SIGNING AND VALIDATING CONTRACTS IN OPEN COMPUTATION EXCHANGE AND ARBITRATION NETWORK (OCEAN)

By

SAHIB SINGH WADHWA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

To my Parents

ACKNOWLEDGMENTS

First and foremost, I appreciate Dr. Michael Frank's consistent advice, encouragement and insightful critiques during the course of my research. I thank him for being my advisor and giving me an opportunity to work on his fascinating research project.

I would like to express sincere thanks to Dr. Sartaj Sahni and Dr. Janise McNair for serving on my supervisory committee and for their meticulous review of my thesis.

I would also like to express my gratitude to all members, past and present, of the OCEAN research team for their painstaking collaboration and contributions.

I thank Dr. Pierce Jones for providing me an opportunity to continue my education by involving me in his research on energy efficient homes as a graduate assistant at Florida Energy Extension at the University of Florida.

I thank my uncle Dr. Atma Singh Wadhwa for encouraging me to come to the USA to continue education and helping me morally and financially.

I would especially like to thank my mother, Sardarni Surinder Kaur Wadhwa, my father, Dr. Devinder Singh Wadhwa, my elder sister, Dr. Amanjot Kaur, and my little sister, Manpreet Kaur, for their guidance, love and support.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

SIGNING AND VALIDATING CONTRACTS IN
OPEN COMPUTATION EXCHANGE AND ARBITRATION NETWORK (OCEAN)

By

Sahib Singh Wadhwa

May 2003

Chair: Michael P. Frank
Major Department: Computer and Information Science and Engineering

The strength of an e-business application relies to a large extent on the underlying architecture of its security protocols. Trade proposals, negotiations, contracts, etc. involve trust. When important documents like these are based on ASCII text, it is hard to believe that they will not fall into the hands of malicious users or be subject to forgery.

Existing transport layer protocols such as SSL and TLS guarantee point-to-point security over the communication channel. They ensure data integrity, confidentiality and authenticity. But these are not sufficient in a peer-to-peer computing environment, where application layer attacks become dominant. Continued security of data after it has been delivered is very important.

The primary reason for using XML digital signatures in my research for securing transactions, which otherwise can be handled by SSL, is to ensure permanent non-repudiability and impossibility of forgery. XML signatures ensure that application layer attacks such as repudiation of authorship can be prevented. A secondary reason for using

XML signatures is to enable the signing of selected elements of a contract (an XML document).

This thesis presents an XML Signature Module (XSM) that works in conjunction with the matching, negotiation and accounting components of the OCEAN system. Being part of the security component, XSM is responsible for signing and validating contracts. The thesis also presents the design and implementation of two protocols. The Registration protocol has been modified from its existing design. Signing and Validating protocol, which is the primary focus of this thesis, has been developed.

The implementation of XSM is carried out in accordance with the recommendations passed by the joint proceedings of a W3C and IETF working group. The XML Signature Module is implemented using J2SE 1.4.1.

CHAPTER 1
INTRODUCTION

In a scenario where a large number of computers are connected in a peer-to-peer e-commerce network, if the contractual documents being exchanged are merely text based, there is a significant risk of security violations. The intermediate nodes might intercept and alter the message in transition. Not only this, there can be serious legal issues when one of the two parties involved in business refuses to take responsibility which otherwise it should as per the contract agreement. This type of disagreement is called repudiation of authorship, which needs to be prevented. According to the American Bar Association [1], non-repudiation is defined as, "Strong and substantial evidence of the identity of the signer of a message and of message integrity, sufficient to prevent a party from successfully denying the origin, submission or delivery of the message and the integrity of its contents." Non-repudiability is a hot topic being addressed by many ongoing projects in the computer industry these days such as OpenEvidence[1], ESSL[2].

1.1 The Problem

In OCEAN, messages such as search request from the buyer, matching responses from the sellers, negotiation between buyers and sellers, contractual agreement between a buyer and a seller, etc., all use XML. The idea of making contracts in XML was to harness the richness and benefits of XML, but, being based on ASCII text, XML is vulnerable to attacks.

Secure Sockets Layer (SSL) is a transport layer protocol. It provides a secure channel for communication of sensitive data. It solves most of the security problems such as confidentiality, integrity, authentication, etc. But one of the main problems, non-repudiability, still remains un-addressed. Once received, there is no way to prove to a third party the identity of the original sender of the data. Protecting data after transmission is the main reason for the increasing attention of security technologies on ensuring non-repudiation. SSL provides point-to-point security by transmitting encrypted data across the communication channel. It does not preserve the authenticity of the original sender if there are multiple intermediate users forwarding that information over separate sessions. Figure 1-1 illustrates a scenario in which each link represents a separate SSL session. Data originates from node 1 and travels to node 2 and node 3. Node 3 forwards it to node 4 and node 4 forwards it to node 5. Once the data arrives at node 4, there is no way for node 3 to prove to node 4 that the forwarded data originated from node 1. Therefore, in a situation where there is a dispute about the authenticity of the data, it would be impossible to prove which of the intermediate nodes modified the data.

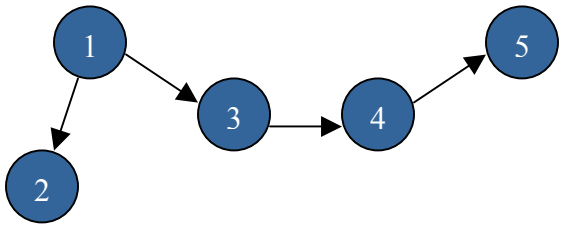

Figure 1-1 Message propagation

Secondly consider a situation where the buyer and the seller reach an agreement. The buyer migrates the job to the seller's machine for execution. In the middle of the execution, the seller refuses to complete the task. Or in another situation, if at the end of

completed execution, the seller refuses to send the results of execution to the buyer. It is also possible that the seller denies to have made any agreement in the first place. What could possibly be a solution in a scenario like this?

1.2 The Proposed Solution

Being an application layer security issue, non-repudiability cannot be addressed by SSL. In fact, SSL only protects the data during transit, where its confidentiality and integrity is, relatively speaking, less likely to be attacked [26].

Digital signatures have been very successful in solving non-repudiation problems. When the documents of concern are XML documents, then conventional methods of signing and encrypting are not sufficient either. Conventional signature methods would generate different signature values for two textually dissimilar XML documents even though they are semantically identical. W3C and IETF have jointly specified the standards [31] for signing XML documents. XML signatures provide application level security. Another benefit of using XML signatures is observed when we need to disclose or sign partial information unlike encrypting an entire document in conventional encryption and signing techniques.

XML signatures provide data integrity (verification that data has not been modified during transit), message authentication (verification of correct message and its checksum), and/or signer authentication (verification of signer's identity) services for data of any type, whether located within the XML that includes the signature or elsewhere [33].

In OCEAN, a client registers with the CAS (Central Accounting Server) over SSL connection as shown in Figure 1-2. It therefore does not need to provide explicit encryption because SSL already provides it. Here we can trust the CAS that it will not

deny services or forge information because this is one of the primary trusted servers in the OCEAN system.

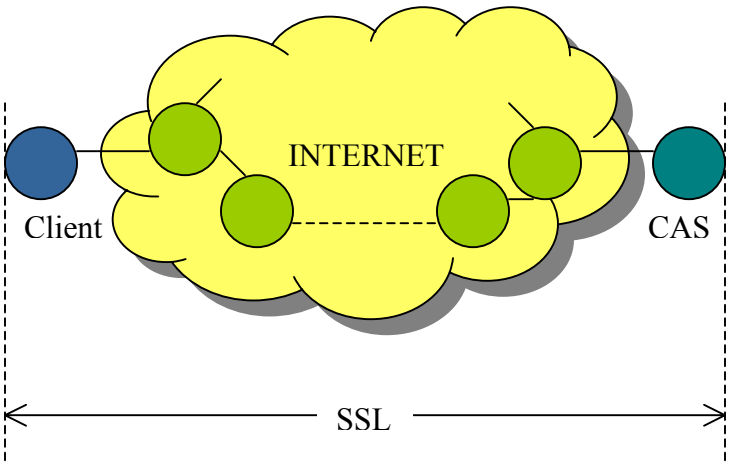

Figure 1-2 Client registers with the CAS over SSL connection.

In order to ensure non-repudiability, every contract is signed by the participating parties. Every party that signs attaches its public-key certificate to the contract for ease of verification. A public-key certificate is obtained from the Certification Authority, which works in conjunction with the CAS. Once a contract is signed, it is nearly impossible for any party to disavow its signature, alter the contract or forge any other party's signature. A key advantage of using XML signatures for signing contracts is that signatures from different parties can simply be added as additional elements in the document, unlike conventional signatures, which are nested within one another. Also, the signer can choose to sign specific elements in the contract if the need be so.

One of the challenges here is to guarantee that a digitally signed document has the same legal status as a paper document with a handwritten signature. The American Bar Association[3] (ABA) has defined the guidelines to help assess and facilitate interoperable trustworthy public-key infrastructures in *PKI Assessment Guidelines* [2]. For instance, "A copy of a digitally signed message is as effective, valid, and enforceable as the original of

the message." *PKI Assessment Guidelines* [2] is a sequel to the *Digital Signature Guidelines* [1] first released by ABA in 1996. It is recommended for the participating parties as well as the programmers to be aware of such guidelines before dealing with any e-business/e-commerce system.

This thesis presents a prototype of an XML Signature Module (XSM) that enables a buyer to sign a contract and the seller to validate the signed contract before co-signing it. XSM has been added to the existing modules in the matching layer of the OCEAN Node architecture.

1.3 Organization of the Thesis

The organization of this thesis is discussed below.

- Chapter 2 provides a survey of the literature reviewed during my research.
- Chapter 3 provides an overview of the OCEAN architecture and explains the position of the XML Signature module in the overall OCEAN architecture.
- Chapter 4 gives an overview of Certification Authority and format of a certificate.
- Chapter 5 gives an introduction to the XML signatures. The XML signature schema and types of XML signatures are explained.
- Chapter 6 gives an explanation of the contract schema used. It also describes the Signature element in detail. The Signature element is the central focus in my thesis.
- Chapter 7 explains the architecture of XML signature module in detail. The design of Registration protocol and Signing and Validating protocol is explained.
- Chapter 8 provides details of the API developed to build XML Signature Module
- Chapter 9 summarizes the conclusions and future work in the direction of extended security.
- A superscript on a word indicates that more information about it can be found at the corresponding URL listed in Appendix D.

CHAPTER 2
LITERATURE REVIEW

The area of signing XML documents is relatively new (the W3C recommendation on XML Signature Syntax and processing was passed on February 12, 2002). Some research has been done in this area but most of the products that were developed are either incomplete or still under development. Papers and thesis written either before or after the recommendation provide a good understanding of the mechanism of signing XML documents.

The American Bar Association defines the guidelines to help assess and facilitate interoperable trustworthy public-key infrastructures in the *PKI Assessment Guidelines* [2] while it defines the digital signature guidelines in the *Digital Signature Guidelines* [1].

Han et al. [15] provide an XML digital signature based design for e-business applications. Miyazawa and Kushida [21] propose an advanced Internet XML/EDI model that allows an Application Service Provider (ASP) to provide services for purchase/shipping of products over the Internet. Takase and Uramoto [29] provide another XML digital signature based system. They propose the concept of a signature proxy that is capable of listening to the XML messages exchanged on the network. Karlinger [16] provides an elaborate implementation of XML digital signatures in Java. Scheibelhofer [25] also provides a design for signing XML documents.

Mactaggart [18] and Simon [26] have given nice introductions to XML encryption and XML digital signatures respectively. Brown [5] goes into the details of legal issues involved in using digital signatures.

Research in the area of signing and validation of crucial documents like contracts, not necessarily XML based, has been carried out for many years. Ben-Or et al. [3] present a fair protocol for signing contracts.

Prior to this thesis, the researchers in the OCEAN group at University of Florida have contributed in the other modules of the project. Tobias [30] provides resource and requirement schemas written in XML. Park [23] gives a partial implementation of the PLUM component. The PLUM component has since been significantly improved with a network evolution technique on the top of it.

Nitin [8] presents a registration and authentication protocol. While reworking on the registration protocol, I have eliminated certain complexities such as encryption of the message using public-keys. In order to compensate for this we are using SSL as a transport mechanism. SSL takes care of encryption implicitly. Secondly I have split the Central Server into two servers. The Central Accounting Server manages user accounts and a log of transactions while the Certification Authority maintains a list of public-key certificates. The CA can also provide its services to non-OCEAN nodes in the Internet.

Chakravarthula [7] provides a trading system design and implementation. In his thesis, Chakravarthula [7] provides a Matching Component that generates a match request, which travels across the peers up to a given hop length, returning a list of suitable matches. This list of matches is then used by the Negotiation component for negotiation.

