INTEGRATION OF SECURITY IN MOBILE AGENT SYSTEMS

A DISSERTATION

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

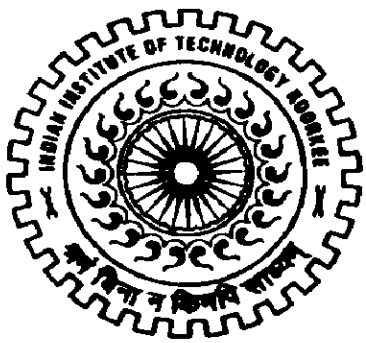
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

RICHA AGGARWAL



ER & DCI
NOIDA

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

FEBRUARY, 2003

CANDIDATE'S DECLARATION

This is to certify that the work, which is being presented in this dissertation, entitled "**INTEGRATION OF SECURITY IN MOBILE AGENT SYSTEMS**", 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 from August 2002 to February 2003, under the supervision of **Mr. P.N. Goswami**, Director, R&D, Electronics Research and Development Centre of India, Noida.

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

Date: 21.02.2003
Place: Noida

Richa
(Richa Aggarwal)

CERTIFICATE

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

Date: 21/02/2003.
Place: Noida

(Mr. P.N.Goswami)
Director (R&D),
ER&DCI, Noida

ACKNOWLEDGEMENT

I hereby take the privilege to express my deepest sense of gratitude for **Prof. Prem Vrat**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K. Verma**, Executive Director, ER&DCI, Noida for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Prof. A.K. Awasthi**, Dean Post Graduate Studies & Research and **Prof. R.P. Agrawal**, Course Coordinator, **Mr. V.N.Shukla** , Course Coordinator for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I have no words to thank my guide, **Mr. P.N.Goswami,** Director, ER&DCI, Noida for his guidance and invaluable suggestions during the entire course of this work. My sincere thanks are due to **Dr. P.R. Gupta** for the continuous inspiration and support she provided me with throughout the course of this dissertation. I am also grateful to **Mr. Munish Kumar** for the cooperation extended by him in the successful completion of this work.

I am highly indebted to **Mr. R.B. Patel**, Research Scholar, Roorkee for his constant support and direction in this work without which it could not have been completed.

It is impossible to mention the names of all those persons who have been involved, directly or indirectly, with this work and I extend my gratitude to all of them.

Richa
(Richa Aggarwal)
019037

CONTENTS

ABSTRACT

Protecting Mobile Agents against hosts is a key element of Mobile Agents Security. Security threats from a host/hosts to a mobile agent include masquerading, denial of service, eavesdropping, and alteration. In this thesis, methods have been studied to address mobile agent security issues; cryptographic techniques are reviewed including their advantages and limitations. This dissertation provides an overview of the range of threats faced by the designers of agent platforms and the developers of agent-based applications. The report also identifies generic security objectives, and a range of measures for countering the identified threats and fulfilling these security objectives. It is an attempt to secure mobile agents using standard cryptographic algorithms. i.e. using Data Encryption Standard , Digital Signature Algorithm, RSA using MD5 and Digital Certificate X.509. The project uses Aglets SDK as the mobile agent platform developed by IBM's Tokyo Research Lab for the development of secure data transfer. Aglet model is based on Java framework, making it platform independent. The Java policy files, security files and properties files are used to achieve the desired goal. These secure agents can be used for secure electronic commerce transactions, secure brokering.

Chapter 1

INTRODUCTION

1.1 Background

Mobile agents are programs capable of executing and migrating from node to node in a networking environment to perform tasks on behalf of their owners. They consist of three parts: code, data, and an execution state.

Mobile agent helps a user perform some task (or set of tasks), possibly by maintaining persistent state and communicating with its owner, other agents or its environment in general. They have code, data and authority so that

1. Communication bandwidth can be preserved.
2. Efficient & flexible service interfaces become practical.
3. To provide an attractive architecture for upgrading fielded systems with new software.
4. Distinct security requirements of mobile code systems are satisfied.
5. Useful mobile code units could access host resources.

An agent is a software object [1] that

1 is situated within an execution environment.

2 possess the following mandatory properties:

   a) Reactive - senses changes in the environment and acts accordingly.

   b) Autonomous - has control over its own actions.

   c) Goal-driven - is proactive.

   d) Temporally continuous - executes continuously.

3 possess one or more of the following orthogonal properties:

   a) Communicative - can communicate with other agents.

   b) Mobile - can travel from one host to another.

4 is learnable - adapts in accordance with previous experience.

5 is believable - appears believable to the end-user.

Advantages of mobile agents are that they reduce the network load, overcome network latency, encapsulate protocols execute asynchronously and autonomously, adapt dynamically, naturally heterogeneous, robust and fault-tolerant [2].

Mobile Agents find applications in electronic commerce, personal assistance (PDAs), secure brokering, distributed information retrieval, telecommunication networks services, monitoring and notification, information dissemination. , parallel processing.

The use of mobile agents can be traced back to the remote job entry systems in the 1960's. It has gradually gained popularity and complexity since then. Unlike many new technologies where security is an add-on feature after all intended functionalities are realized, security is a part of mobile agent's functionalities. Security poses a major threat in the mobile agent systems.

1.2 Objective of Dissertation

The objective is to implement and incorporate the security features such as authenticity, confidentiality, integrity and nonrepudation in the existing mobile agent platform Aglets.

Authentication will be implemented based on the shared secret via the Agent Transfer Protocol and involves three packets. For an instance lets suppose server A wants to dispatch an agent to server B, the protocol is as follows:

1. A ->B: nonce (A) (packet 1)
2. B->A: hash (nonce (A)+shared secret), nonce (B) (packet 2)
3. [A verifies that the hash is correct]
4. A->B: hash (nonce (B)+shared secret) (packet 3)
5. [B verifies that the hash is correct]

Assuming that hashes were correct, the agent and its state (its non-transient, non-null, serializable class objects) are transferred to B over the TCP socket.

The above protocol describe the basic security requirements of confidentiality, data integrity, authentication of origin, availability and non repudiation.

1.3 Scope of the Work

Mobile agent is an emerging technology that makes it much easier to design, implement, and maintain distributed systems. Mobile agents are programs that can be dispatched from one computer and transported to a remote computer for execution.

Aglet is a very popular mobile agent system. It is designed specifically for creating mobile agent applications and has a very complete and complex API for mobile

agent. The project uses Aglets SDK as the mobile agent platform developed by IBM's Tokyo Research Lab for the development of secure data transfer. Aglet model is based on Java framework so it is platform independent and agents can run on heterogeneous environment. There are no encryption services in the Aglet Framework.

The security features so implemented will secure the agent system using symmetric key algorithm, digital signature and digital certificate maintenance. With the help of these features aglet will be a secure agent and thus help the owner and receiver in secure transactions.

The work is carried out in the following sequence. Firstly Symmetric Algorithms that is DES (data encryption standard) is implemented in which encryption key and decryption keys are same. So user sends the key via an alternate path. Then Digital Signatures - Asymmetric algorithms are implemented in which data is encrypted using private key and decrypted using public key that is Public Key Infrastructure (PKI) i.e. RSA with MD5, digital signatures using DSA (digital signature algorithm). Then digital certificates are implemented which is a directory server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address as well as other attributes and information about users.

1.4 Organisation of the Thesis

The chapter 1 gave a brief overview of mobile agents along with their applications and then discussed the objective and scope of the work. The chapter 2 presents a detailed literature survey of security issues related to mobile agents and discusses relevant theoretical issues to protect mobile agents. Then chapter 3 presents a detailed overview of the security algorithms available and their mechanisms to implement security features. Chapter 4 presents the detailed study of agent systems – aglets covering aglet elements, anatomy and trajectory. Chapter 5 presents the design of agents to implement security-describing Aglets – Tahiti Server, the policy files and all the features to be embedded in respective agents. The following chapter presents the user interfaces so implemented for each agent along with its explanation. The last chapter presents the summary of work done in implementing security, its weakness and work that

Chapter 2

SECURITY ISSUES

Security is a fundamental concern for a mobile agent system. Security is a severe concern and it is regarded as the primary obstacle to adopting mobile agent systems. The operation of a mobile agent system is subjected to various agreements, whether declared or tacit. The parties they are intended to serve can violate these agreements accidentally or intentionally. A mobile agent system can also be threatened by parties outside of the agreements that is by create rogue agents, they may hijack existing agents.

There are a variety of desirable security goals for a mobile agent system. Most of these concern the interaction between agents and interpreters. The user on behalf of whom an agent operates wants it to be protected to the extent possible from malicious interpreters and from the intermediate hosts, which are involved in its transmission. Conversely, an interpreter, and the site at which it operates, needs to be protected from malicious or harmful behavior by an agent [3].

In mobile agents, one of the primary motivations is to allow a broad range of users access to a broad range of services, which are offered by different frequently competing organizations. Thus, in all the applications, all the parties do not trust each other. The parties require a degree of trust among the participants.

2.1 Security Threats

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information [4]. There are a variety of ways to examine these classes of threats as they apply to agent systems. So the components of an agent system can be used to categorize the threats, as it is the best way to identify the possible source and target of attacks. The main threat categories are -

a Agent attacking an agent platform.

b Agent platform attacking an agent.

c Agent attacking another agent.

d Agent attacking an agent on another agent platform.

All these attacks are shown in figure 2.1.1.

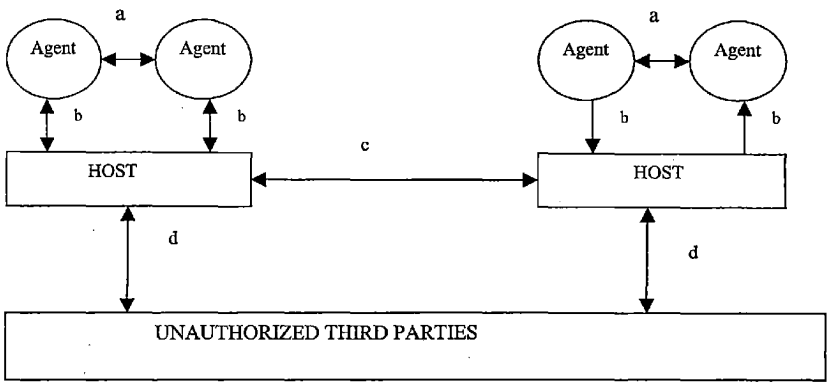


Figure 2.1.1:Security issues in Mobile Agent Systems

2.1.1 Agent to Platform

This category represents the set of threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. These sets of threats are described below.

