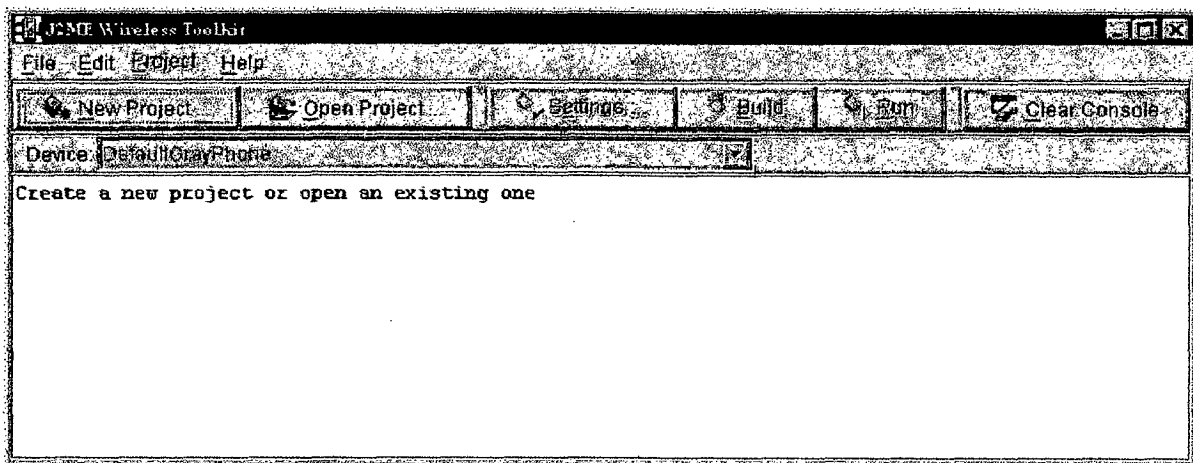


Figure: A.4 Window showing the Ktoolbar in J2MEWTK

#### A.2 TOMCAT Servlet Engine

Developed by members of the Jakarta Project, Tomcat 3.2.1 is a servlet container based on Java Servlet 2.2 technology and a JavaServer Pages[tm] (JSP[tm]) 1.1 implementation.



There are other products like it, but Tomcat is the official reference implementation for these two technologies, and one of the few available that offers support for JSP 1.1.

Tomcat version 3.2.1 is the latest quality release build. Version 4.x is in its first beta stage and supports the latest Servlet API 2.3 and JSP 1.2 specifications that are currently in development.

### **What's the Difference Between Tomcat and Apache JServ?**

Many people confuse Tomcat with Apache JServ. Tomcat 3.2.1 is a servlet container compliant with Servlet API 2.2 and JSP 1.1, while JServ is a container compliant with Servlet API 2.0 and provides no support for JSP technology. Early versions of Tomcat used some of the code written for JServ, especially the JServ Apache server adapter, but that's where the similarities ended. Tomcat now uses its own Apache server adapter, which will not work with JServ.

### **Deploying Tomcat**

Tomcat can be deployed as either a standalone product with its own internal Web server or in conjunction with several other Web servers, including:

Apache, version 1.3 or later

Netscape Enterprise Server, version 3.0 or later

Microsoft Internet Information Server (IIS), version 4.0 or later

Microsoft Personal Web Server, version 4.0 or later

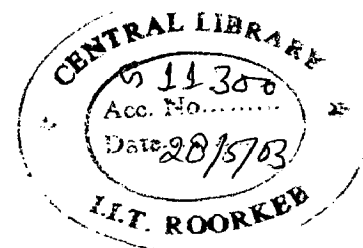
Tomcat requires a Java Runtime Environment that conforms to JRE 1.1 or later, including any Java 2 Platform Enterprise Edition (J2EE[tm]) system. Tomcat is easy to install and set up, and its features include automatic generation of Apache configuration files

Tomcat has several features that differentiate it from the competition, including:

**Support for JSP 1.1:** Tomcat's most important feature. Very few commercial-quality JSP implementations currently support the 1.1 specifications.

**Servlet API 2.2 compliance:** Because of this, Tomcat offers response output buffering and enhanced control over HTTP headers, as well as additional security and internationalization features.

Other features in Tomcat also make it worth consideration:



1. **Automatic servlet reloading:** Servlets can be configured to be reloaded when code changes. However, this is useful only in development and causes serious performance issues in a production environment.
2. **Thread-pooling and JVM load-balancing:** Servlet containers that are using a thread pool free themselves from directly managing their threads. Instead of allocating a new thread whenever they need one, they ask for it from the pool, and when they are done, the thread is returned to the pool. The thread pool can now be used to implement sophisticated thread-management techniques.
3. **Automatic deployment of Web Archive (WAR) files:** Servers that conform to the Java Servlet 2.2 specification are required to accept a Web Application Archive in a standard format. Applications are placed in the Webapps directory. The WAR files are automatically expanded and executed based on the information contained in the WAR file.
4. **NSAPI and ISAPI integration:** Integration with Netscape Enterprise Server 3.0+ and Microsoft IIS 4.0+ provides more flexibility in architecture and integration with existing Web infrastructure.
5. **A command-line JSP-to-servlet code tool:** The command line tool allows you to compile your JSP pages into byte code without having to request a page through the Tomcat server