Chokkareddy [9] describes an automated negotiations approach towards a computational market. He provided a “Contract Specification language”. The schema of

contracts is used in this thesis to sign using XML digital signature standards. He also explains two protocols for negotiation.

Both Chakravarthula [7] and Chokkareddy [9] have used XML messages in their study. Therefore I have presented a signing and validation protocol which can be integrated with the OCEAN system to provide security. Presently we are focusing on the contracts, but in future even the match requests [7] can be signed if it is deemed necessary.

In addition, the following RFC's were referred during the research:

- RFC 2807, July 2000, "XML Signature Requirements"
- RFC 3075, March 2001, "XML-Signature Syntax &Processing"
- RFC 3076, March 2001, "Canonical XML Version 1.0"
- RFC 3275, March 2002, "XML-Signature Syntax & Processing"

CHAPTER 3
THE OCEAN SYSTEM

3.1 Overview

There are currently millions of people and organizations whose computing resources are connected via the Internet. It has been observed by numerous sources that these resources are frequently idle [6,13]. Many computers that are online are not involved in any compute-intensive tasks. The overall available computing power is underutilized. The computing world can potentially be revolutionized if systems can transparently buy, sell and use remote computing resources via the Internet.

OCEAN (Open Computation Exchange and Arbitration Network) is a major ongoing project at the CISE (Computer and Information Science and Engineering) department of University of Florida. It aims at developing a fully functional market-based infrastructure supporting automated, commercial buying and selling of computing resources (CPU cycles, memory, disk space, network bandwidth, other specialized remote resources of interest, etc.) via distributed applications and mobile agents, across the Internet.

OCEAN is a layered system that exposes its different layers as web services accessible via SOAP[4]. Within a single node we have a matching service, negotiation service, security service and mobility service [14].

Two important requirements of OCEAN are scalability and interoperability. Scalability is achieved by implementing OCEAN as a distributed, peer-to-peer based architecture. At present, portability is achieved by using OCEAN on the Java

programming language. A parallel implementation for .NET is also being done in the C# programming language. Any node can perform more than one task at a time, e.g. act as a matcher, execute a buyer's computation, serve as an application development environment, or as an application launch pad. In the present version of OCEAN, buyers are active and sellers are passive, in the sense that buyers initiate the deal and sellers will be waiting for incoming purchase offers.

If an OCEAN user wants to deploy a job that requires extra resources, a match request describing the user's requirements is formed and submitted. The matching system [7] will propagate the request to the node's peer nodes. These peer nodes will propagate it further until the maximum hop count has been reached or a time out is expired. During this process, the resource requirements are compared to resource descriptions and comparable matches are returned to the original sender. The negotiation system comes into the picture next, in order to settle the price between the buyer and the seller. If an agreement between the two parties is reached during negotiation, both the parties sign a contract. When both of the parties have validated each other's signatures, the code is migrated to the seller's machine for execution. The seller's machine executes the code. However, if the parties were not able to reach an agreement during negotiation, the next best seller is considered for negotiation and the process continues. For the current implementation we are using the "YesNo" protocol for negotiation [9], that is to say, if an exact match is found, it is taken as an agreement otherwise the next best seller is considered.

In the present implementation of OCEAN, there is one exception to the proposed completely distributed nature of the OCEAN system, namely, the Central Accounting

System. The Central Accounting System consists of Central Accounting Server (CAS) and the Certification Authority (CA). Both of these are very essential since every user needs to register and get a public key certificate before it can actually make any transaction. In future implementations, we plan to remove this bottleneck by employing several such systems, one for each region. The result of doing so will lead to a formation of hierarchy of servers rooted by a master Central Accounting System. The servers will however need to talk to each other periodically to keep the information consistent.

Figure 3-1 gives a high-level network diagram of the OCEAN framework in action. There are a large number of OCEAN *nodes* on the network, connected by peering arrangements in arbitrary fashion (cloud), of which two nodes have been picked out belonging to an individual buyer and seller of some resource. The buyer locates a compatible seller using just the peer-to-peer network, during this process placing no load on the central accounting server (CAS). Then, it begins a direct one-on-one negotiation with the seller. Only after narrowing down the sellers and negotiating with one to produce a contract already digitally signed by both parties, is the CAS then involved, for the sole purpose of payment processing. The way the payment processing is done depends on the details of the agreed-upon contract. One possibility is that the buyer first ponies up the cash for the transaction, which the CAS then holds in an escrow account that is associated with the specific contract in question.

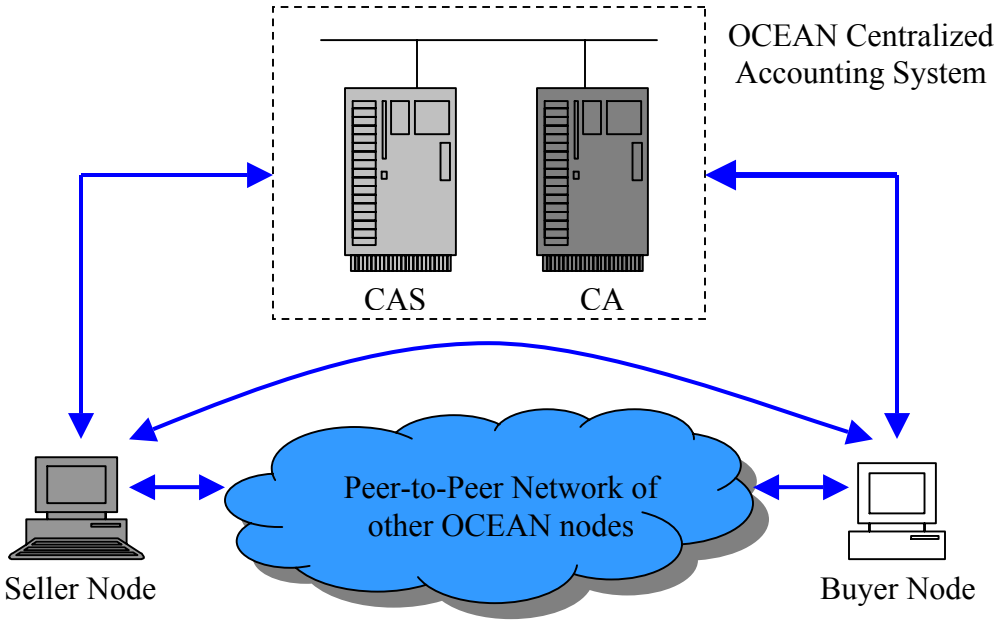

Figure 3-1 High-level network diagram of the OCEAN framework in action. Blue lines show communication pathways.

The seller can then verify that the money is available before delivering the goods. Once the buyer is satisfied that the goods have been delivered, he authorizes the CAS to release the funds to the seller. In case of a dispute, the CAS holds on to the disputed funds until the dispute is resolved.

3.2 Node Architecture

An OCEAN node has a layered architecture. The main components are shown in the Figure 3-2. In the figure, the interactions of the Security component with other components are highlighted. A brief description of each component is given below.

3.2.1 The Matching Component

Every OCEAN node contains a Matching component, but it is active on Seller nodes only. The Matching component at each seller node performs the matching process. It checks whether the search request that arrives from a buyer matches its resource description.

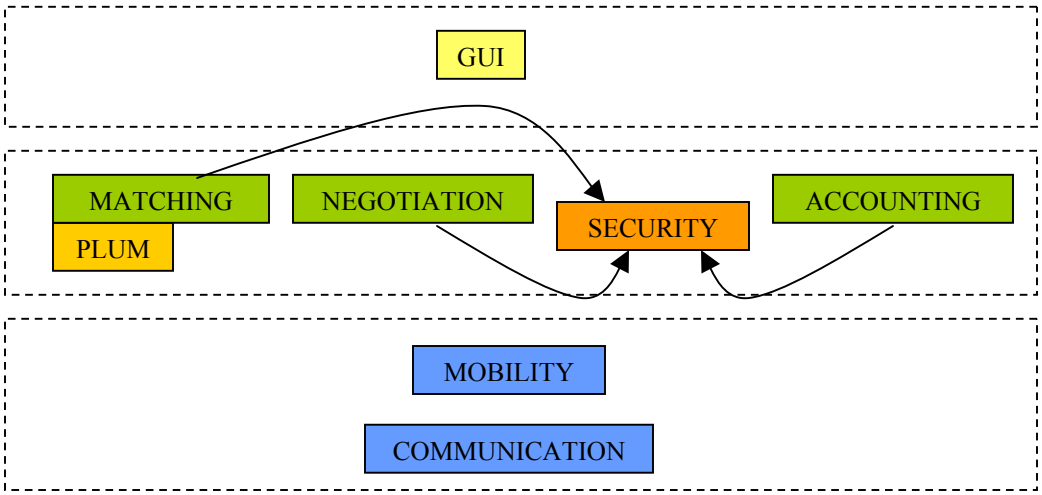

Figure 3-2 Node architecture. The Security component works in conjunction with Matching, Negotiation and Accounting components

3.2.2 Peer List Update Manager (PLUM)

The PLUM at each node maintains a list of addresses of its peer nodes in a priority queue. The priority is determined on the basis of a utility function. The PLUM periodically queries its peers and receives statistics about them, which it uses to update its peer-list.

3.2.3 The Mobility Component

The Mobility component is responsible for spawning and migrating the jobs in the Network. However, other Grid technologies can be used for this purpose as well. After the contract is signed, the Mobility component will spawn computing tasks on remote servers through the Security and Communication layers at each node and execute them.

3.2.4 The Negotiation Component

After the Matching component is finished matching comparable trading partners, it provides a list of sellers to the Negotiation component. The Negotiation component then chooses a seller from the list using predefined criteria and negotiates with that seller on several items of the deal. The negotiation component uses a rule-based system to

resolve conflicts during negotiation. Though in the initial release of OCEAN both matching and negotiation components are included as part of the whole system along with the other components, it is planned that in future the matching and negotiation services will be provided as web services which existing Grid architectures like Globus can make use of.

3.2.5 The Security Component

Security is one of the major issues discussed during the design of any system. It needs special attention in any distributed system in general and automated computational markets in particular. The Security component in OCEAN works in conjunction with the Matching, Negotiation and Accounting components. It comes into play every time a message is sent out or an incoming message needs to be validated. At present only the contracts are being signed and validated. But in the future versions even the matching request, and the negotiation transactions might be signed for greater security.

Security is the focus of this thesis. Several security issues have been identified in OCEAN. Buyer, seller, arbitrator, service provider, software provider or any outside party, etc., any of these can be a possible threat to the security of the system.

After an agreement has been reached between the buyer and the seller, both of them sign the contract one after the other. The public-key certificates of each party are appended with the signed contract for validation. Once signed, the contract is saved at each party's machine safely for use in future disputes and legal actions (if any). The next step is to move the code to be executed to the seller's machine, which the Mobility component takes care of. If in any case the seller refuses to execute the code or does not execute the code completely, the buyer can sue the seller in the court by presenting the signed contract or vice-versa.

3.3 Centralized Accounting System

OCEAN's Centralized Accounting system consists of the Central Accounting Server and the Certification Authority.

3.3.1 Central Accounting Server (CAS)

The CAS registers the users and works in conjunction with CA, which keeps track of the client's public-key certificate. In addition to this, it also logs transactions of the business that occurred on OCEAN, both buying and selling. All OCEAN users register with the CAS before they can join the OCEAN financial network. They can set up the CAS to accrue the money in their accounts for small transactions until the sum reaches some threshold and then transfer the money to external financial institutions like banks.

We are also considering an alternate architecture that would eliminate the CAS and allow the buyer to pay the seller directly.

3.3.2 Certification Authority (CA)

The OCEAN CA, like any other third party CA, is a trusted party, which issues and revokes certificates to the OCEAN users. It is thus very important to secure the CA physically as well as over the wire. It is the backbone of the trust on the basis of which buyers and sellers validate contracts. The CA is responsible for listening to the incoming CSR's (Certification Signing Request) and signing them. It is also responsible for maintaining a database of all the registered users along with their certificates. A request for any user's certificate has to be entertained. It also maintains a CRL (Certification Revocation List) of revoked certificates in order to prevent misuse.

3.4 Requirement Specification

The buyer needs to be able to identify the quality of the resources available at the seller's node. Govindaramanujam et al. [14] defines three techniques to determine the quality of the resource.

Resource Listing. The resource description is defined in an XML document. Based upon the description a quality rating is assigned. This approach takes limited overhead, but might not provide accurate or reliable information for every task.

Benchmark. The sellers' machine can be rated on the basis of several well-known benchmarks. The performance ratings for these benchmarks can be used as an indication of the quality of the resource. These benchmarks could be included in the OCEAN distribution.