2.1.1.1 Masquerading

When an unauthorized agent claims the identity of another agent it is said to be masquerading. The masquerading agent may pose as an authorized agent in an effort to gain access to services and resources to which it is not entitled. The masquerading agent may also pose as another unauthorized agent in an effort to shift the blame for any actions for which it does not want to be held accountable and it may also damage the trust the legitimate agent has established in an agent community and its associated reputation.

2.1.1.2 Denial Of Service

Mobile agents can launch denial of service attacks by consuming an excessive amount of the agent platform's computing resources. This denial of service attacks can be launched intentionally by running attack scripts to exploit system vulnerabilities, or unintentionally through programming errors. The mobile computing paradigm, however, requires an agent platform to accept and execute an agent whose code may have been

developed outside its organization and has not been subject to any a prior review. A rogue agent may carry malicious code that is designed to disrupt the services offered by the agent platform, degrade the performance of the platform, or extract information for which it has no authorization to access.

2.1.1.3 Unauthorized Access

Each agent visiting a platform must be subject to the platform's security policy. Applying the proper access control mechanisms requires the platform or agent to first authenticate a mobile agent's identity before it is instantiated on the platform. An agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. A platform that hosts agents representing various users and organizations must ensure that agents do not have read or write access to data for which they have no authorization, including access to residual data that may be stored in a cache or other temporary storage.

2.1.2 Agent To Agent

These sets of threats include various ways in which one agent can harm another agent and are given below.

2.1.2.1 Masquerade

Agent-to-agent communication can take place directly between two agents or may require the participation of the underlying platform and the agent services it provides. In either case, an agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating. An agent may pose as a well-known vendor of goods and services for example, and try to convince another unsuspecting agent to provide it with credit card numbers, bank account information, some form of digital cash, or other private information.

2.1.2.2 Denial Of Service

In addition to launching denial of service attacks on an agent platform, agents can also launch denial of service attacks against other agents. Agent communication languages and conversation policies must ensure that a malicious agent doesn't engage another agent in an infinite conversation loop or engage the agent in elaborate conversations with the sole purpose of tying up the agent's resources. Malicious agents

can also intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in a timely manner.

2.1.2.3 Repudiation

Repudiation occurs when an agent, participating in a transaction or communication, later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious disputes that may not be easily resolved unless the proper countermeasures are in place. Since an agent may repudiate a transaction as the result of a misunderstanding, it is important that the agents and agent platforms involved in the transaction maintain records to help resolve any dispute.

2.1.2.4 Unauthorized Access

Modification of an agent's code is a particularly insidious form of attack, since it can radically change the agent's behavior (e.g., turning a trusted agent into malicious one). An agent may also gain information about other agent's activities by using platform services to eavesdrop on their communications.

2.1.3 Platform to Agent

It represents the threats in which platforms compromise the security of agents. These set of threats are described below.

2.1.3.1 Masquerade

One agent platform can masquerade as another platform in an effort to deceive a mobile agent as to its true destination and corresponding security domain. An agent platform masquerading as a trusted third party may be able to lure unsuspecting agents to the platform and extract sensitive information from these agents.

2.1.3.2 Denial of service

When an agent arrives at an agent platform, it expects the platform to execute the agent's requests faithfully, provide fair allocation of resources, and abide by quality of service agreements. A malicious agent platform, however, may ignore agent service requests, introduce unacceptable delays for critical tasks such as placing market orders in a stock market, simply not execute the agent's code, or even terminate the agent without notification.

2.1.3.3 Eavesdropping

Since the platform has access to the agent's code, state, and data, the visiting agent must be wary of the fact that it may be exposing proprietary algorithms, trade secrets, negotiation strategies, or other sensitive information. Even though the agent may not be directly exposing secret information, the platform may be able to infer meaning from the types of services requested and from the identity of the agents with which it communicates.

2.1.3.4 Alternation

Since an agent may visit several platforms under various security domains throughout its lifetime, mechanisms must be in place to ensure the integrity of the agent's code, state, and data. A compromised or malicious platform must be prevented from modifying an agent's code, state, or data without being detected.

2.1.4 Other - To - Agent Platform

It represents the set of threats in which external entities, including agents and agent platforms, threaten the security of an agent platform. These threats are briefly explained below.

2.1.4.1 Masquerade

An agent on a remote platform can masquerade as another agent and request services and resources for which it is not authorized. Agents masquerading as other agents may act in conjunction with a malicious platform to help deceive another remote platform or they may act alone.

2.1.4.2 Unauthorized Access

Remote users, processes, and agents may request resources for which they are not authorized. Remote access to the platform and the host machine itself must be carefully protected, since conventional attack scripts freely available on the Internet can be used to subvert the operating system and directly gain control of all resources.

2.1.4.3 Denial of Service

The agent services offered by the platform and inter-platform communications can be disrupted by common denial of service attacks. Agent platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols.

2.1.4.4 Copy and replay

Every time a mobile agent moves from one platform to another it increases its exposure to security threats. A party that intercepts an agent, or agent message, in transit can attempt to copy the agent, or agent message, and clone or retransmit it.

2.2 Security Requirements

The users of networked computer systems have following main security requirements [5].

1) **Confidentiality** : assurance that communicated information is not accessible to unauthorized parties.

2) **Data Integrity** :assurance that communicated information cannot be manipulated by unauthorized parties without being detected.

3) **Authentication Of Origin** :assurance that communication originates from its claimant.

4) **Availability** :assurance that communication reaches its intended recipient in a timely fashion.

5) **Non-Repudiation** : assurance that the originating entity can be held responsible for its communications.

2.3 Countermeasures

Most agent systems rely on a common set of baseline assumptions regarding security. The first is that an agent trusts the home platform where it is instantiated and begins execution. The second is home platform and other equally trusted platforms are implemented securely, with no flaws or trapdoors that can be exploited, and behave non-maliciously. The third is public key cryptography, primarily in the form of digital signature, is utilized through certificates and revocation lists managed through a public key infrastructure. We can protect the platform and agent by following methods [6].

2.3.1 Protecting the Platform

2.3.1.1 Software-Based Fault Isolation

Mechanisms to isolate processes from one another and from the control process.

2.3.1.2 Safe Code Interpretation

Mechanisms to control access to computational resources.

2.3.1.3 Signed Code

Cryptographic methods to encipher information exchanges.

2.3.1.4 Authorization and Attribute Certificates

Cryptographic methods to identify and authenticate users, agents, and platforms,

2.3.1.5 State Appraisal

Mechanisms to audit security-relevant events occurring at the agent platform.

2.3.1.6 Path Histories

Mechanisms to track the path of mobile agent.

2.3.2 Protecting the Agent

2.3.2.1 Partial Result Encapsulation

This technique deals with encapsulating information depending on the encapsulation capabilities of the agent and there by relying on a third party to timestamp using digital certificates.

2.3.2.2 Mutual Itinerary Recording

This technique deals with tracking the path history and so detecting the malicious behavior of agents.

2.3.2.3 Execution Tracing

Execution tracing is a technique for detecting unauthorized modifications of an agent through the faithful recording of the agent's behavior during its execution on each agent platform.

2.3.2.4 Environmental Key Generation

The approach centers on constructing agents in such a way that upon encountering an environmental condition (e.g., string match in search), a key is generated, which is used to unlock some executable code cryptographically.

2.3.2.5 Computing with Encrypted Functions

The goal of Computing with Encrypting Functions is to determine a method whereby mobile code can safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments and operates autonomously without interactions with the home platform.

2.3.2.6 Obfuscated Code (Time Limited Black box)

The strategy behind this technique is simple -- scramble the code in such a way that no one is able to gain a complete understanding of its function (i.e., specification and data), or to modify the resulting code without detection. A serious problem with the general technique is that there is no known algorithm or approach for providing Black box protection.

2.4 Advantages Of Working Environment - Java

Java is an object-oriented network-savvy programming language. It is the language of the Internet. Some of the properties of Java that make it a good language for mobile agent programming are given below [7]

2.4.1 Platform-Independence

Java is designed to operate in heterogeneous networks. To enable a Java application to execute anywhere on the network, the compiler generates architecture-neutral byte code, as opposed to non-portable native code. For this code to be executed on a given computer, the Java runtime system needs to be present. There are no platform-dependent aspects of the Java language.. Even libraries are platform-independent parts of the system. It allows us to create a mobile agent without knowing the types of computers it is going to run on.

2.4.2 Secure Execution

Java simply does not allow illegal type casting or any pointer arithmetic. Programs are no longer able to forge access to private data in objects that they do not have access to. This prevents most activities of viruses. Even if someone tampers with the

byte code, the Java runtime system ensures that the code will not be able to violate the basic semantics of Java. The security architecture of Java makes it reasonably safe to host an untrusted agent, because it cannot tamper with the host or access private information.

2.4.3 Dynamic class loading

This mechanism allows the virtual machine to load and define classes at runtime. It provides a protective name space for each agent, thus allowing agents to execute independently and safely from each other. The class-loading mechanism in extensible and enables classes to be loaded via the network.

2.4.4 Multithread Programming

Agents are by definition autonomous. That is, an agent executes independently of other agents residing within the same place. Allowing each agent to execute in its own lightweight process, also called a thread of execution, is a way of enabling agents to behave autonomously. Fortunately, Java not only allows multithread programming, but also supports a set of synchronization primitives that are built into the language. These primitives enable agent interaction.

2.4.5 Object Serialization

A key feature of mobile agents is that they can be serialized and deserialized. Java conveniently provides a built-in serialization mechanism that can represent the state of an object in a serialized form sufficiently detailed for the object to be reconstructed later.

2.4.6 Reflection

Java code can discover information about the fields, methods, and constructors of loaded classes, and can use reflected fields, methods, and constructors to operate on their underlying counterparts in objects, all within the security restrictions.