Probe. This approach allows a buyer to run custom code on a potential sellers' machine to get an accurate assessment of the real-time capability of the resources offered. The probe is tuned to represent the task at hand and will be limited in the time it can execute. For e.g. a probe that renders only one frame of a movie scene. This approach is time expensive but gives most accurate indication of resource capability. It is very useful in ultra compute-intensive tasks.

3.5 Locating Resources

There can be three approaches [14] towards constructing a network for locating a suitable seller in the OCEAN network. These are:

Client-Server. A client-server type of architecture for OCEAN would need to employ a powerful server to listen to several client requests simultaneously. It offers a simple network where everyone is easy to locate. But as the number of users increases it

becomes a potential bottleneck. Secondly it is not scalable and a single owner incurs all the expenses.

Distributed Servers. A distributed network of market servers coordinating with each other can be employed in order to implement market mechanisms. This approach definitely improves the fault tolerance over the client-server approach but does not achieve maximum scalability. If the number of users increases dramatically in a short time, which is possible in the OCEAN system because of its nature, it might happen that we fall short of market servers to keep the market operating efficiently.

Peer-to-Peer. In a peer-to-peer based approach, a user application does not need to go through the central server every time it makes a transaction. However during initial registration it has to get itself registered with the CAS so that it can get a public key certificate. But this is a one-time thing. We have employed this approach in the OCEAN system. It is difficult to co-ordinate actions in pure peer-to-peer based systems, as they lack a central registry but their scalability and robustness outweigh this disadvantage.

3.6 Existing Grid Architectures and The OCEAN Grid

Like OCEAN, there are many projects that provide infrastructures for distributed computing. GriPhyN[5], Globus[6], Tryllian[7], Beowulf[8], SETI@Home[9], Distributed.Net[10], Gnutella[11], Sun ONE Grid Engine[12], Condor[13], Legion[14], and Ubero[15] are examples of distributed computing systems. They differ from OCEAN in a way that they lack a market mechanism (for e.g. compensating the resource providers). United Devices, Entropia, Parabon and Porivo are examples of ventures that do sell distributed computing resources, but they are closed markets. The markets in these systems are controlled by those companies themselves and often may not give the best possible compensation to the

resource provider. They also do not provide an open, standard API that any developer can target and test like OCEAN.

These systems are less flexible than OCEAN, since resources must be typically leased in advance, and cannot be purchased on-demand dynamically by arbitrary applications. They are also less scalable than OCEAN, because of their centralized architecture, unlike OCEAN, which uses a peer-to-peer architecture. OCEAN is being designed to be compatible with all the existing Grid technologies in the market that offer different solutions to certain basic problems, such as transfer of code to new hosts, and communication between distributed components of an application. Developers using any of the above-mentioned Grid technologies can access OCEAN to find trading partners, perform trading, and reimburse sellers, and then use their own Grid technology for task migration.

The Grid-neutral architecture of OCEAN is planned for development along with the Node architecture illustrated in Figure 3-2. The position of the OCEAN Grid is indicated in the Figure 3-3. Grid and mobile agent applications (which reside at the uppermost layer) can use OCEAN market services (available as web services) after which they can use any of the available Grid API's including the OCEAN Grid for executing the task suited for specific platforms. The main aim of the Grid-Neutral architecture of OCEAN is to develop OCEAN in such a way that the existing Grid architectures can make use of the market-based services of OCEAN with very little effort.

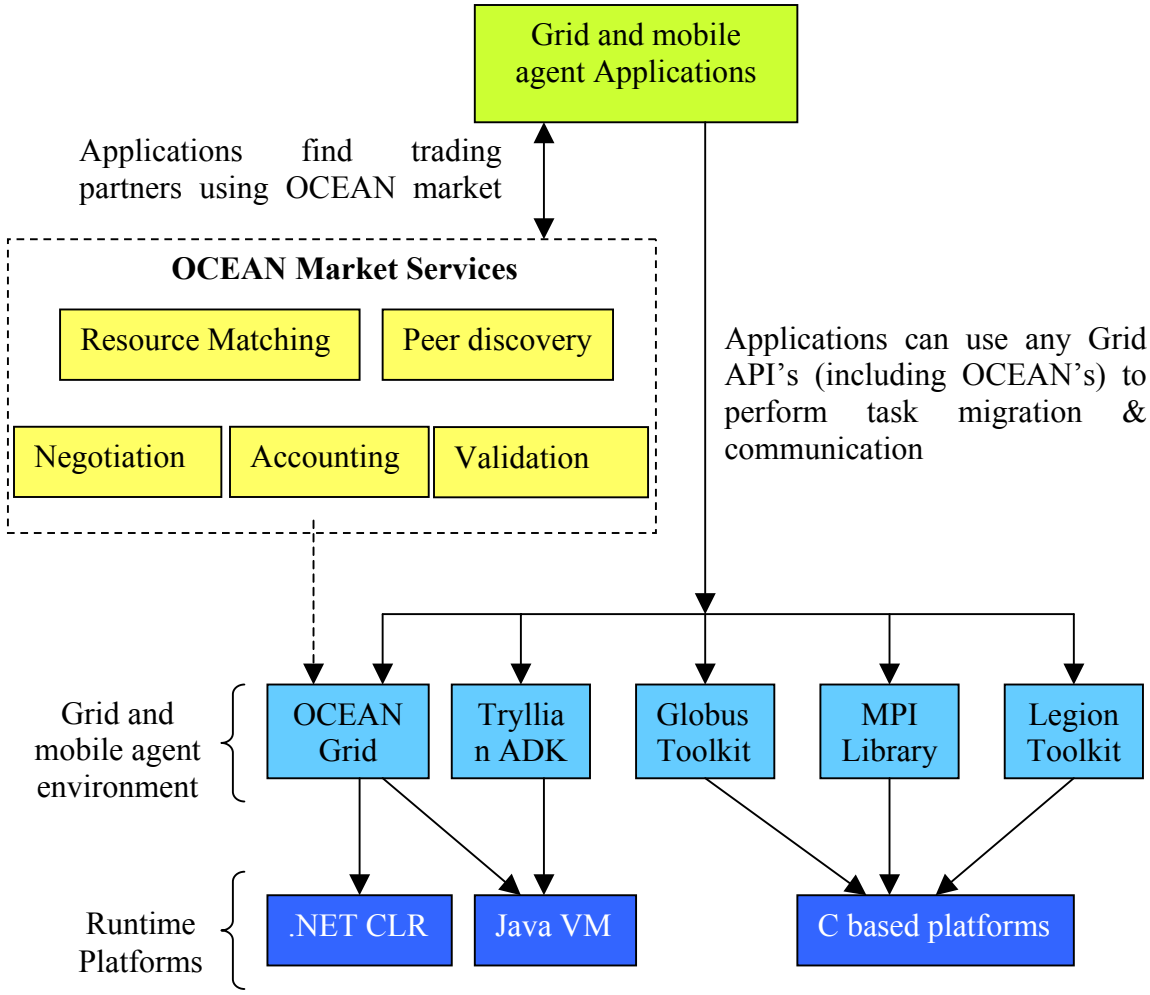

Figure 3-3 OCEAN Grid-neutral architecture.

3.7 Potential Applications

OCEAN is a very general-purpose service and we should not expect to be able to anticipate all the possible applications. According to Frank [12], some potential applications that might use OCEAN include:

Computational Science

- Theoretical physics & astrophysics: modeling quantum chromodynamics, materials, black holes, supernovae, galaxies, cosmology, etc., etc.
- Chemistry: Quantum chemistry, Molecular modeling

- Biology: Protein folding, Simulating biochemical reactions, genomics and proteomics, etc., Models in population biology.

Computational Engineering

- Stress analysis in structures, finite element models
- Aerodynamic fluid flow
- Electrical circuit simulations
- Analysis of geologic data (e.g. sonograms)

Computational Nanotechnology

- Nanostructured materials
- Nano-device design

AI Techniques

- Combinatorial optimization problems
- Genetic algorithms.
- Other machine learning algorithms
- Any search through a large model space

Computer Graphics

- Ray-tracing and radiosity-based rendering.
- Particle models of complex systems.
- Uses in motion picture, TV and advertising industries.

Miscellaneous

- Trend analysis for financial industry.
- Data mining for insurance companies.
- Solving scheduling problems for airlines.
- Distributed web testing – and many others.

The last group (except Distributed web testing) tends to be data-intensive & will require highly trusted providers that can store large amounts of data & provide numerous processors with high local bandwidth for a relatively long term.

CHAPTER 4
CERTIFICATION AUTHORITY

4.1 Definition

A Certification Authority (CA) is an entity that acts as a trusted third party organization or company guaranteeing that the identity of the individual (or organization) is who it claims to be. The CA enlists various means of verifying the identity of the individual (or organization) including calling them over phone. Once the verification of the identities of the parties in question is confirmed, CA will issue a unique digital certificate to that individual (or organization). A digital certificate serves to authenticate the identity of the owner as the CA has digitally signed it by using its own private key. The CA is also known as an issuer.

A Certification Authority needs to perform certain routine operations. This includes issuing new certificates, renewing old certificates, managing a Certificate Revocation List (CRL) for the certificates that have been revoked and maintain a database of active certificates in order to respond to the queries of the third parties.

In order to have a secure communication, communicating parties exchange certificates. They first obtain certificates from the Certification Authority. The certificate exchange mechanism is depicted in Figure 4-1.

The OCEAN CA works in conjunction with the Central Accounting Server (CAS). There are about 20 certification authorities that have been trusted so far so much that the browsers that are available these days come equipped with their root certificates. Some of the major CA's are:

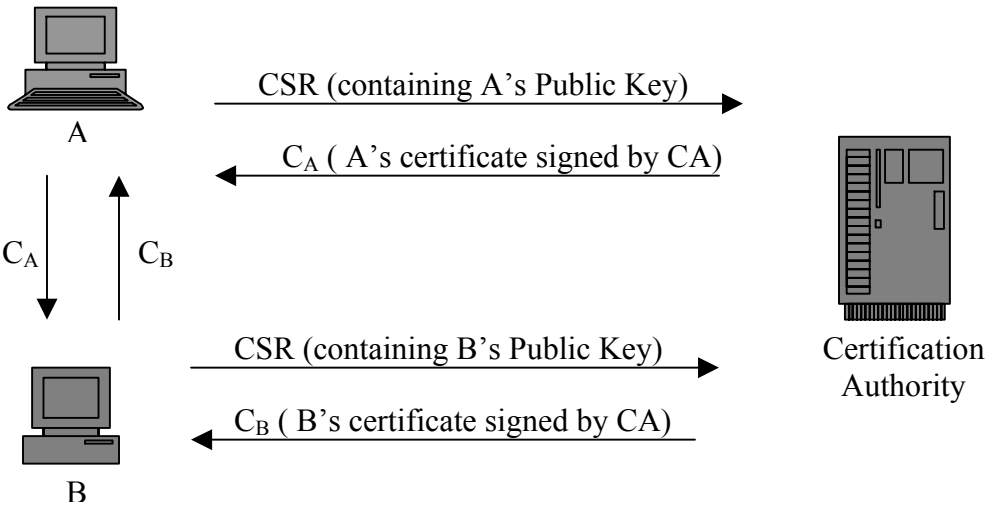

Figure 4-1 Certificate exchange mechanism

- Thawte, (www.thawte.com)
- Verisign, (www.verisign.com)
- Entrust, (www.entrust.com)
- Microsoft CA (www.microsoft.com)
- GTE Cyber Trust (www.cybertrust.gte.com)
- Secure Net (www.securenet.com.au)

In order to set up a CA for a Java based system, the following tools are required:

- `keytool` (usually comes with JDK[16]) for creating self-signed certificates on the client side and generating Certificate signing request (CSR).
- OpenSSL[17] for signing CSR's.

4.2 Certificates

A Certificate (or a digital certificate), also called Digital ID, is an electronic document that contains the public key of a user, together with some other information, rendered unforgeable by encipherment with the private key of the certification authority which issued it.

A digital certificate binds a public key to an individual (or organization). Digital Certificates are based on Public Key Cryptography (PKI), a scheme that uses public and private key pairs. The private key is known only to the owner and is used to create a digital signature. The public key is widely known and is used to verify the digital signature.

4.2.1 Root Certificate

A root certificate is the digital certificate of a certification authority. The public key in this certificate is used to verify the signature of the certification authority. With the corresponding private key the certification authority signs all certificates issued. The root certificate confirms that the public key and the certification authority are linked.

In case of OCEAN CA, the root certificate will be generated once and distributed along with the OCEAN software package.

4.2.2 Format of a certificate

The certificate contains all or some of the following

- Version
- Serial Number
- Signing Algorithm
- Issuer
- Valid From
- Valid To
- Subject
- Public Key
- Thumbprint Algorithm
- Thumbprint
- Key Usage

The "Issuer" and the "Subject" are the names written in a format called "Distinguished Name"[18] which consists of the following elements:

- CN-Common Name
- OU-Organizational Unit
- O-Organization
- L-Locality Name
- E-Email Address
- S-State
- C-Country

The most commonly used certificate format is X.509.It is a recommendation from CCITT. The version 3 of X.509 usually written as X.509v3 is the one that is currently being used. An X.500 directory is a service that stores X.509 Certificates and CRL's. It uses the X.520 naming convention (DN)

CHAPTER 5
XML SIGNATURES

Traditionally, documents of value were signed by hand. A handwritten signature, in most cases, is the name of the person written with his or her own hand. A handwritten signature is an acceptable standard till now because it is relatively difficult to reproduce or forge. A Digital signature however is different. Technically, a digital signature is a value generated upon the application of a private key to a message via a cryptographic algorithm. The essential characteristics of a digital signature are:

- Uniqueness: The application of a given private-key on a given data should always generate the same signature value.
- Verifiability: Anybody should be able to check the signature to see if it is valid.
- Un-forgeability: It should be impossible for anybody but the signer to attach its signature to the document.
- Non-reusability: It should be impossible to "lift" a signature off one document and attach it to another.
- Un-alterability: It should be impossible for anybody to change the document after it has been signed.
- Non-deniability (or non-repudiability): It should be impossible for the signer to disavow the signature once it is created.

5.1 Extensible Markup Language (XML)

Being based on ASCII text, XML [32] is platform independent. It has rich standards and a strong Java support. It is semantically rich and can be secured by the use of XML digital signatures.

5.2 XML digital Signatures

An XML signature is an XML document. More precisely, it is an element in an XML document composed of a set of standard tags arranged in a certain order. An example of an XML signature modified from W3C's website [33] is shown in Figure 5-1.

[s01-s50] The Signature element is composed of various elements in a certain order defined by the XML signature schema. This is the root element for any signature.

[s02-s12] The SignedInfo element contains the information that is actually signed. In order to validate the SignedInfo there are two mandatory processes that need to be performed.

- Validation of the signature over SignedInfo
- Validation of each Reference digest within SignedInfo

Note that the algorithms used in calculating the SignatureValue are also included in the signed information while the SignatureValue element is outside the SignedInfo.

[s03] The CanonicalizationMethod is the algorithm that is used to canonicalize the SignedInfo element before it is digested as part of the signature operation.

[s04] The SignatureMethod is the algorithm that is used to convert the canonicalized SignedInfo into the SignatureValue. It is a combination of a digest algorithm and a key dependent algorithm and possibly other algorithms such as padding, for example RSA-SHA1. The algorithm names are signed to resist attacks based on substituting a weaker algorithm.

[s05-s11] Each Reference element includes the digest method and resulting digest value calculated over the identified data object. It also may include transformations that produced the input to the digest operation. A data object is signed by computing its digest

value and a signature over that value. The signature is later checked via reference and signature validation.

```
[s01] <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
[s02]   <SignedInfo>
[s03]     <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s04]     <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
[s05]     <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06]        <Transforms>
[s07]           <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s08]        </Transforms>
[s09]        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s10]        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[s11]     </Reference>
[s12]   </SignedInfo>
[s13]   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
[s14]   <KeyInfo>
[s15]    <KeyValue>
[s16]      <DSAKeyValue>
[s17]        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
[s18]      </DSAKeyValue>
[s19]    </KeyValue>
[s20]    <X509Data>
[s21]      <X509IssuerSerial>
[s22]         <X509IssuerName>
[s23]             EMAILADDRESS=ssingh@cise.ufl.edu, CN=Sahib Singh Wadhwa, OU=CISE,  O=UF,
[s24]             L=Gainesville, ST=FL, C=US
[s25]         </X509IssuerName>
[s26]         <X509SerialNumber>1</X509SerialNumber>
[s27]      </X509IssuerSerial>
[s28]      <X509SubjectName>CN=localhost, OU=CISE, O=UF, ST=FL, C=US</X509SubjectName>
[s29]      <X509Certificate>
[s30]            MIIDKDCCAtKgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBjTELMAkGA1UEBhMCVVMx
[s31]            CzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtHYWluZXN2aWxsZTELMAkGA1UEChMCVUYx
[s32]            DTALBgNVBAsTBENJU0UxGzAZBgNVBAMTElNhaGliIFNpbmdoIFdhZGh3YTEiMCAG
[s33]            CSqGSIb3DQEJARYTc3NpbmdoQGNpc2UudWZsLmVkdTAeFw0wMzAxMjQxODM5NTJa
[s34]            Fw0wNDAxMjQxODM5NTJaMEoxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJGTDELMAkG
[s35]            A1UEChMCVUYxDTALBgNVBAsTBENJU0UxEjAQBgNVBAMTCWxvY2FsaG9zdDCBnzAN
[s36]            BgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAzMwAw9ZmukeP6Ae3IIKsWsVKya32J5QX
[s37]            ZCjXQc7xT2tdCwRdauwPkFKtEbFj+yFQlTIc4x8MQd8uDzNjHqm7RV63LYjTbL2d
[s38]            EojKI5Lme3mFqwm2ESFv6ClVRuiPxp5iRprwaVQUYFOpZbgBUjD9WsremvLagE99
[s39]            9+pzuHa5ZvcCAwEAAaOCARkwggEVMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8W
[s40]            HU9wZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBRlmKOCiunJ
[s41]            OQcg0aEBb+y+5RjFsTCBugYDVR0jBIGyMIGvgBRrzsFln+bU9fICtrYx+xumHSHP
[s42]            IKGBk6SBkDCBjTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtH
[s43]            YWluZXN2aWxsZTELMAkGA1UEChMCVUYxDTALBgNVBAsTBENJU0UxGzAZBgNVBAMT
[s44]            ElNhaGliIFNpbmdoIFdhZGh3YTEiMCAGCSqGSIb3DQEJARYTc3NpbmdoQGNpc2Uu
[s45]            dWZsLmVkdYIBADANBgkqhkiG9w0BAQQFAANBAFle1sDvcS6aeYOdHEQvmS0D/IC2
[s46]            /+3iH6cwLPESab2Ed8CfyGMyYDE7W1p+lKyTNDuVn5HlUuafOVpSTu8Ra34=
[s47]      </X509Certificate>
[s48]    </X509Data>
[s49]   </KeyInfo>
[s50] </Signature>
```

Figure 5-1 Example of an XML signature

[s14-s49] KeyInfo indicates the key to be used to validate the signature. Possible forms for identification include certificates, key names, and key agreement algorithms and information. KeyInfo is optional for two reasons. First, the signer may not wish to reveal key information to all document processing parties. Second, the information may be known within the application's context and need not be represented explicitly. Since

KeyInfo is outside of SignedInfo, if the signer wishes to bind the keying information to the signature, a Reference can easily identify and include the KeyInfo as part of the signature.

5.3 XML Signature Types

There are three types of XML signatures depending on the arrangement of certain tags [33]. If the signature is parent to the signed object, it is called enveloping. If the signature is child to the signed object, it is called enveloped. If the signature is a sibling to the signed object, it is called detached. Figure 5-2 illustrates the parent child relationships of the signature and the data objects.

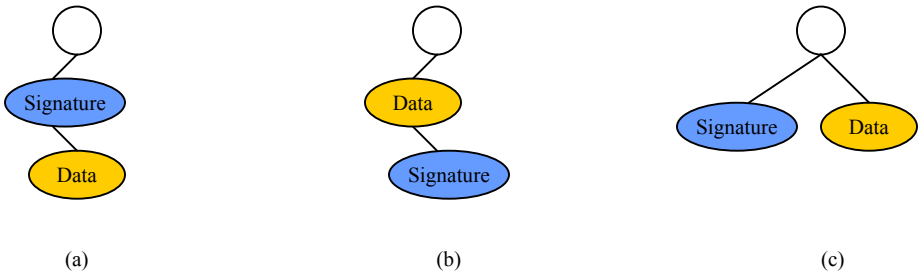

Figure 5-2 Types of XML signatures: (a) Enveloping, (b) Enveloped, (c) Detached

5.3.1 Enveloping Signature

The data entity that is referred to by the Reference element resides in the XML signature itself. This is achieved by using an Object element that is able to contain arbitrary data. Thus, every kind of data, and not only XML data, can be signed by applying this signature type. Figure 5-3 illustrates an enveloping signature.

5.3.2 Enveloped Signature

The data entity that is referred to by the Reference element is the document containing XML signature itself. Thus, the XML signature is wrapped by the referred

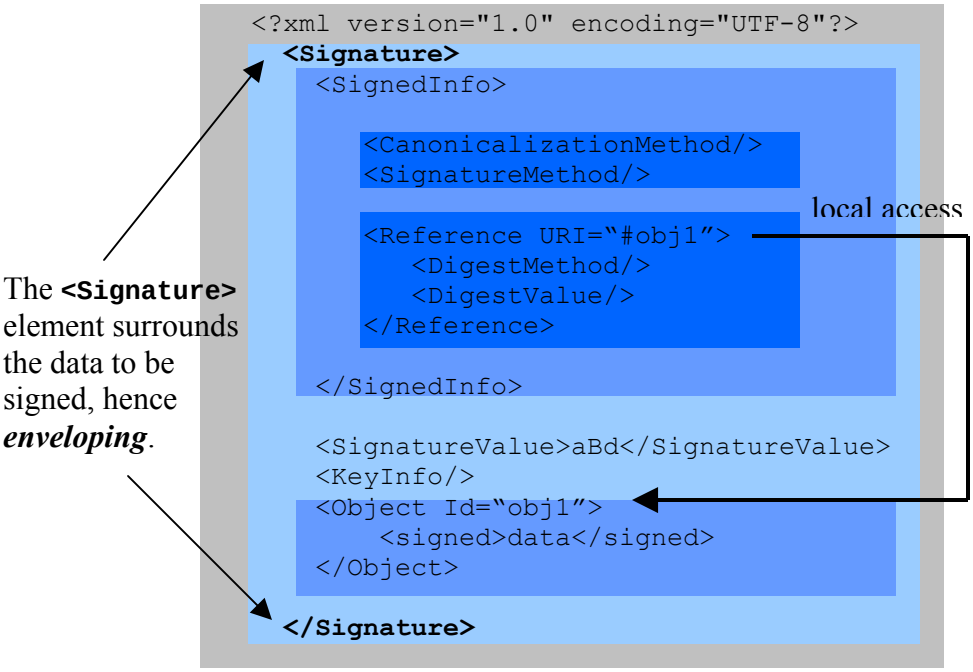

Figure 5-3 Enveloping signature

data entity. From the definition of this signature type it becomes clear that this kind of XML signature can only be used to sign XML data. Figure 5-4 illustrates an enveloped signature.

5.3.3 Detached Signature

The third type of XML signatures is a type where the data entity that is referred to by the Reference element is not contained in an Object element. Furthermore, the data entity does not contain the Signature element. Thus, this type is split up into two sub-types, where the first one describes the case in which the data entity and the XML signature are contained in the same document. In the second case the data entity is located in a separate document. The first case allows only the signing of XML documents, and the second case allows the signing of documents of arbitrary format. Figure 5-5 illustrates a detached signature.

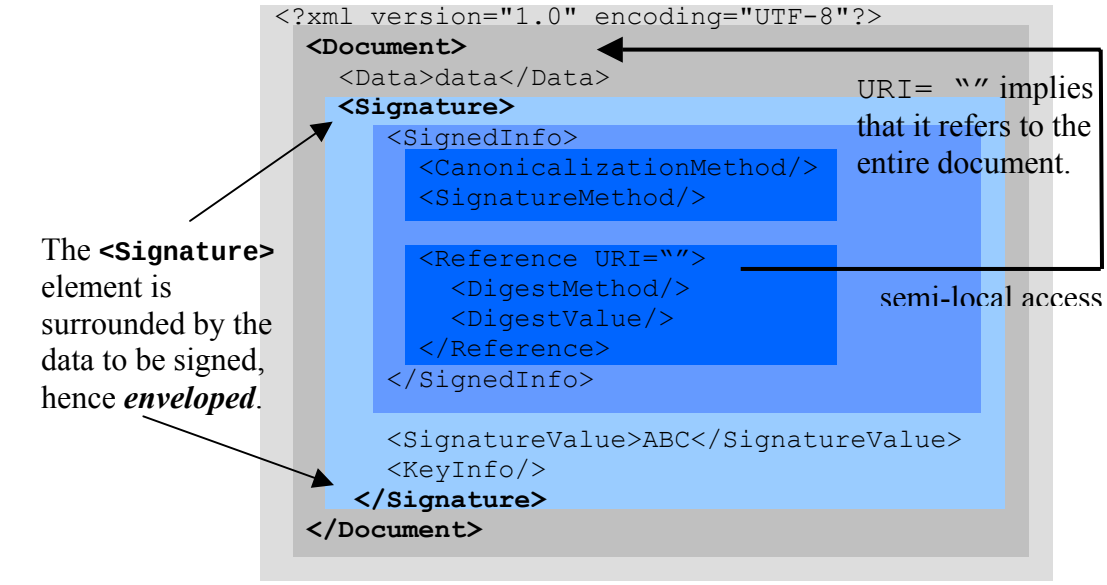

Figure 5-4 Enveloped signature

Figure 5-5 Detached signature

CHAPTER 6
CONTRACT SCHEMA

A contract is an agreement that binds two or more parties to a deal in which an offer is made and accepted to serve the interests of the involved. In the real world an agreement can be formal, informal, written, oral or just an understanding. In order to be legally accepted, a contract in the OCEAN is an XML document digitally signed by the parties involved in the deal.