2.5 Currently available Mobile Agents

2.5.1 Aglets

This system mirrors the applet model in Java. The goal was to bring the flavor of mobility to the applet. It is a research work of IBM LABS, Tokyo. This platform is studied in this thesis.

2.5.2 Concordia

Mitsubishi's Concordia is a framework for the development and management of mobile agent applications, which extend to any system supporting Java. Like most Java-based systems, it provides agent mobility using Java's serialization and class loading mechanisms, and does not capture execution state at the thread level. Each agent object is associated with a separate Itinerary object, which specifies the agent's migration path (using DNS hostnames) and the methods to be executed at each host. Concordia has extensive support for agent communication. Agent state is protected during transit, as well as in persistent stores, using encryption protocols. Each agent is associated with a particular user, and carries a one-way hash of that user's password.

2.5.3 Voyager

Object Space's Voyager is a platform for agent-enhanced distributed computing in Java. While Voyager provides an extensive set of object messaging capabilities, it also allows object to move as agents in the network. Voyager combines the properties of a Java-based object request broker with those of a mobile agent system. In this way Voyager allows Java programmers to create network applications using both traditional and agent-enhanced distributed programming techniques.

2.5.4 Agent Tcl

Agent Tcl, developed at Dartmouth College, allows Tcl scripts to migrate between servers that support agent execution, communication, status queries and non-volatile storage. A modified Tcl interpreter is used to execute the scripts, and it allows the capture of execution state at the thread level. When an agent migrates, its entire source code, data and execution state is transferred. Migration is absolute, and the destination is specified using a location-dependent name. It is also possible to clone an agent and dispatch it to the desired server. Agents have location-dependent identifiers based on DNS hostnames, which therefore change upon migration. Inter-agent communication is accomplished either by exchanging messages or setting up a stream connection. Agent Tcl uses the Safe Tcl execution environment to provide restricted resource access. It ensures that agents cannot execute dangerous operations without the appropriate security mediation

2.6 Mobile Agent Framework

A mobile agent environment or mobile agent system is a framework that implements the mobile agent paradigm. It provides services and primitives that help in the use, implementation and execution of systems developed using the mobile agents paradigm.

2.6.1 Mobile Agent Standardization: MASIF

All the mobile agent systems differ widely in architecture and implementation, thereby impeding interoperability and rapid deployment of mobile agent technology in the marketplace. To promote interoperability, some aspects of mobile agent technology must be standardized. The companies Crystaliz, General Magic Inc., GMD Fokus, IBM Corporation, and the Open Group have jointly developed a proposal for a Mobile Agent System Interoperability Facility (MASIF) and brought it to the attention of the Object Management Group (OMG). MASIF addresses the interfaces between agent systems, not between agent applications and agent systems. Even though the former seems to be more relevant for application developers, it is the latter that allows mobile agents to travel across multiple hosts in an open environment. MASIF is clearly not about language interoperability. Language interoperability for mobile objects is very difficult and MASIF is limited to interoperability between agent systems written in the same language, but potentially by different vendors. Furthermore, MASIF does not attempt to standardize local agent operations such as agent interpretation, serialization, or execution. MASIF standardizes the following four areas:

1) **Agent Management.** There is interest in the mobile agent community to standardize agent management. It is clearly desirable that a system administrator who manages agent systems of different types can use the same standard operations. It should be possible to create an agent given a class name for the agent, suspend an agent's execution, resume its execution, or terminate it in a standard way.

2) **Agent Transfer.** It is desirable that agent applications can spawn agents that can freely move among agent systems of different types, resulting in a common infrastructure.

3) **Agent and Agent System Names.** In addition to standardizing operations for interoperability between agent systems, the syntax and semantics of various parameters must be standardized too. Specifically, agent name, and agent system name should be

standardized. This allows agent systems and agents to identify each other, as well as applications to identify agents and agent systems.

4) Agent System Type and Location Syntax. The location syntax must be standardized so that an agent can access agent system type information from a desired destination agent system. The agent transfer can only happen if the destination agent system type can support the agent. Location syntax also needs to be standardized so that agent systems can locate each other.

This architecture is shown in Fig. 2.6.1. An agent region is defined as a set of agent systems that can access each other, possessing similar authority and identifying a default migration pattern. Mobile agent facilities include the storage and retrieval of agents, remote agent creation transfer and agent method invocation. The agent system is loaded on the operating system. There can be different agent system on the same machine. Each agent system consists of the place and communication infrastructure, which is required for communication between two agent environments. Agent to agent communication is possible between two agents on same or different machine. The agent communication is based on the protocol used by the system or it is through its own protocol called as agent transfer protocol [8].

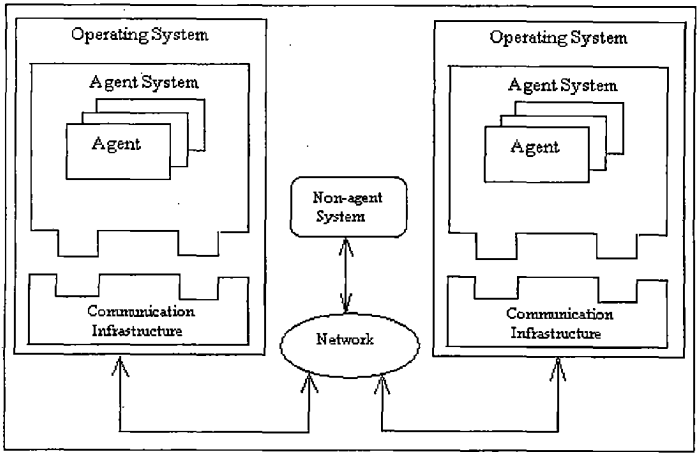


Fig.2.6.1: Mobile Agent Facility Architecture

Chapter 3

CRYPTOGRAPHY BACKGROUND

3.1 Cryptography Basics

In today's world where most of the transactions and transfer of confidential documents takes place in the digital form, there is a need for a highly secure means of communication so that we can rely on the ingenuity of the data received. This is where the security concept is considered. The process of disguising a message (plain text) in such a way so as to hide its substance is known as encryption. The encrypted message is known as ciphertext. The process of converting ciphertext to plaintext again is known as decryption .The figure 3.1.1 explains it. The art and science of keeping messages secure is called cryptography. The art and science of breaking ciphertext is known as cryptanalysis and the practitioners of cryptanalysis are called cryptanalysts. The branch of mathematics encompassing both cryptography and cryptanalysis is Cryptology [9].

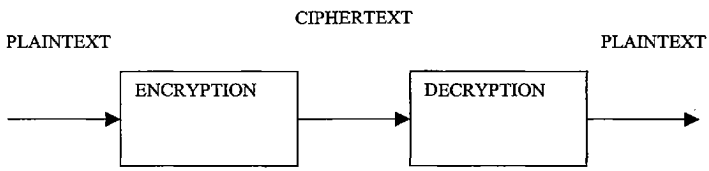


Figure 3.1.1: Simple Diagram Of Message Encryption And Decryption

3.1.1 Symmetric Algorithm

Symmetric algorithms are called conventional algorithms where the encryption key can be calculated from the decryption key and vice versa. In most of these algorithms the encryption as well as the decryption keys are same. These algorithms are also called secret key algorithms or one key algorithm. This requires that sender and receiver agree upon a key before they can communicate securely as shown in figure 3.1.1.1. For Example DES, Blowfish, RC5 etc

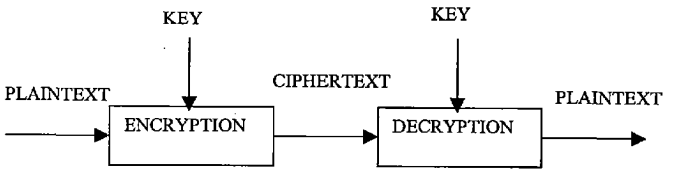


Figure 3.1.1.1:Encryption and Decryption with a Single Key

3.1.2 Asymmetric Algorithm

They are designed so that the key for encryption is different from the key used for decryption. Furthermore the decryption key cannot be calculated from the encryption key. These algorithms are called public key because the encryption key can be made public as shown in figure 3.1.1.2. For example Public Key Infrastructure using Digital Signature Algorithm and one way hash function – MD5 , SHA-1.

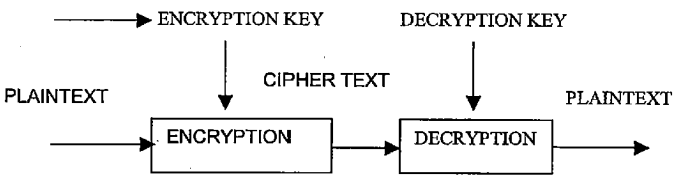


Figure 3.1.1.2. Encryption with a different key and decrypting with a different key.

3.2 Data Encryption Standard

The Data Encryption Standard (DES) specifies a FIPS approved cryptographic algorithm as required by FIPS 140-1. This publication provides a complete description of a mathematical algorithm for encrypting (enciphering) and decrypting (deciphering) binary coded information. Encrypting data converts it to an unintelligible form called cipher. Decrypting cipher converts the data back to its original form called plaintext. The algorithm described in this standard specifies both enciphering and deciphering operations, which are based on a binary number called key. Data can be recovered from

cipher only by using exactly the same key used to encipher it. Unauthorized recipients of the cipher who know the algorithm but do not have the correct key cannot derive the original data algorithmically. However, anyone who does have the key and the algorithm can easily decipher the cipher and obtain the original data. A standard algorithm based on a secure key thus provides a basis for exchanging encrypted computer data by issuing the key used to encipher it to those authorized to have the data [5].

3.3 Digital Signature

Digital signatures uses "public key cryptography," which employs an algorithm using two different but mathematically related "keys;" one for creating a digital signature or transforming data into a seemingly unintelligible form, and another key for verifying a digital signature or returning the message to its original form [5].

Thus, use of digital signatures usually involves two processes, one performed by the signer and the other by the receiver of the digital signature.

(a) Digital Signature Creation: uses a hash result derived from and unique to both the signed message and a given private key. For the hash result to be secure there must be only a negligible possibility that the same digital signature could be created by the combination of any other message or private key.