6.1 Contract Schema

Chokkareddy [9] provides a description of the contract schema pertaining to the design of the Negotiation component. In my thesis I have focused on the <Signature> element of the contract schema. While working on the schema, I modified two elements and eliminated a few elements that did not have relevance, especially, when the negotiation phase is over. Figure 6-1 shows the contract schema visualized using `xmlspy`[19]. The description of the elements is as follows:

1. **Contract**: This is the root element of a contract. It has two attributes viz. "Date" and "Id". "Id" identifies the contract uniquely. Date indicates the date and time when the contract was finally agreed upon by the consent of the participating parties. This "Date" however, does not indicate the time of signature because that would be different for both parties.

2. **Application** specifies the name of the application that a contract belongs to. An XML parser would first parse this node to find out the name of the application the contract belongs to and then handle the contract according to the needs of the particular application. For example: The value of this node for the OCEAN application would be "OCEAN". Once the processor finds the name "OCEAN" in the application element, it will only look for OCEAN specific elements in the contract.

3. **Buyer** specifies the name of the buyer. In OCEAN this will be an OCEAN client id.

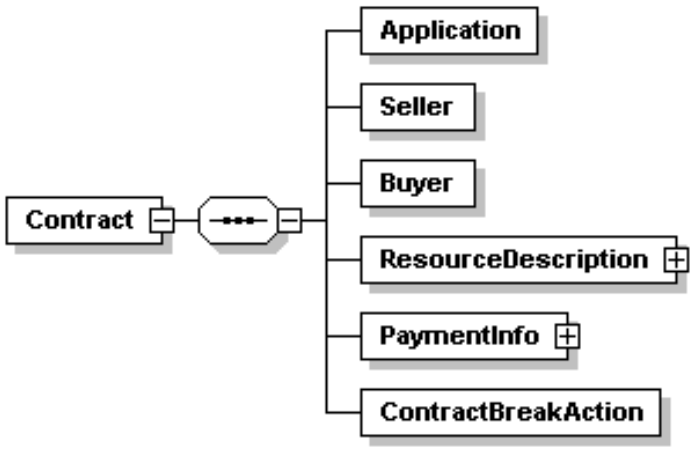

Figure 6-1 Contract schema diagram

4. **Seller** specifies the name of the seller. In OCEAN this will be an OCEAN client id.

5. **ResourceDescription** contains a list of resources being traded. Each resource has a list of features. Each feature has name, value, etc. as attributes.

6. **PaymentInformation** contains information about the payment i.e. the mode and type of payment method. For now the only types allowed for <Mode> are "subscription" and "oneshot" and the only types allowed for <Type> are "paypal"[20] and "eGold"[21] and "OFX"[22]. In future versions of OCEAN more options can be provided.

7. **ContractBreakAction** specifies the necessary action to take in the event a party breaks the contract. In the present implementation the only value allowed for this element is "noAction" and it is planned to provide another option, "kickOut" which will kick the offending party out of the OCEAN system by removing the party from central accounting server, in the near future. In future, it can also be used to lower the rating of the party depending on the seriousness of the issue.

The above list of elements is quite generic but it is designed keeping OCEAN's requirements under consideration. There can be additional elements that find importance in other applications. They can be added when required.

6.2 Signed Contract

When a contract is signed a signature element is generated. Each signatory generates a unique signature element. <SignedContract> element contains the contract element and the all the signature elements. The SignedContract element is illustrated in Figure 6-2

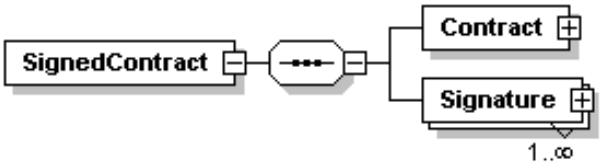

Figure 6-2 Signed contract

- **Signature** contains the signature of the participating party. Figure 6-3 shows the signature element in more detail.

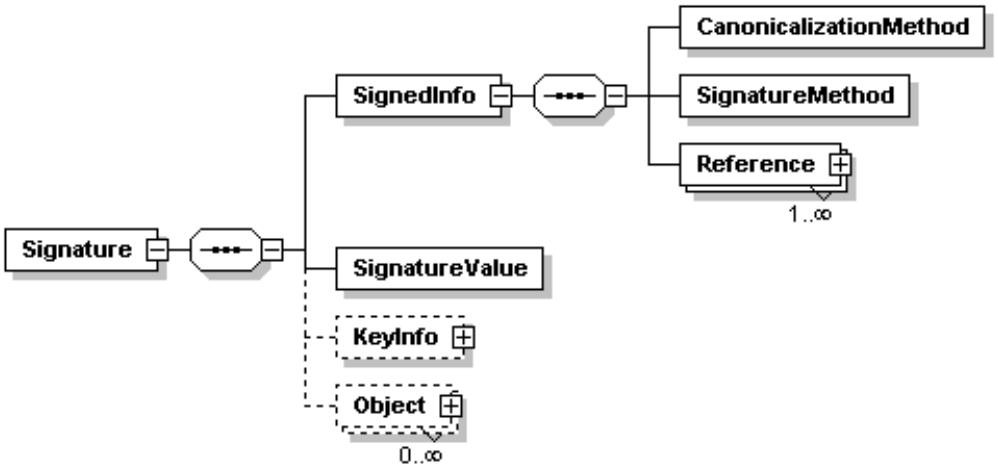

Figure 6-3 Signature element in detail

The explanation of the individual elements of the signature schema is given in Section 5.2. A more detailed diagram of the contract schema and the signature schema are listed in Appendix A. An example of a signed contract is listed in Appendix B.

CHAPTER 7
XML SIGNATURE MODULE

7.1 Architecture

Figure 7-1 shows high-level functional architecture of XML Signature Module.

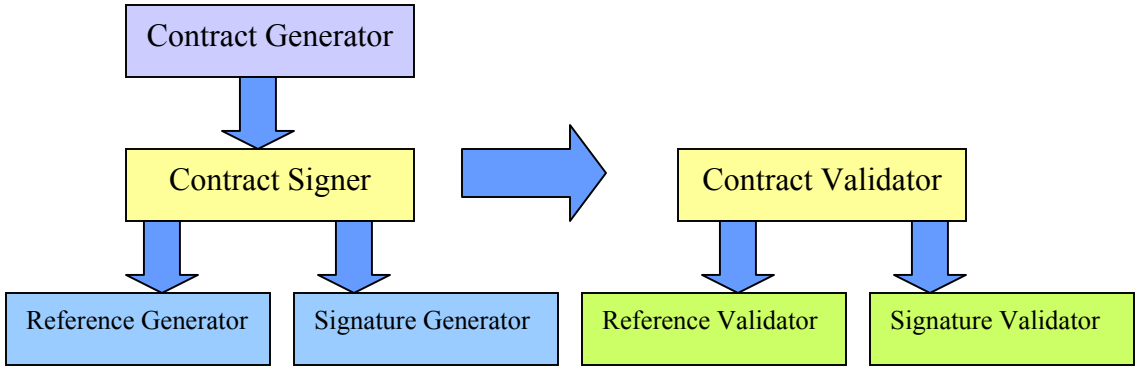

Figure 7-1 Functional architecture of XML Signature Module

7.2 Requirement Analysis

In order to design the system, the following requirements were realized from the Certification Authority, Central Accounting Server and the client.

Responsibilities of the Certification Authority

1. Generate a self-signed certificate and private key using OpenSSL (only once at the time of installation).
2. Issue new certificates
   a. Listen to the CSR's coming from CAS.
   b. Sign CSR and return signed certificate back to CAS.
3. Maintain a Certificate Revocation List (CRL) for revoked certificates.
4. Renew expired certificates on request.

5. Listen to request for certificates from OCEAN users and other users on the Internet.

Responsibilities of the Central Accounting Server

1. Listen to registration requests.
2. Register a new user
3. Communicate with the CA for getting CSR signed.

Responsibilities of a client

1. Collect personal details in the GUI.
2. Generate key-pair using `keytool` and store securely on local machine.
3. Generate a Certificate Signing Request (CSR) using `keytool`.
4. Pack personal details along with CSR in an XML document and send it to the Central Accounting Server (CAS).

7.3 Design

7.3.1 Protocols

7.3.1.1 Registration protocol

In order to sign the contract, the client needs a private-key. In OCEAN, we also append a public-key certificate to the signed contract for verification. A Private key is generated at the time of installation when a client first installs the OCEAN software. Now it needs to obtain a public-key certificate from the OCEAN Certification Authority. This is obtained during the registration process. The registration process is outlined in Figure 7-2.

Steps in Registration

1. The user at the client machine submits the registration details (R) entered in the GUI
2. The client machine generates a key-pair of public and private keys.

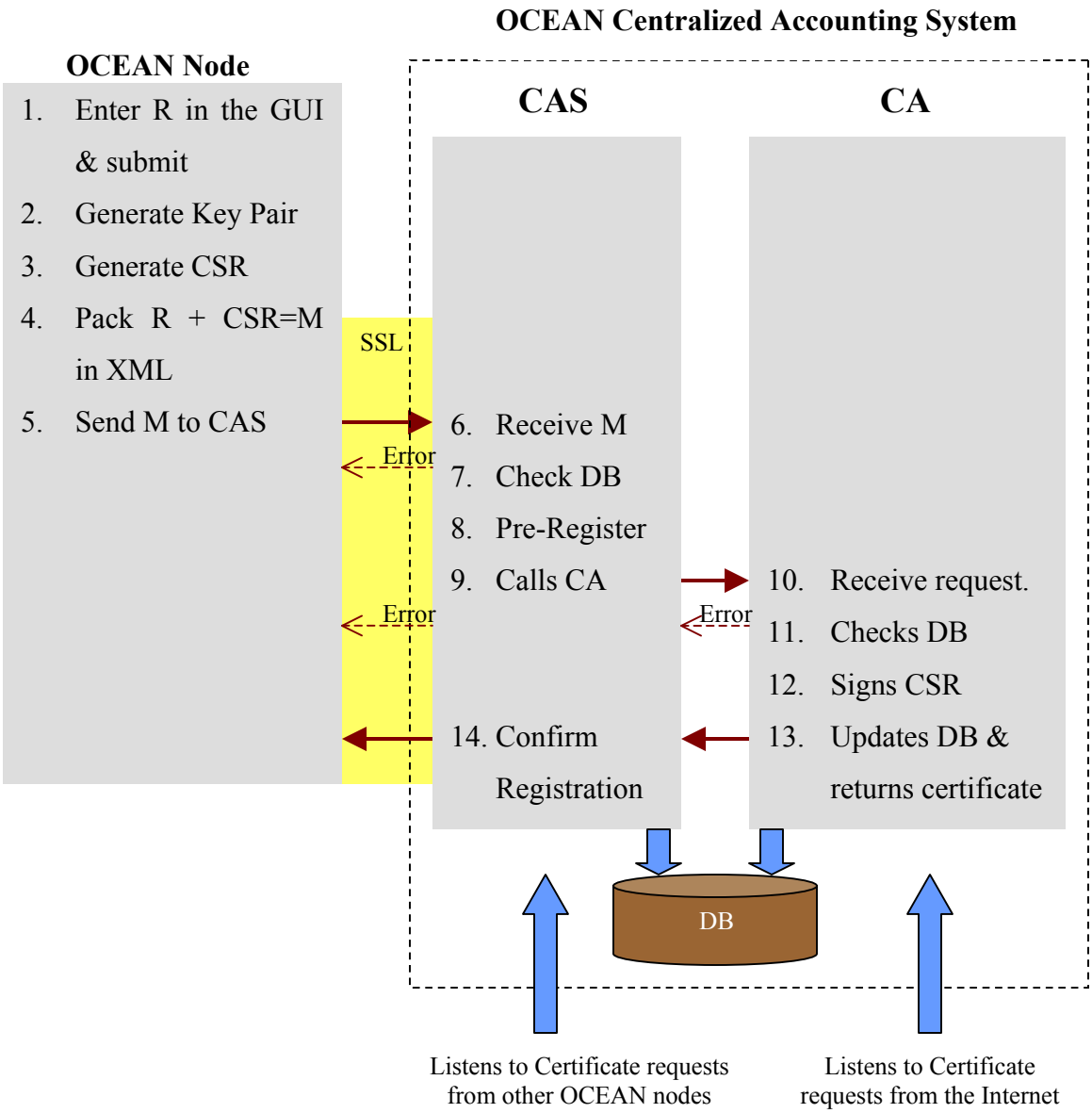

Figure 7-2 Registration protocol

3. A Certificate Signing Request (CSR) is generated using the public key of the client.
4. The Registration Information along with the CSR is packed into an XML document to generate the message M.
5. The Message is sent over SSL to the Central Accounting Server (CAS).
6. The CAS receives the message.

7. Upon receiving M, the CAS checks its database for duplicates. In case there is any, it notifies the client and quits the session. If not, it proceeds to the next step.

8. The CAS pre-registers the client in its database waiting for other information from the CA to complete the registration process.

9. It then calls the CA to issue the client a public-key certificate.

10. The CA receives the request from the CAS.

11. The CA checks for duplicates in its database of existing certificates.

12. After verifying the client by usual methods, it signs the CSR.

13. The CA updates its database and returns the public-key certificate to the CAS.

14. The CAS confirms registration by providing the public-key certificate to the client and updating its database.

7.3.1.2 Signing & Validating protocol

Figure 7-3 shows the steps of the signing and validating protocol.

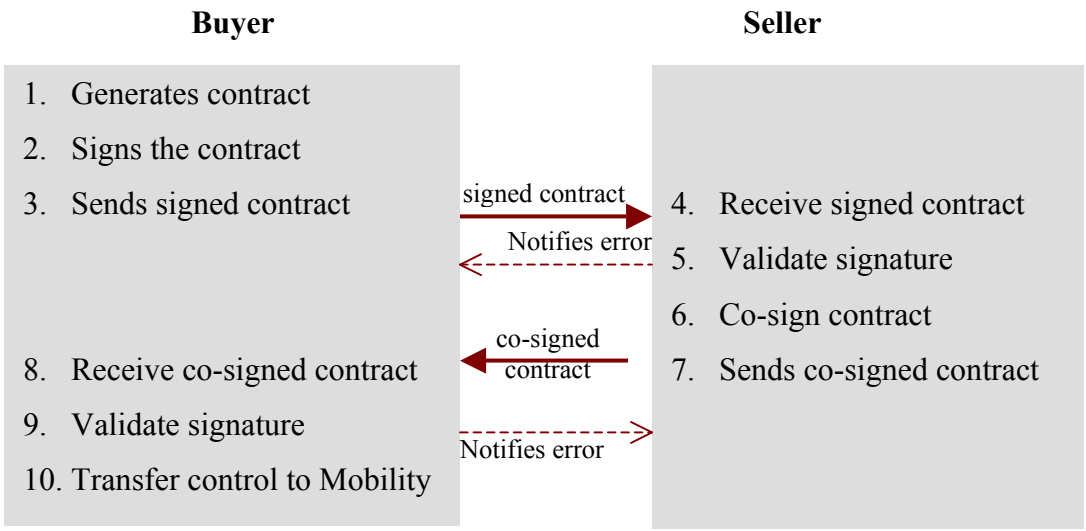

Figure 7-3 Signing and Validating protocol

1. Once Buyer and Seller reach an agreement (after negotiation), Buyer generates the final contract document

2. Buyer signs the contract document with its private key.

3. Buyer sends the signed contract to the Seller for counter signature.

4. Seller receives the signed contract from the Buyer.

5. Seller validates the signature of the Buyer using Buyer's public-key. If correct, then, proceed to step 5 else requests again (possible attack).

6. Seller co-signs the same contract appending its signature using its own private key.

7. Seller sends the counter signed contract to the Buyer. It keeps a copy for its own records.

8. Buyer receives the counter signed contract.

9. Buyer validates the signature of the Seller using seller's public-key. If correct, then, proceed to step 10 else requests again (possible attack).

10. Buyer transfers control to the Mobility component for scheduling and migrating the task.

7.3.2 Signing

Signing an XML document is challenging. Since there are three different types of XML Signatures, a choice has to be made that suits best for a given application. This section gives a detailed explanation of the signing process.

In a generic signature process, the signer and the verifier have the following roles:

Signer

- Calculate a hash code of the message using a strong, one-way hash function.
- Encrypt the hash code using a private key.
- Send the encrypted data along with the original message.

Verifier

- Receive the encrypted data and the original message.
- Calculate the hash code for the message received.
- Decrypt the encrypted hash code using the public key to see if the hash codes matches.

When we deal with XML documents, the data to be signed is not just an entire XML document, but it can be any set of the following:

- A single element.
- More than one element.
- Entire XML document
- An object not residing in the XML document but referred to remotely by an element in the document.

In OCEAN, there will be at least two parties, a buyer and a seller, who will sign the contract before commencing any transaction. The elements in the contract that need to be signed have to be identified. There can be certain items in the contract that might also need to be encrypted for privacy, but in the present version element level encryption is not being done. The buyer and the seller are assumed to be able to maintain security of their own files and SSL secures the transmission.

7.3.2.1 Signing XML documents

In order to understand the process for signing XML documents, lets revisit the basic structure of XML Signature [33]. Figure 7-4 shows the skeleton structure of an XML signature. The process of signing an XML document is two-fold.