(b) Digital Signature Verification: is the process of checking the digital signature by reference to the original message and a given public key, thereby determining whether the digital signature was created for that same message using the private key that corresponds to the referenced public key as shown in figure 3.3.1. Various asymmetric cryptosystems create and verify digital signatures using different algorithms and procedures, but share this overall operational pattern.

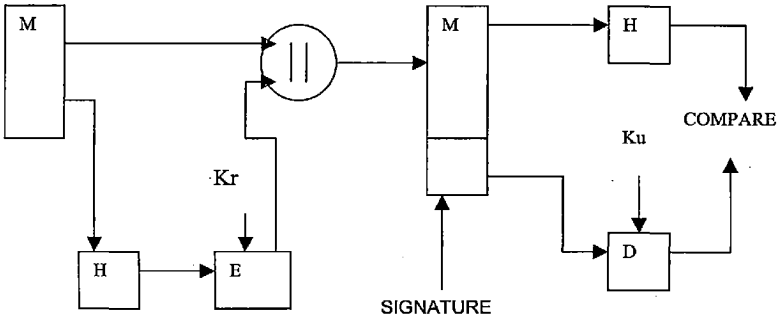


Figure 3.3.1: Explaining How Digital Signature Works

3.3.1 Digital Signature Generation Using RSA & MD5

In this type of signature generation data (i.e. the message M) as shown in figure 3.3.1 first through a one-hash function (usually we use MD5). The output of one-way hash function is encrypted using public key and then the cipher text so produced is attached to the message as the digital signature. At the receiving side, message is separated from the signature and then hash function is applied on the message. The signature is decrypted using private key and then it is compared with the hashed output of message .If both are same, then message is accepted else it is rejected as it is tampered by an intruder.

RSA KEYS

1	P, Q - two prime numbers	(PRIVATE, CHOSEN)
2	$N = P*Q$	(PUBLIC, CALCULATED)
3	$F(N) = (P-1)(Q-1)$	(PUBLIC, CALCULATED)
4	E , WITH GCD(F(N) , E) = 1 , 1<E<F(N)	(PUBLIC , CHOSEN)
5	$d = e^{-1} \mod(F(N))$	(PRIVATE,CALCULATED)

C= cipher text. , M = message

The Encryption is $C = M^e \pmod n$

The Decryption is $M = C^d \pmod n$

The one way hash function MD5 compression function is

A = B+ ((A+G(B.C.D) +X[K] +T[I])<<<S)

A, B,C , D = the four words in buffer

G = one of the primitive functions

<<<s = circular shift function

X{K] = M[Q*16+K] = the kth 32 bit word in the q^{th} 512 th bit block of message

T[i] = addition modulo 2^{32}

3.3.2 Digital Signature Using Digital Signature Algorithm.

In this type of signature generation data (i.e. the message M) is first through a one way hash function (SHA -1) as shown in figure 3.3.1. The output of one-way hash function is encrypted using private key and then the cipher text so produced is attached to the message as the digital signature. At the receiving side, message is separated from the signature and then hash function is applied on the message. The signature is decrypted using public key and then it is compared with the hashed output of message .If both are same, then message is accepted else it is rejected as it is tampered by an intruder. This digital signature can be verified by the third parties also and so it is used for digital certificates also using keystore file as the database file.

3.4 Digital Certificates

ITU-T recommendation X-509 is part of the X-500 series of recommendations that define a directory service. The directory is a server or distributed set of servers that maintains a database of information about users. The Information includes a mapping from user name to network address as well as other attributes and information about users Digital certificates authenticate that their holders - people, web sites, and even network resources such as routers - are truly who or what they claim to be. It contains, among other fields, a serial number, the subject name, the subject's public key, and the issuer's

name. The issuer, or Certificate Authority, digitally signs the certificate to provide integrity protection and assurance that the certificate is authentic [5].

Elements of an attribute certificate are pictured in figure 3.5.1. It include the identity of the owner (formed by a secure hash over the agent's code and information), the identity of the issuer, the identifier of the algorithms used to protect the certificate, the lifetime of the certificate, and the subject attributes, which may be expressed as simple type-value pairs or as more complex syntactical expressions. These elements can be used to establish the validity of the certificate and the binding between the attribute certificate and the agent. They are electronic files that act like a kind of online passport. They are tamper-proof and cannot be forged. . It uses policy files, security files and keystore entries for the same. Digital Certificate uses digital signature with one way hash function ie RSA or DSA with MD5 or SHA 1 (can be used described in the earlier section). Java supports a special password-protected database of private keys and their associated digital certificates called the key store, and its contents used when signing JAR files [10].

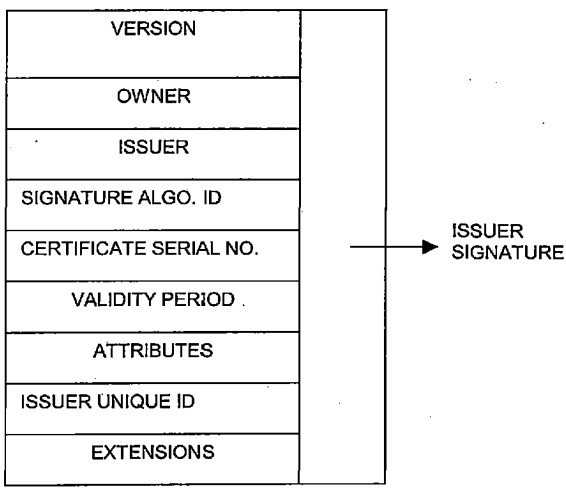


Figure 3.4.1: Attribute of Certificate Elements

3.4.1 Policy Files

Java provides a single system-wide policy file and an optional user policy file, as well as a tool for specifying other policies. Each entry in a policy file indicates the set of permissions authorized for code from a specified code source. Policy rules are expressed using a grant-style policy specification language, whereby all permissions are denied unless explicitly assigned to a code source. Permissions represent authorized actions on system objects. The loader uses the assigned permissions to manage the name space and form a protection domain for any loaded code. Actions attempted by the code are checked against the domain permissions via the security manager. Besides standard Java permissions, developers may also define permissions specific to an application.
The figure 3.4.1.1 shows a Java based agent system with enhancements using policy files and keystore entries.

3.4.2 Keystores

A keystore is a database of private keys and their associated certificates or certificates chains, which authenticate the corresponding public keys . The keystore format is provided by sun Microsystems in Java. This format protects the integrity of the entire keystore with a keystore password .A hash value of the entire keystore is used to protect the keystore from alternation. A keystore can contain two types of entries: the trusted certificate entries, and key/certificate entries, each containing a private key and the corresponding public key certificate. Each entry in a keystore is identified by an *alias*. A keystore owner can have multiple keys in the keystore, accessed by different aliases. An alias is typically named after a particular role in which the keystore owner uses the associated key. An alias may also identify the purpose of the key [11]. The keytool tool can be used to:

1. Create private keys and their associated public key certificates
2. Issue certificate requests, which you send to the appropriate certification authority
3. Import certificate replies, obtained from the certification authority user contacted.
4. Import public key certificates belonging to other parties as trusted certificates

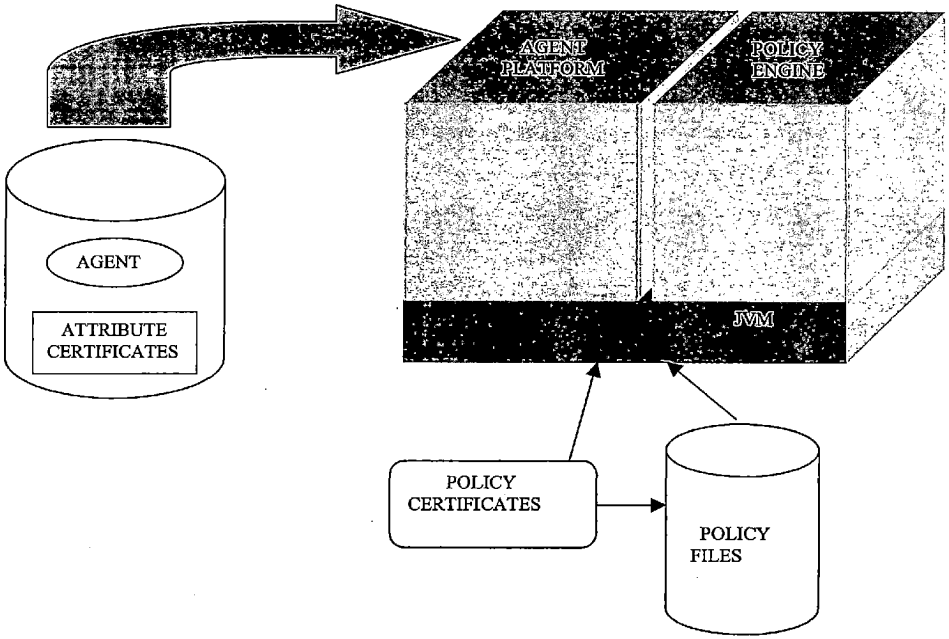


Figure 3.4.1.1: A Java Based System with Enhancements

Chapter 4

AGLET SYSTEM STUDY

The Aglets Software Development Kit (ASDK) is an implementation of Aglet API designed by IBM's Tokyo research lab. Aglet mirrors the applet model in Java. The goal is to bring the flavor of mobility to the applet. The term aglet is indeed a portmanteau word combining agent and applet [7].

4.1 Basic Elements

The ASDK model defines a set of abstractions and the behavior needed to leverage mobile agent technology in Internet-like open wide-area networks. The key abstractions are as follows.

4.1.1 Aglet.

An Aglet is a mobile Java object that visits aglet-enabled hosts in a computer network. It is autonomous, since it runs in its own thread of execution after arriving at a host, and reactive, because of its ability to respond to incoming messages.

4.1.2 Proxy.

A proxy is a representative of an aglet. It serves as a shield for the aglet that protects the aglet from direct access to its public methods. The proxy also provides location transparency for the aglet; that is, it can hide the aglet's real location of the aglet.

4.1.3 Context

A context is an aglet's workplace. It is a stationary object that provides a means for maintaining and managing running aglets in a uniform execution environment where the host system is secured against malicious aglets. One node in a computer network may run multiple servers and each server may host multiple contexts. Contexts are named and can thus be located by the combination of their server's address and their name.

4.1.4 Message

A message is an object exchanged between aglets. It allows for synchronous as well as asynchronous message passing between aglets. Message passing can be used by aglets to collaborate and exchange information in a loosely coupled fashion.

4.1.5 Future Reply

A future reply is used in asynchronous message sending as a handler to receive a result later asynchronously.

4.1.6 Identifier

An identifier is bound to each aglet. This identifier is globally unique and immutable throughout the lifetime of the aglet. Behavior supported by the aglet object model is based on a careful analysis of the "life and death" of mobile agents. There are basically only two ways to bring an aglet to life either it is instantiated from scratch (creation) or it is copied from an existing aglet (cloning). To control the population of aglets user can of course destroy aglets (disposal). Aglets are mobile in two different ways: active and passive.

The figure 4.1.1 gives a block diagram of Aglets API.

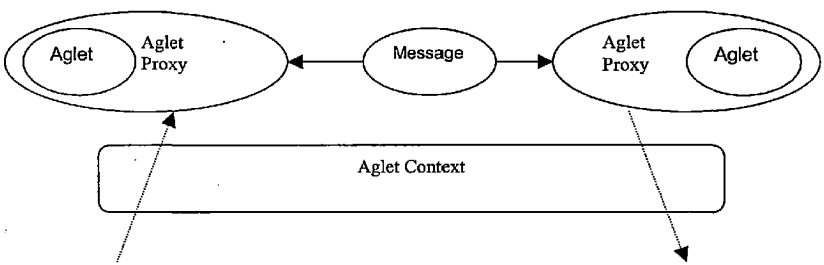


Figure 4.1.1 Aglet API

4.2 Aglet Anatomy

The fundamental sets of operations of aglets are designed with a view of distributed mobile agent environment. It has a lightweight API that is both easy to learn to use and sufficiently complete and robust for real applications. Aglets API is often called the "RISC" of mobile agents. The fundamental operations are described below.

4.2.1 Creation

The creation of an aglet takes place in a context. The new aglet is assigned an identifier, inserted into the context, and initialized. The aglet starts executing as soon as it has been successfully initialized.

4.2.2 Cloning

The cloning of an aglet produces an almost identical copy of the original aglet in the same context. The only differences are the assigned identifier and the fact that execution restarts in the new aglet. Note that execution threads are not cloned.

4.2.3 Dispatching

Dispatching an aglet from one context to another will remove it from its current context and insert it into the destination context, where it will restart execution (execution threads do not migrate).

4.2.4 Retraction

The retraction of an aglet will pull (remove) it from its current context and insert it into the context from which the retraction was requested.

4.2.5 Activation and deactivation

The deactivation of an aglet is the ability to temporarily halt its execution and store its state in secondary storage. Activation of an aglet will restore it in a context.

4.2.6 Disposal

The disposal of an aglet will halt its current execution and remove it from its current context.

4.2.7 Messaging

Messaging between aglets involves sending, receiving, and handling messages synchronously as well as asynchronously.

These operations can be explained from figure 4.2.1.

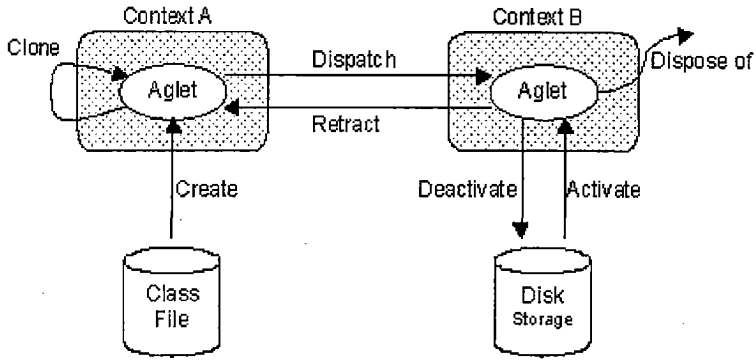


Figure 4.2.1: Aglet life cycle model explaining all the fundamental operations

4.3 Aglet Trajectory

The block diagram is shown is shown in figure 4.3.1. In the first step the agent packs the byte code (Java class code) and the current state in source host .Due the agent transfer as shown in step 2, the agent disappears from Aglet will disappear from the current host machine and reappear in the same state at the specified destination. First, a special technique called object serialization is used to preserve the state information of the Aglet by making a sequential byte representation of the Aglet. Next, this representation is passed to the underlying transfer layer that brings the Aglet (byte code and state information) safely over the network. i.e. on the host B. Finally, the transferred bytes are de-serialized to recreate the aglet's state.

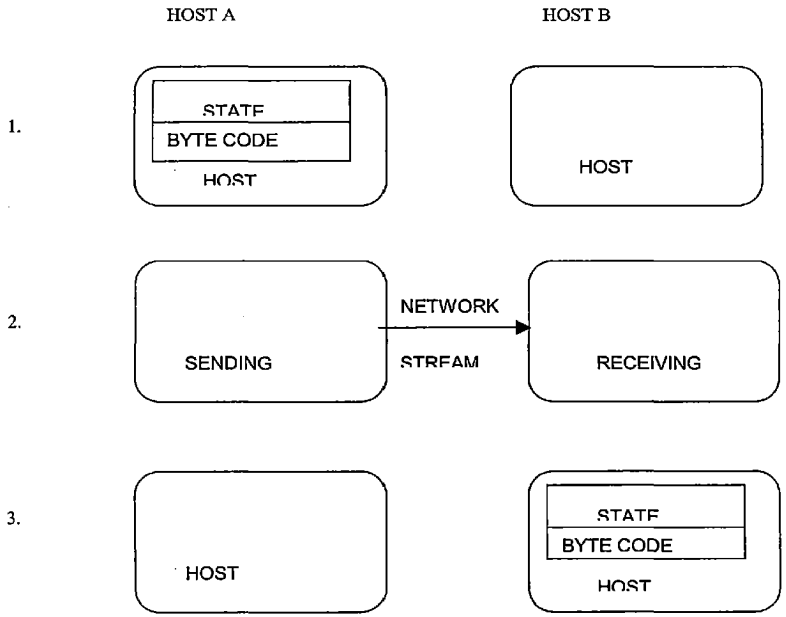


Figure 4.3 .1:Transfer Of an Agent

There are no encryption services in the Aglet Framework. This project is to protect the Aglets' state (that is the data objects the Aglet carries) from tampering, eavesdropping and replay attacks. Replay attacks are effectively prevented by the ATP authentication protocol but if user uses code signing then no such security mechanism are supported by Aglet Framework.

Chapter 5

DESIGN AND IMPLEMENTATION

5.1 Design Issues

Aglet is a very popular mobile agent system. It is designed specifically for creating mobile agent applications and has a very complete and complex API for mobile agent .The project will use Aglets SDK as the mobile agent platform developed by IBM's Tokyo Research Lab for the development of secure data transfer. The Aglets Software Development Kit (ASDK) is an implementation of the Aglet API. It includes Aglet API packages, documentation, sample agents, and the Tahiti aglet server. This Aglet Workbench works on JDK1.1 or higher versions. It is qualified to run on Win95/NT and SPARC/Solaris 2.5 [12].

This package includes

- Documentation
  - Release Notes
  - Documentation of Aglet API
  - Description of sample programs
- Software
  - Aglets Library
  - Aglets Server + Tahiti Aglets Viewer
  - sample aglets
  - script file for execution of the Tahiti (agletsd)

5.1.1 Running the Aglet Server

For launching the agents first of all user need to start the aglet server.

Aglet server is started using the script file 'c:\aglets-2.0.1\bin\agletsd'. This aglet server will invoke an aglet viewer, named Tahiti, for managing aglets. The Tahiti window is as shown in Fig.5.1.1.1.

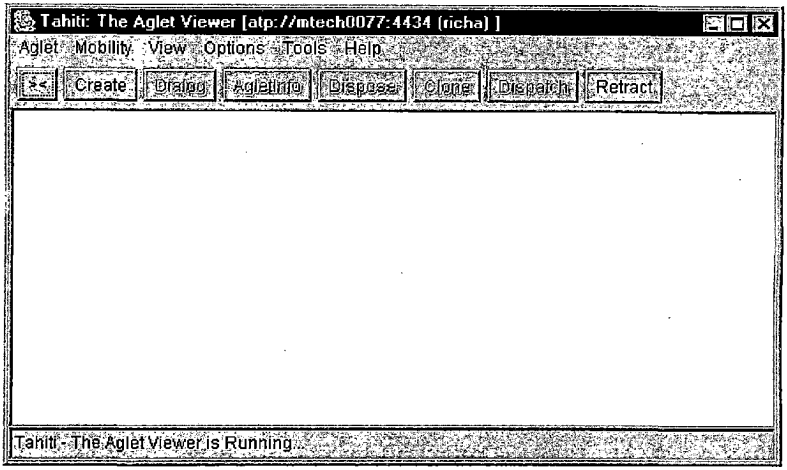


Fig.5.1.1.1: Tahiti Server

5.1.2 Design of Mobile Agents

A simple agent consists of basically the main class and two methods onCreation() and run() [7]. First start by importing the aglet package, which contains all the definitions of the Aglet API. Next define the MyFirstAglet class, which inherits from the Aglet class:

```
import com.ibm.aglet;
public class MyFirstAglet extends Aglet {
        // aglet's methods here....
}
```

For example, if user want your aglet to perform some specific initialization when it is created, user can override its onCreation method:

```
public void onCreation(Object init) {
        //Do some initialization here....
}
```

When an aglet has been created or when it arrives in a new context, it is given its own thread of execution through a system invocation of its run method. The run method is called every time the aglet arrives at or is activated in a new context. So the run method becomes the main entry point for the aglet's thread of execution.

```
public void run() {
        //Do something else here...
}
```

5.1.3 Policy Files for Aglets

The main steps of policy entry in policy file in a java enabled system is -

1. Start policy tool
2. Make corresponding entry in it.
3. Save the file

Whenever Policy Tool is started, it tries to fill in the window with policy information from "user policy file" as explained in chapter 3. The user policy file is by default a file named ". Java.policy" in user home directory . The Policy Tool file is shown in the figure 5.1.3.1.