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
      <SignatureMethod/>
       (<Reference URI? >
         (<Transforms>)?
           <DigestMethod>
           <DigestValue>
        </Reference>)+
  </SignedInfo>
  <SignatureValue>
       (<KeyInfo>)?
       (<Object ID?>)*
</Signature>
```

Figure 7-4 Skeleton structure of an XML signature

Reference Generation

1. Determine which elements are to be signed.

2. Obtain transforms for each element (this is optional) in <Transforms> tag.

3. Calculate the digest of each element (using the MD5 or SHA1 algorithms). This constitutes <DigestMethod> and <DigestValue> tags.

4. Put each reference element within a <Reference> tag.

Signature Generation

1. Collect all reference elements in <SignedInfo> tag.

2. Apply Canonicalization algorithm. Include a <CanonicalizationMethod> tag.

3. Sign the resulting <SignedInfo> component with algorithm specified in the <SignedInfo>. Put the signature value in a <SignatureValue> tag.

4. Add key info (optional) in <KeyInfo> tag.

5. Enclose everything in a <Signature> tag.

In OCEAN, a contract is signed by at least two parties. First the buyer signs and then the seller signs the contract. Therefore there will be two signature elements in a contract. In an ideal case, the parties should sign the contract simultaneously, but in an e-commerce/e-business environment this is nearly impossible. According to Ben-Or et al. [3] protocols are needed to approximate simultaneity by exchanging partial commitments in a piece-by-piece manner. During such a protocol, one party or another may have a slight advantage; a “fair” protocol keeps this advantage within acceptable limits. Ben-Or et al. [3] present a protocol that is fair in the sense that, at any stage in its execution, the conditional probability that one party cannot commit both parties to the contract given that the other party can, is close to zero. This is true even if both the parties have vastly different computing powers.

The signatories sign the same data in the contract. Hence there is a question of replication of data. Should each signature element replicate the data it signs? In order to avoid replication, we decided to use detached type of XML signature. A **<Contract>**

element contains the necessary terms of the contract. The **<Signature>** element for the buyer contains a reference to the <**Contract**> element and is identified by the attribute "Id", which is the buyer's Id. Similarly, the **<Signature>** element for the seller contains a reference to the <**Contract**> element and is identified by the attribute "Id", which is the seller's Id. Both the signature elements appear as siblings to the **<Contract>** element. The root element of the signed contract is called **<SignedContract>**.

This approach is very useful in situations where we have multiple signatories. Suppose in future we need a contract to be signed by more than two parties, additional signatories can refer to the same data in the **<Contract>** element. Also, if in future we wish to sign the match request, we can easily incorporate it in our design because a match request would require only one signature. For instance, the buyer will sign the match request (offer document) whereas the seller will sign the match response (bid document). Figure 7-6 shows the flowchart for signing process in OCEAN.

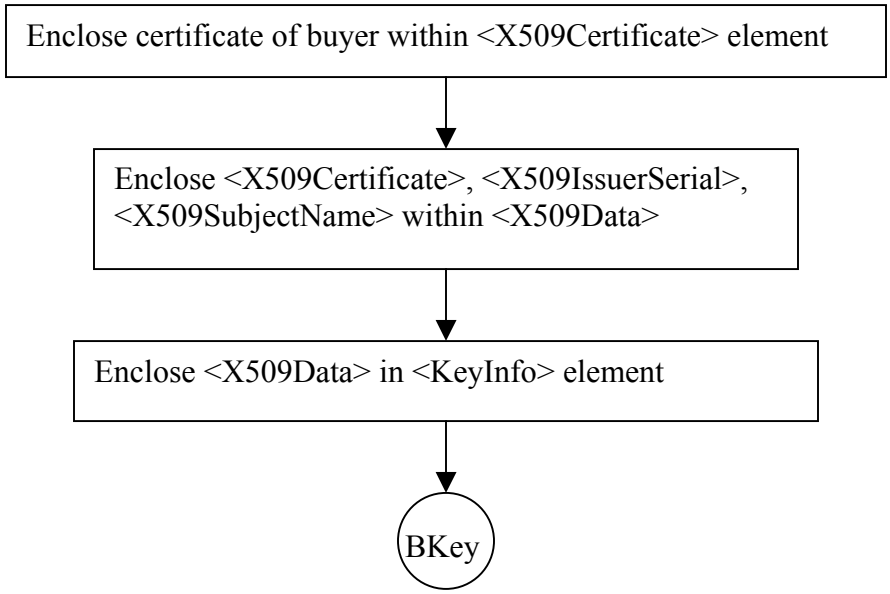

Figure 7-5 Flowchart for generating the <KeyInfo> element of Buyer (procedure for Seller is similar)

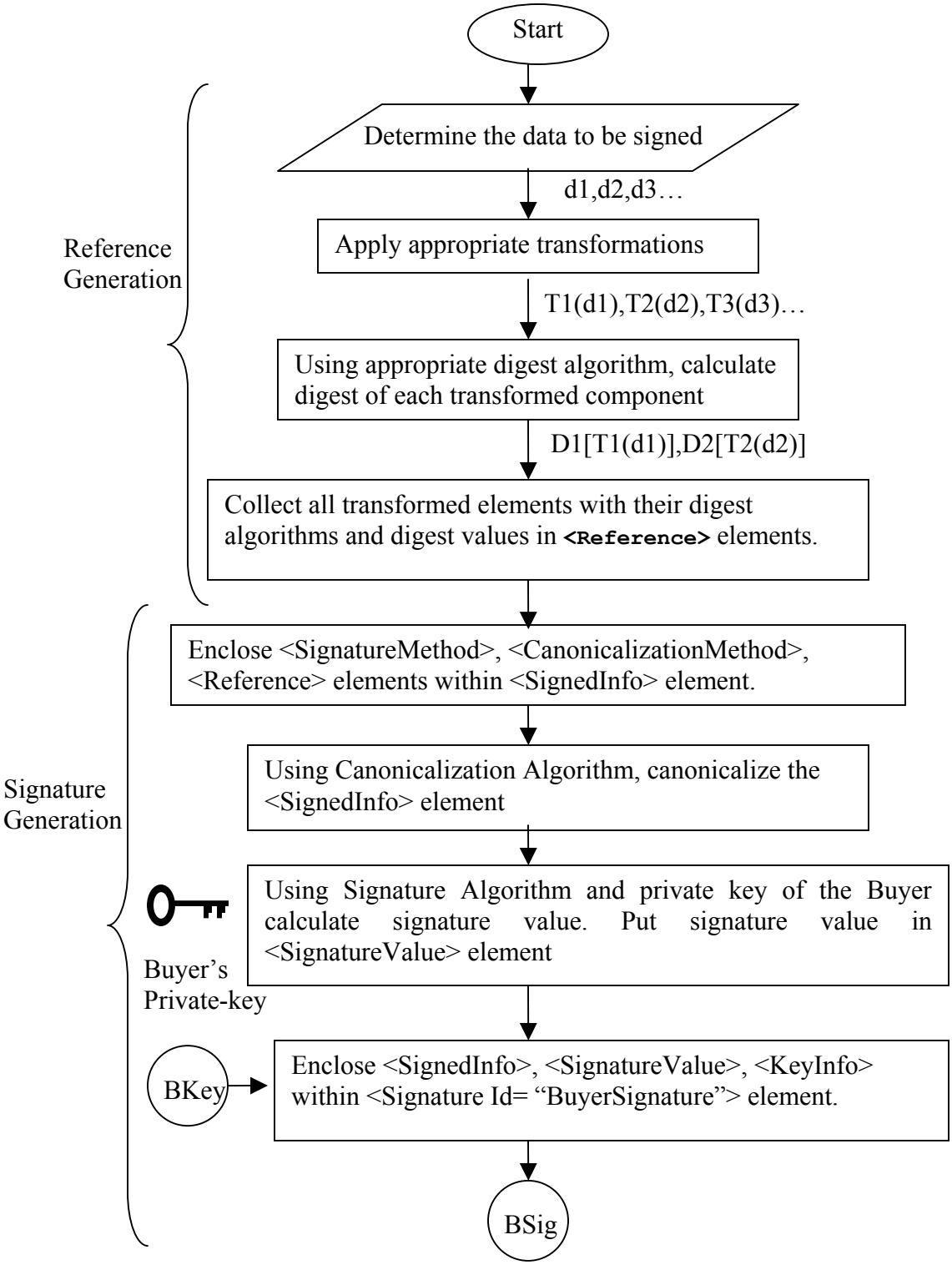

Figure 7-6 Flowchart for generating the **<Signature>** element of the Buyer

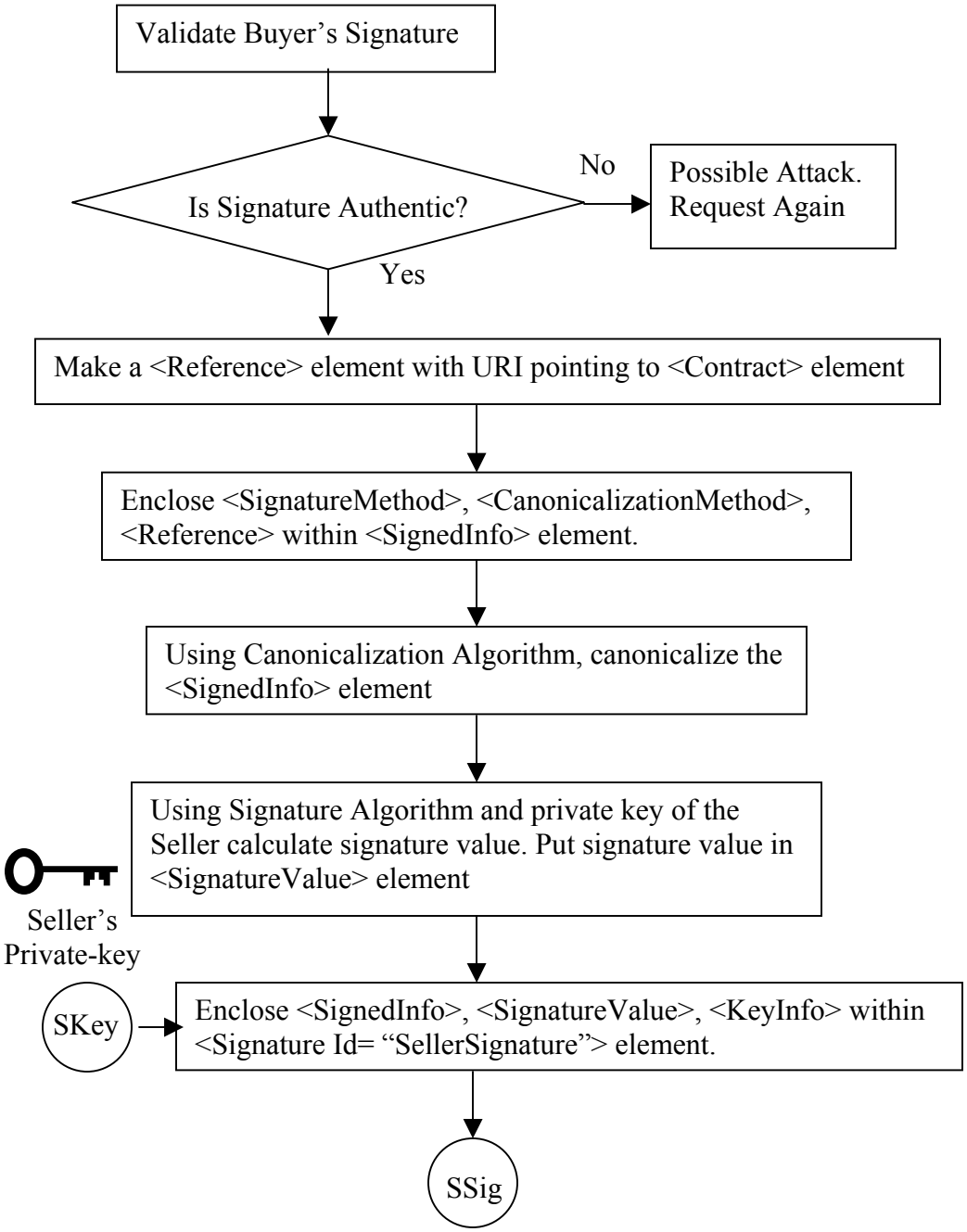

Figure 7-7 Flowchart for generating the **<Signature>** element of the Seller**.**

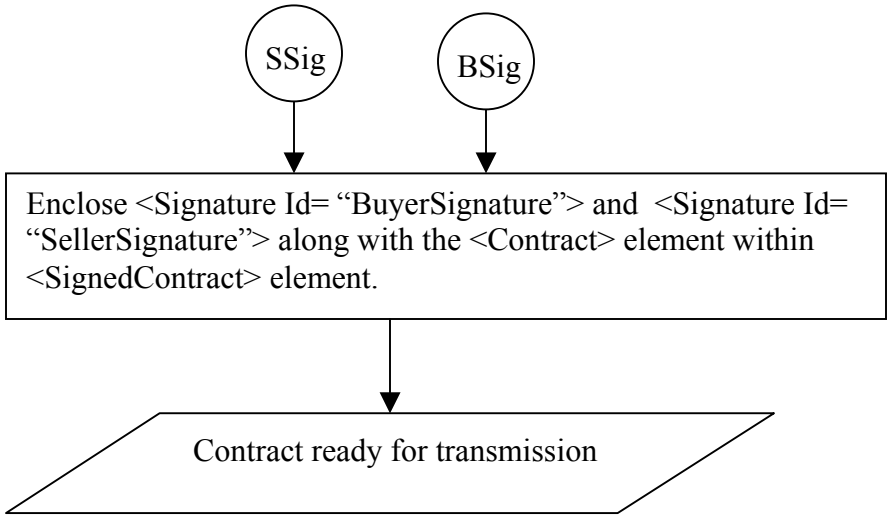

Figure 7-8 Flowchart for generating the root element of the contract

7.3.3 Validation

Like Signing, the process of validating a signed XML document is also two-fold.

7.3.3.1 Signature Validation

Verify signature value of the **<SignedInfo>** element. Recalculate digest of **<SignedInfo>** (using digest algorithm specified in **<SignatureMethod>**) and use public key to verify that the value of **<SignatureValue>** is correct for the digest of the **<SignedInfo>.** If it matches, then proceed to reference validation.

7.3.3.2 Reference Validation

1. Canonicalize the <SignedInfo> element based on the canonicalization method in the <SignedInfo>.

2. For each reference in the <SignedInfo>

    a. Obtain the data object to be digested. (For example, the signature application may dereference the URI and execute Transforms provided by the signer in the reference element, or it may obtain the content through other means such as a local cache.)

    b. Digest the resulting data object using the **DigestMethod** specified in its reference specification.

c. Compare the generated digest value against **`DigestValue`** in the **`SignedInfo`** reference. If there is any mismatch, validation fails.In OCEAN, the validation process does the following

- Check certificate using public key of the CA (this key is shipped along with the OCEAN package)
- Check the certificate for the expiration date
- Optional certificate verification for revocation from the CA

Figure 7-9 shows the flowchart for validation process in OCEAN.

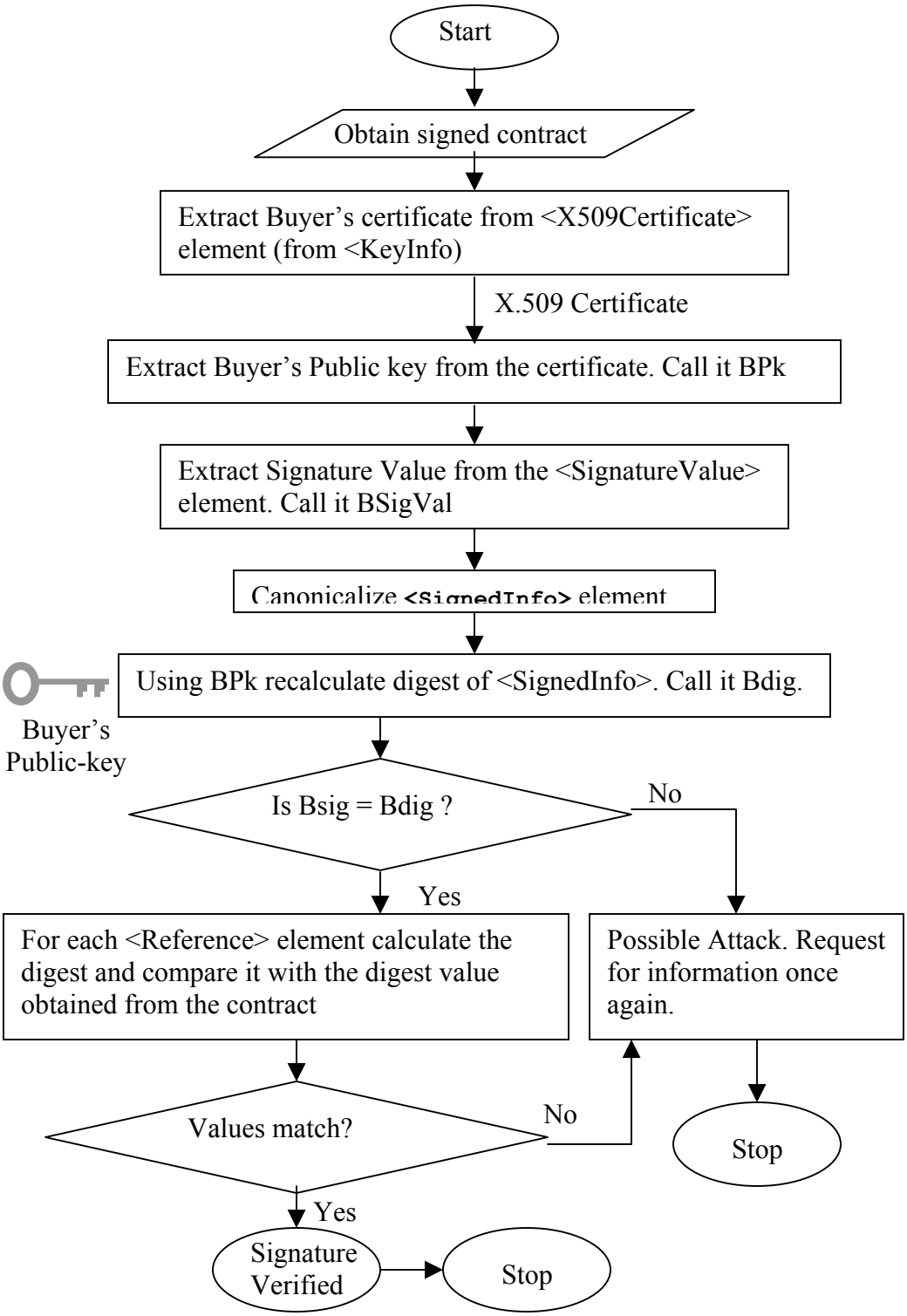

Figure 7-9 Flowchart for validating Buyer's signature by the Seller (similar procedure for the inverse)

CHAPTER 8
API

8.1 Registration

It is usually cumbersome to generate XML documents manually. Therefore two different GUI's have been developed to make things easier. First GUI is used during the registration process and the second during the generation of the contract.

`class Registration`

This class interacts with two classes namely PersonalDetails and CSRGenerator. The class diagram is shown in Figure 8-1.

`class PersonalDetails`

This class provides a GUI that obtains registration details from the OCEAN user. The class diagram is shown in Figure 8-1.

`class CSRGenerator`

This class generates a Certificate Signing Request (CSR). A CSR is sent to the Certification Authority in order for it to issue a certificate. In order to generate a CSR it has to initialize a KeyStore, generate a key-pair and store it in user's home directory. The class diagram is shown in Figure 8-1.

`class SimpleClient`