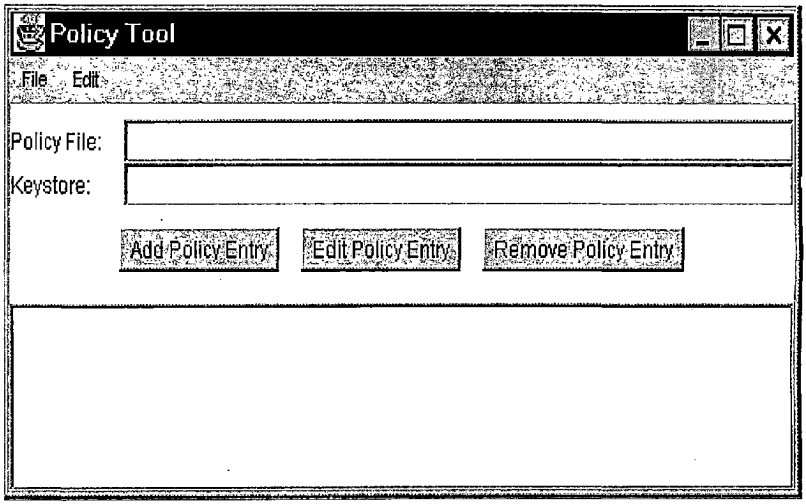


Figure 5.1.3.1: The Policy Server

A policy entry specifies one or more permissions for code from a particular code source - code from a particular location (URL), code signed by a particular entity, or both. The CodeBase and the SignedBy text boxes are used to specify which code user want to grant the permission(s) user will be adding [11].

a) CodeBase value indicates the code source location; you grant the permission(s)to code from that location. An empty CodeBase entry signifies "any code" .

b) A SignedBy value indicates the alias for a certificate stored in a keystore. The public key within that certificate is used to verify the digital signature on the code; user grants the permission(s) to code signed by the private key corresponding to the public key in the keystore entry specified by the alias. The SignedBy entry is optional; omitting it signifies "any signer" -- it doesn't matter whether the code is signed or by whom.

If you have both a CodeBase and a SignedBy entry, the permission(s) will be granted only to code that is both from the specified location and signed by the named alias. After that corresponding permissions can be added to the code and saved. The

corresponding entry is save in the policy file and can be viewed on editor .The URL of file is "java.home\lib\security\java.policy "

5.2 Agent

The agent system for secure mobile agent will consist of agents for DES – symmetric algorithm, RSA – MD5 digital signature, DSA digital signature and digital certificate maintenance agent.

5.2.1 DES Agent

This agent will be able to transfer the encrypted data from source to destination using DES – FIPS symmetric algorithm as explained in chapter 3. The agent will transfer the encrypted data and the receiver will use the key to decrypt it. The code is written in java. For this to be possible, java.security file, aglet. policy file has to be used to allocate permissions to read and right. The corresponding entry has to made in the policy tool file also. The keystore entry has also to be registered.

The source transmits the encrypted data with the key and assigns the corresponding permission in the policy file as explained earlier in this chapter. The encrypted data is the permission to only read and not write and so that the receiver cannot modify the text.

5.2.2 RSA Agent

This agent will be able to transfer the data along with the signatures from source to destination using RSA – MD5 asymmetric algorithm as explained in chapter 3. The agent will transfer the data and signature signed by the public key so that only the receiver having the private key will be able to decrypt it. The code is written in java. For this to be possible, java. security file, aglet. policy file has to be used to modified and allocated permissions to read and right as explained earlier in the chapter. The corresponding entry has to made in the policy tool file also.

5.2.3 DSA Agent

This agent will be able to transfer the encrypted data from source to destination using DSA asymmetric algorithm as explained in chapter 3. The agent will transfer the data and the signature. This signature is stored in the keystore entry and so the corresponding entry is referred at the receiver side. Receiver will use the keystore password to decrypt it. The code is written in java. For this to be possible, java. security

file , aglet. policy file has to be used to allocate permissions to read and right. The corresponding entry has to made in the policy tool file also. The keystore entry has also to be registered.

The source transmits the data with the signature and assigns the corresponding permission in the policy file as explained earlier in this chapter. The encrypted data is the permission to only read and not write and so that the receiver cannot modify the text.

5.2.4 Digital Certificate Agent

This agent deals with the maintenance of digital certificates stored in the keystore files at the time of starting the Tahiti server. With the help of this agent, one can view the certificate of that particular entry, view the private and public keys of that particular user's keystore entry, view the corresponding users stored in that keystore file, send a particular user entry as the trusted key entry in the receiving side 's keystore and receive a user 's entry as the trusted key entry in the corresponding keystore..

A keystore file with multiple users entries with their names and corresponding details is created and then added to the Tahiti Server on start up.

RESULTS AND DISCUSSION

6.1 User Interface of Initial Login to Tahiti Server

The figure 6.1.1 shows the initial login procedure to Tahiti server. The user name and password given by the user is verified from the keystore file. This keystore file is separately created to facilitate multiple users on the same Tahiti server.

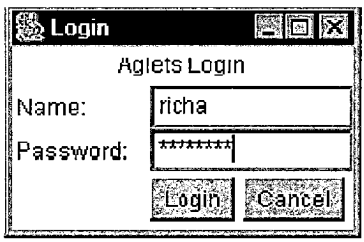


Figure 6.1.1: Initial Login Process for Tahiti Server

6.2 User Interface of DES Agent

The figure 6.2.1 shows the user interface of DES agent. The sender enters the data in the text area and then presses the encrypt button. After that, in the text field below, the encrypted message is shown and then the agent is ready to dispatch .If the folder des does not exists in the sender directory, it is created and then the corresponding files are stores there. Then at receiving side, the receiver decrypts the data as shown in figure 6.2.2 and gets the original message if the cipher file and key file is not tampered. Else it shows error message on the console.

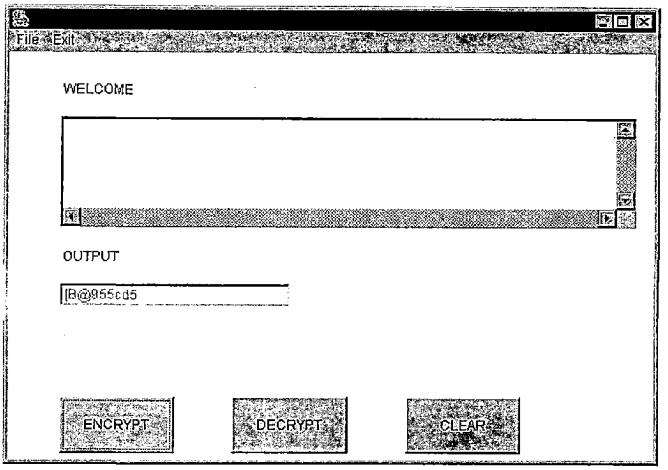


Figure 6.2.1:User Interface of DES Agent (Sender Encryption)

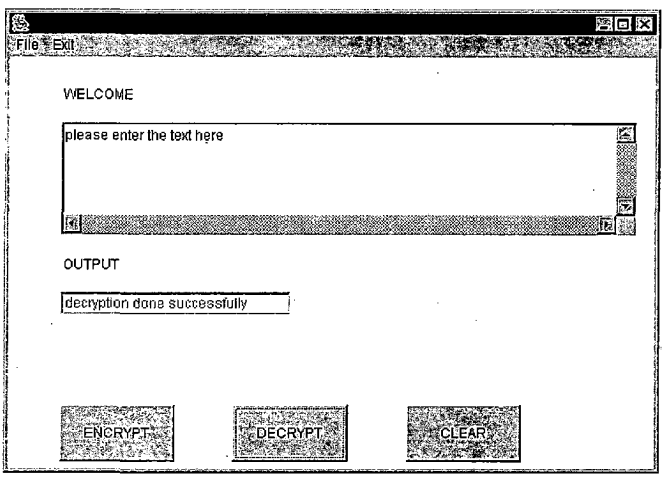


Figure 6.2.2:User Interface of DES Agent (Receiver Decryption)

6.3 User Interface of RSA Agent

The figure 6.3.1 shows the user interface of DES Agent. The sender enters the data in the text area and then presses the encrypt- rsa button. After that, in the text field below, the encrypted signature is shown and then the agent is ready to dispatch .If the folder rsa does not exists in the sender directory , it is created and then the corresponding files are stores there. Then at receiving side, the receiver decrypts the data using the private key as shown in figure 6.3.2 and gets the original message if the signature file and key file is correct. Else it shows error message on the console.

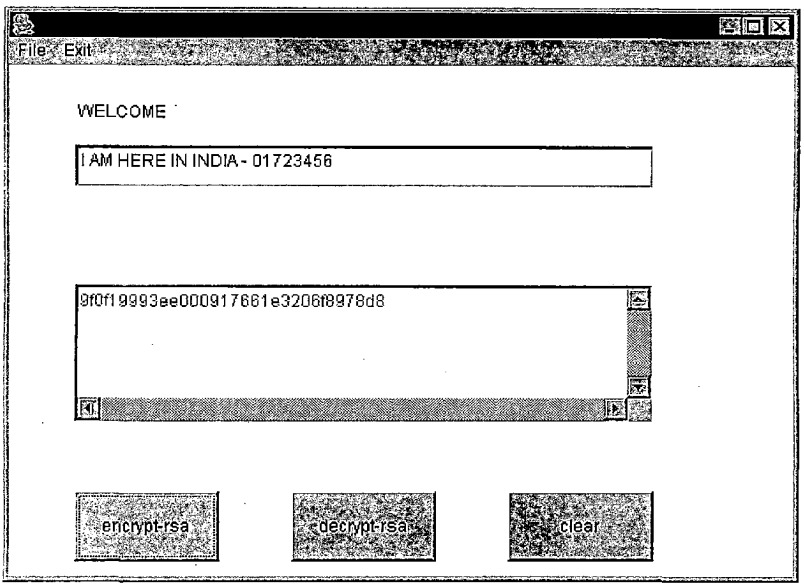


Figure 6.3.1: User Interface of RSA Agent (Sender Encryption)

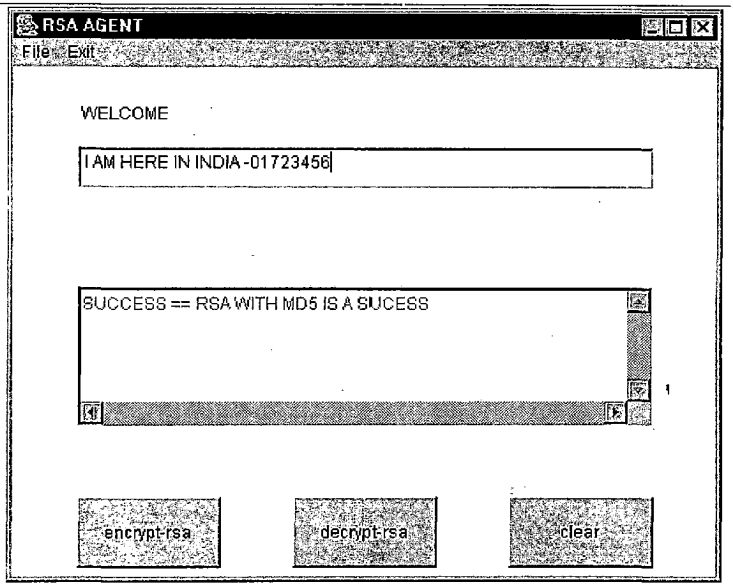


Figure 6.3.2: User Interface of RSA Agent (Receiver Decryption)

6.4 User Interface of DSA Agent

The figure 6.4.1 shows the user interface of DSA Agent. The sender enters the data in the text area, enters the alias name and password and then presses the encrypt button. After that in the text field below, the signature is shown and then the agent is ready to dispatch .If the folder dsa does not exists in the sender directory, it is created and then the corresponding files are stored there. Then at receiving side, the receiver decrypts the data as shown in figure 6.4.2 and gets the original message if the signature file is not tampered and keystore entry of receiver has the entry of that corresponding user. Else it shows error message on the console.

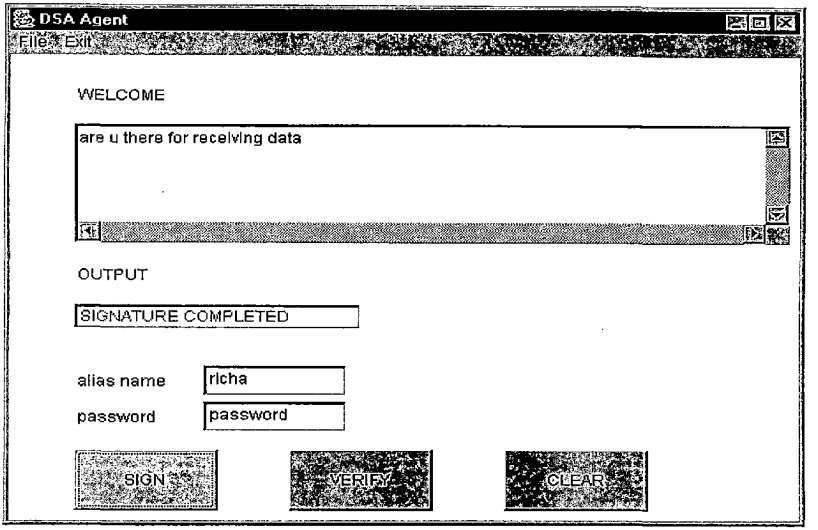


Figure 6.4.1:User Interface of DSA Agent , Sender Encryption

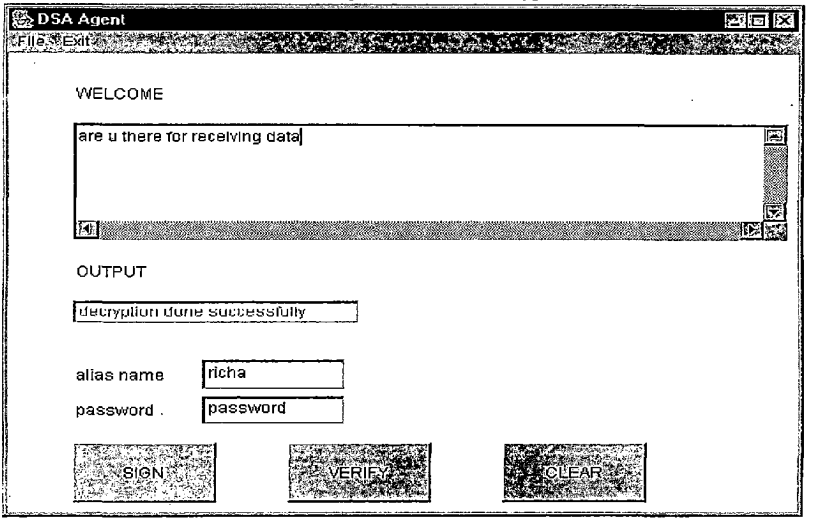


Figure 6.4.2 User Interface Of DSA Agent, Receiver Decryption

6.5 User Interface Of Digital Certificate Agent

The figure 6.5.1 shows the user interface of Digital Certificate Agent. The user enters the alias name and password in the respective field and then presses the desired button .The button print is to view the certificate of that user in the keystore file .The button users is to view all the (trusted entry users) users in that corresponding keystore file as shown in figure 6.5.2. The button key is to view all the keys of that particular alias in the keystore. The button send is to send the entry of one user to the receiver side. The button receiver is to receive the entry of that user to the receiver side keystore.

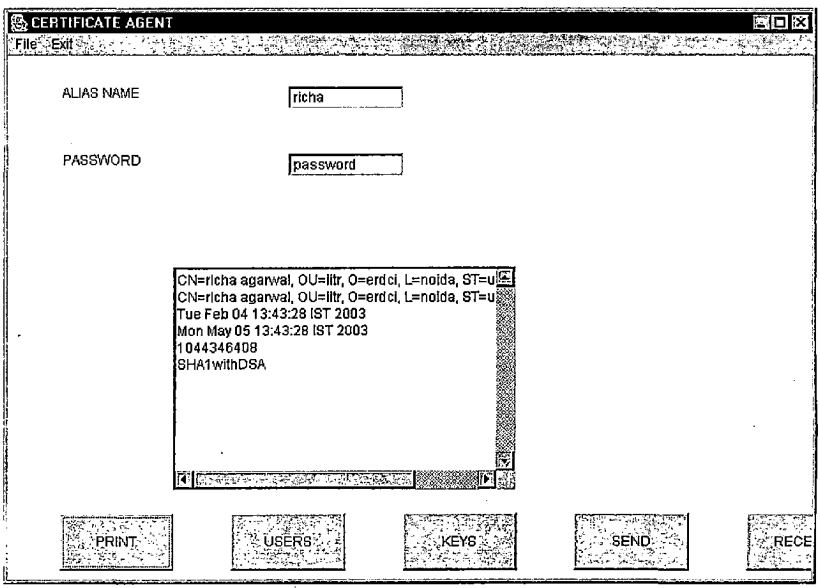


Figure 6.5.1: User Interface Of DIGITAL CERTIFICATE Agent

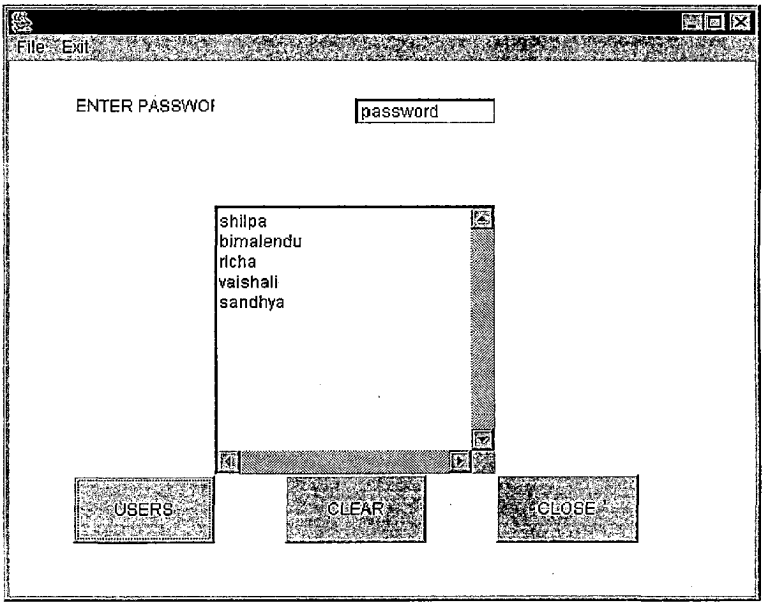


Figure 6.5.2: User Interface Of DIGITAL CERTIFICATE Agent showing all the users in that corresponding keystore file

Chapter 7

CONCLUSION

Agents were designed , implemented and tested for all the fundamental operations for aglets.

The design ,implementation and testing of following agents was done successfully

1) Data Encryption Standard Agent - symmetric key algorithm which authenticates the receiver to the sender.

2) RSA Agent – a code signing agent that uses RSA with MD5 as the asymmetric digital signature (1024 bit key size of encryption and decryption key).

3) DSA agent – a agent that generates digital signature and verifies as per the digital signature standard by taking key values from the keystore file.

4) Digital Certificate agent - a agent that lets the user know about other users present in that keystore file, the respective keys stored in the keystore file, the certificates associated with each entry in the keystore file and transfer of an trusted key entry from sender to receiver.

The limitation of this work is that it is the Aglet Tahiti Server has to be installed on the source and destination machines for the agent to be in functional mode.

The future work can be to implement distributed digital certificates using JNDI (java network distributed interface) and LDAP (light directory access protocol). To implement this, multiple servers must be used to install Tahiti servers and then implement distributed digital certificates.

REFERENCES

[1] Sharon A. Wheeler,"A Survey of Design Issues In Developing Trusted Mobile Agents", July 29, 2001.

[2] Lange, d., B., And M. Oshima, "Seven Good Reasons for Mobile Agents Communication of the ACM", March 1999.

[3] G. KarJoth, D.B. Lange and M. Oshima, "A Security Model for Aglets", IEEE Internet Computing, pages 68-77, July 1997.

[4] National Institute of Standards and Technology, Special Publication 800- 19, Mobile Agent Security, Wayne Jansen , Tom Karygianms.

[5] William Stallings, Cryptography and Network Security, Prentice Hall of India, Second Edition , 2000.

[6] W. M. Farmer, J. D. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements, MITRE sponsored research program, 1996.

[7] D. B. Lange, M. Oshima, Programming and Deploying Java™ Mobile Agents with Aglets, Addison-Wesley, August 1998.

[8] A. Bieszczad, B.Pagurek, and T.White, Mobile Agents for Network Management, IEEE Communications Surveys, September 1998.

[9] B. Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, ISBN:0471117099, 1996

[10] Wayne A. Jansen, Determining Privileges of Mobile Agents, National Institute of Standards and Technology, 2000.

[11] Java tutorials available at " www.java.sun.com/tutorials"

[12] Aglets framework available at " www.trl.ibm.co.jp/aglets "

Tahiti Menu Structure

Tahiti provides the functions for handling agents and for controlling the server, which can access from the following menu items.

A.1 Aglet: for handling aglets

Create...	Create an Aglet.
Dialog...	Sends a request to an aglet to open its dialog panel.
Dispose...	Destroys the agent.
Clone...	Make a copy of the agent.
AgletInfo	Shows the properties of the agent.
Kill	Destroys the agent. Aglet does not call onDisposing().
Exit	Shutdown the server.

A.2 Mobility

Dispatch	Send the aglet to another server.
Retract	Retract a dispatched aglet from another server.
Deactivate	Deactivate the aglet with time.
Activate	Activate a deactivated aglet.

A.3 View

Memory Usage	Show the memory usage amount
Log	Show the logged records of agents behavior on this server.

A.4 Options

General Preference	Font, Startup Aglet, Cache.
Network Preference	Set parameters for Http Tunneling, Authentication, etc.
Security Preference	Set access privileges for aglets. Specify File System, Network Access, and others.
Server Preference	Server Setting.

A.5 Tools

Invoke GC	Start garbage collection.
Threads	Display the current thread information on the console.

A.6 Help

About Tahiti...	Information about this aglet viewer.
About Aglets...	Information about the aglet library.
Release Notes	Open release notes.
Aglets Home Page	Open Aglets page on the WWW
Feedback	Open Aglets feedback page on the WWW
Bug Report	Open Aglets bug report page on the WWW
Frequently Asked Questions	Open Aglets FAQ page on the WWW

A.7 Customization with dialog interface

You need to specify some parameters for your aglet server. Setting with the default values does not provide the full functionality of this server. By tuning up Tahiti with those parameters, your aglet server works effectively.

A.7.1 General Preferences

Font: Defines the size of the font in Tahiti window. The change of the setting becomes effective immediately.

List View: Defines the order of the Aglet items in the list box of the Tahiti window. The change of the setting will be reflected after clicking one of the items on the list box.

Startup: You can specify an aglet that automatically starts up when Tahiti is started. (The Launch Startup Aglet checkbox enables this function.)

Cache Control: Clearing up the class cache.

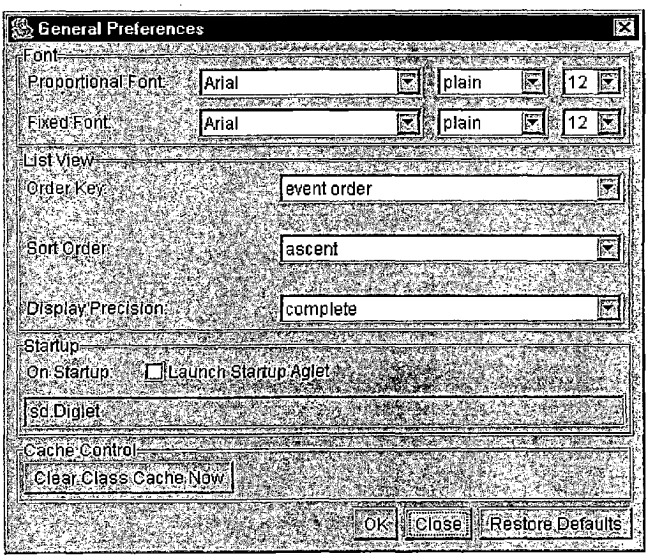


Fig.A.7.1: General Preferences

A.7.2 Network Preferences

Http Tunneling: In case you are protected by a firewall, you can specify a http proxy server to access the information at the outside of the firewall. We call this method as Http Tunneling. Check "Use HTTP Proxy" and Specify your http proxy information like a setting in a world wide web browser.

Authentication: This is a switch for security options. When you enable Authentication, Tahiti have to keep at least one secret file. Tahiti can communicate with each other only if the other aglet server has (can access) the same secret file.

Accept HTTP Request as a Message: We can create an aglet, which has a URL and returns html. When we enable this option, aglet receives HTTP request as a message.

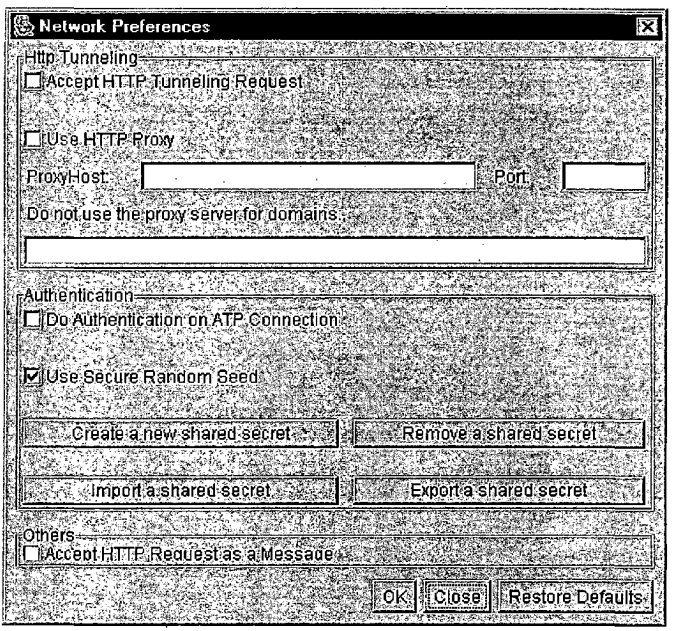


Fig.A.7.2: Network Preferences

A.7.3 Security Preferences

We have a dialog box for specifying security. User can specify the access privilege for the following items.

- FileSystem: Restricts the access to the files. "File/Directory" field is defined by absolute path name. Possible "Actions" are read, write, and execute.
- Socket
- Window
- Property
- Runtime
- Security
- All
- Aglet
- Message
- Context
- Protection (Aglet)
- Protection (Message)

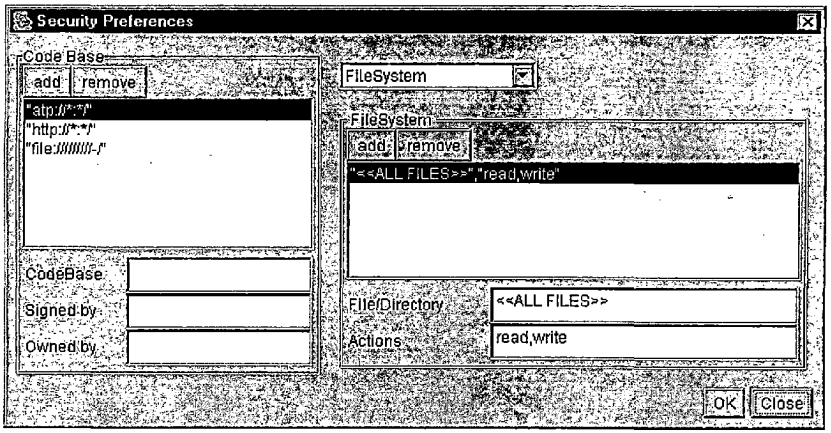


Fig.A.7.3: Security Preferences

A.7.4 Server Preferences

User can assign a public path name to a directory as an alias.

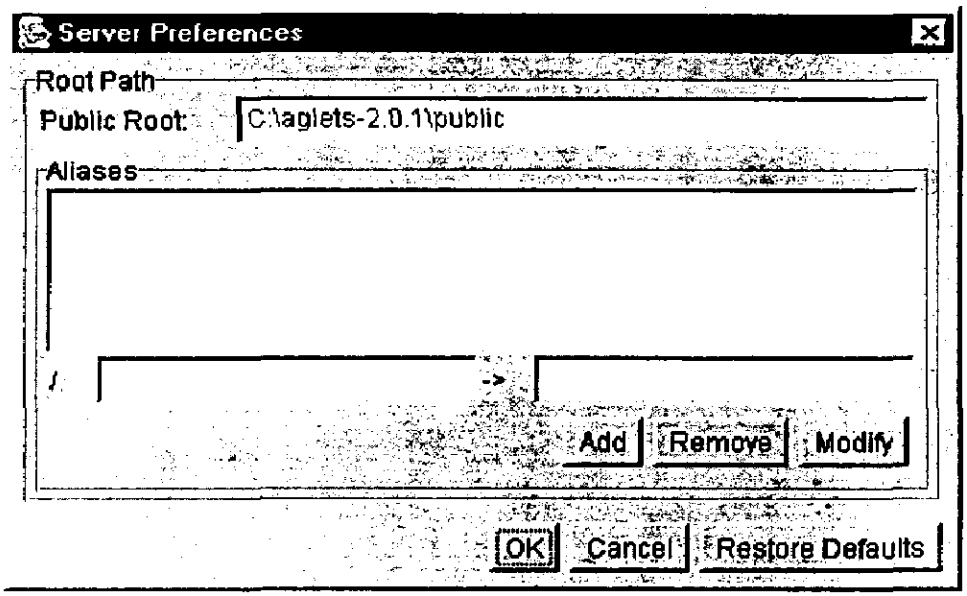


Fig.A.7.4: Server Preferences