This is a client program responsible for establishing an SSL socket connection with the Certification Authority. Once a connection is established, this program sends the XML file containing CSR and the registration details to the Certification Authority. The Registration class interacts with this class as shown in Figure 8-1.

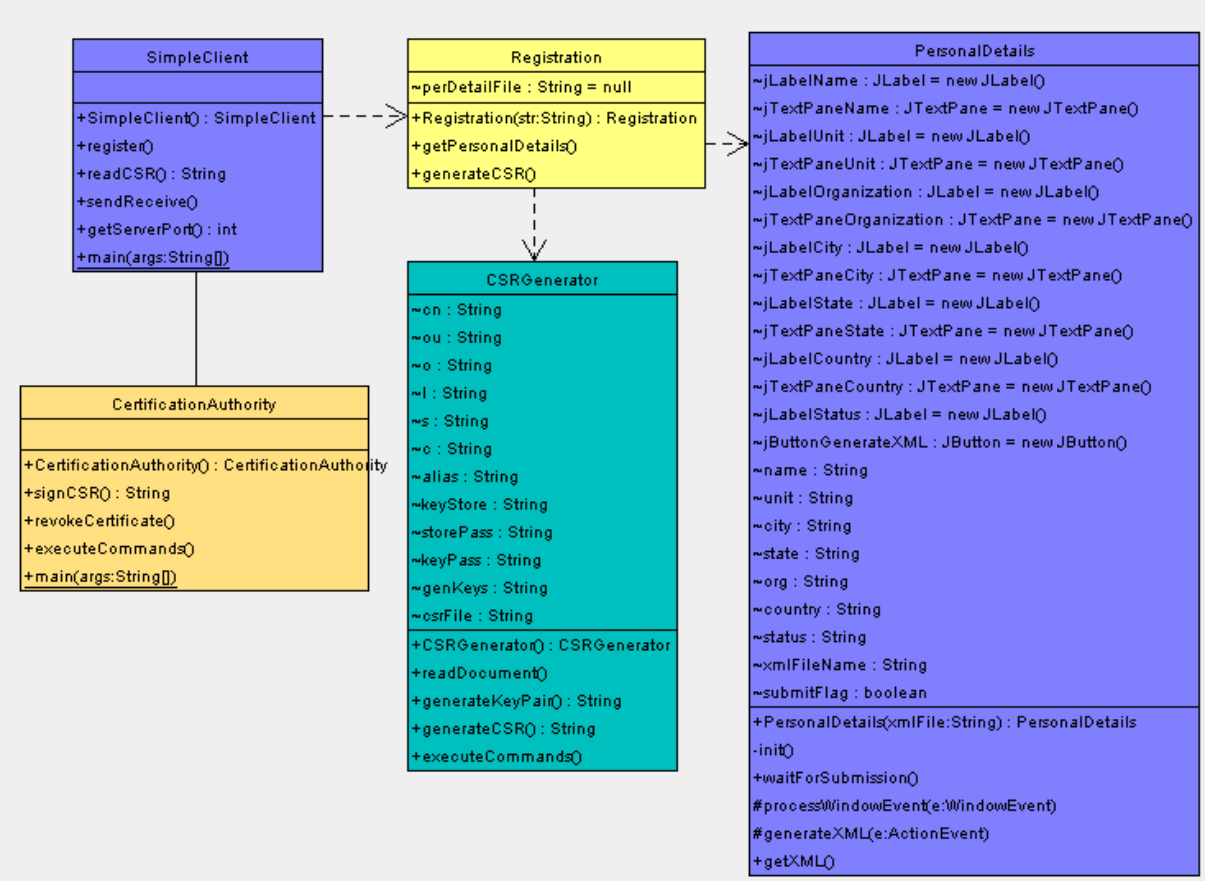

Figure 8-1 UML diagram for the Registration Protocol

`class CertificationAuthority`

This is a server program that is developed to listen to the client requests for issuing certificates. It is also planned that in the future, this class will be capable of listening to the requests for certificates from the non-OCEAN nodes on the Internet. The class diagram is shown in Figure 8-1.

8.2 Signing and Validation

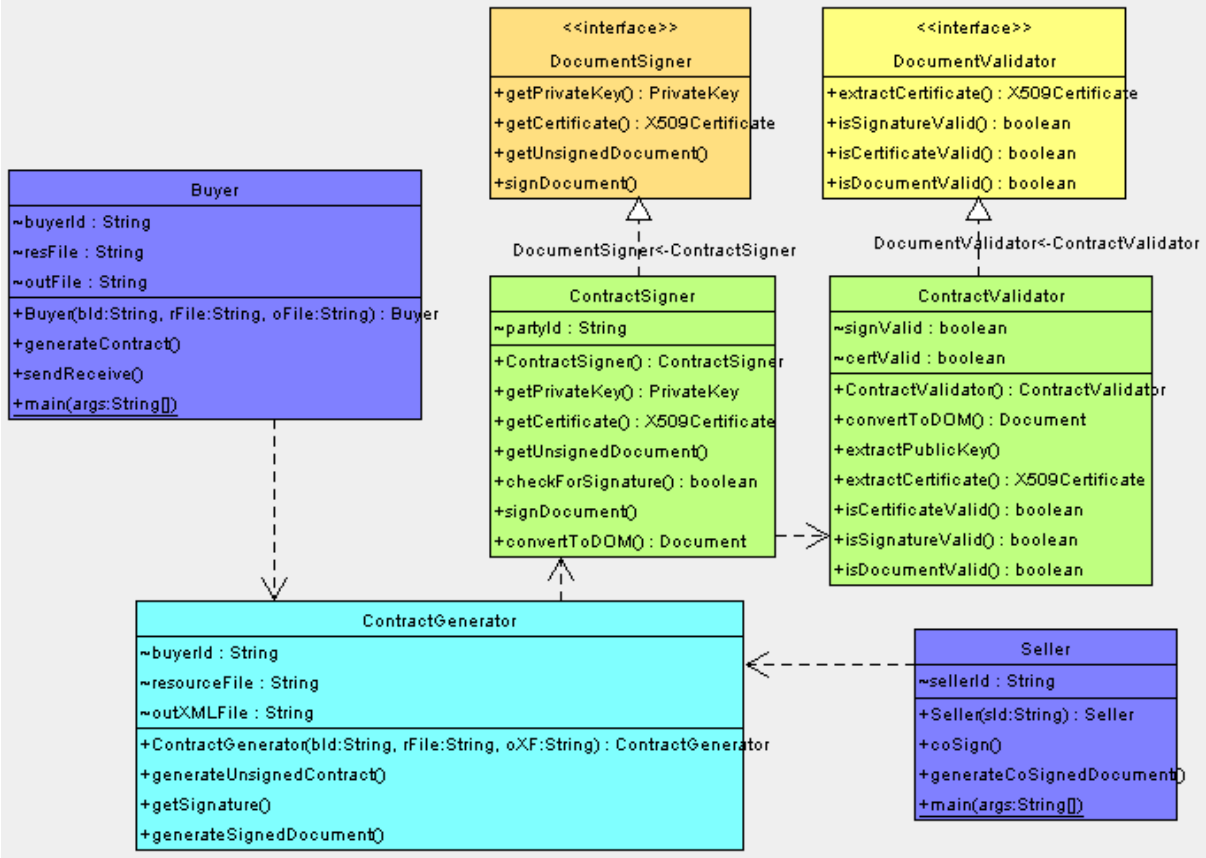

Figure 8-2 UML diagram for the Signing and Validating Protocol

`class Buyer`

The buyer uses this class to establish a socket based connection with the seller. It calls the ContractGenerator to generate a contract, the ContractSigner to sign it and then send the signed contract to the seller node over TCP based connection. The class diagram is shown in Figure 8-2.

class Seller

This is a stub class similar to the Buyer, developed to test the signing and validation mechanism. It acts as a server that listens to the incoming requests. In the actual release version this functionality would be provided every OCEAN node irrespective of whether it is a buyer node or seller node.

class ResourceDescription

ResourceDescription provides a GUI that was developed for testing purposes only to input the resource description. It would not appear in the OCEAN software because a user is not supposed to input resource description via GUI, it has to done automatically. The resource description would be provided by the Negotiation component after an agreement has been finalized.

interface DocumentSigner

This interface defines the basic functions necessary to sign any XML document. ContractSigner implements this interface to sign contracts. The class diagram is shown in Figure 8-2.

interface DocumentValidator

This interface defines the basic functions necessary to validate any XML document. ContractValidator implements this interface to validate contracts. The class diagram is shown in Figure 8-2.

class ContractGenerator

This class obtains a contract in an XML document from ResourceDescription and gets it signed by the ContractSigner. The class diagram is shown in Figure 8-2.

`class ContractSigner`

ContractSigner signs the contract generated by the ContractGenerator. The class diagram is shown in Figure 8-2.

`class ContractValidator`

ContractValidator checks for the validity of a contract by verifying the signature. If a contract needs co-signature it checks the validity of the contract before the signature process proceeds. The class diagram is shown in Figure 8-2.

CHAPTER 9
CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

It is expected that there would not be any repudiability problems or any problems pertaining to the forgery of contracts provided that such documents are signed using the XML Signature Module. The signature schema is very flexible in a sense that if in future there are more than two parties, it can simply include additional signatures as siblings to the existing ones in the same XML document. The data that is referenced by the signatory does not need to be replicated each time. A key benefit of using the XSM is its modular structure. It can be integrated into an existing system with very little programming overhead.

9.2 Future Work

There are umpteen security issues in OCEAN that need to be addressed in order to make a one hundred percent leak proof system. One of the very important issues that needs attention pertains to the correctness of computation at the seller's machine. Not just the correctness but also whether it actually took place, or if it did then how far did it go in situations (for example: accidental termination due to power shortage or a natural disaster) where it got aborted without any personal intent. In certain compute-intensive calculations or computations of great value, the buyer would like to obtain partial results in order to be assured that the seller's machine is performing the job right. This demand gains importance when we are confident in the design of a given algorithm *A*, but less so in the physical computer that runs it [20]. For instance, the computer hardware may be

defective. Alternatively, the hardware may function properly, but the operating system may be flawed. Alternatively yet, hardware and software may be fine, but some alpha rays succeed in flipping a bit together with its controls, so that the original bit value is not restored [20], or the operator of the machine may maliciously interfere with its operation.

According to Micali [20], certified computation provides a special way to answer the question "*Even assuming that the physical computer is correct, after running A on input x so as to obtain a result y, can we convince someone else that y is indeed equal to A(x) without having him re-do the computation himself ? Certified computations can in principle be quite useful when 'contracting out' computer time."*

The agreement in a contract can be redefined in a way that a buyer will pay the seller only if the seller returns the results of executing A on a given input x on its computer and gives back the value $y = A(x)$ together with the certificate of correct execution, which can be quickly verified by the buyer.

In future versions of OCEAN, we may look into implementing Micali's [20] computation certification algorithm to prevent disputes about whether a contracted computation was performed correctly.

.

APPENDIX A
SCHEMAS

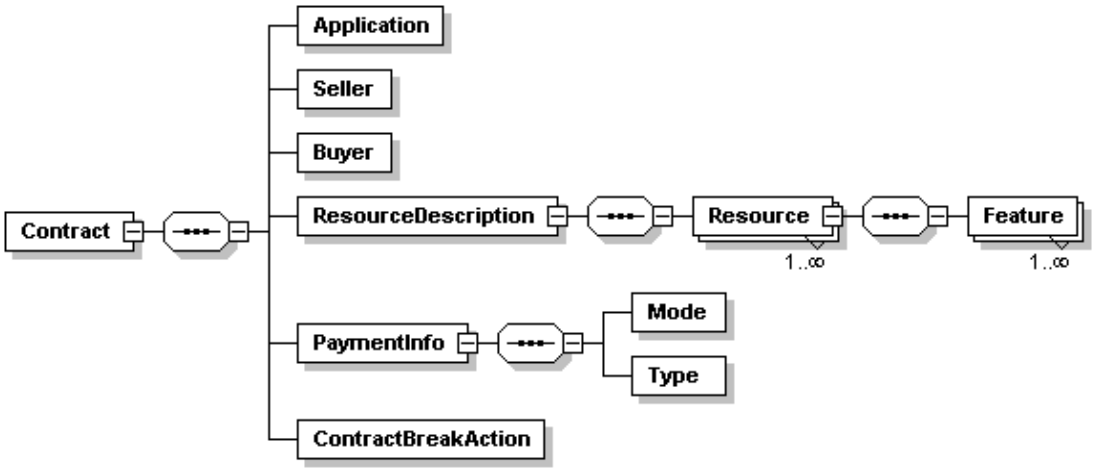

Figure A-1 Detailed contract schema diagram

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Contract">
   <xs:complexType>
    <xs:sequence>
        <xs:element name="Application"/>
        <xs:element name="Seller"/>
        <xs:element name="Buyer"/>
        <xs:element name="ResourceDescription">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Resource" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Feature" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:attribute name="value" type="xs:anySimpleType" use="required"/>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="name" type="xs:string" use="required"/>
                </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        </xs:element>
        <xs:element name="PaymentInfo">
          <xs:complexType>
            <xs:sequence>
                <xs:element name="Mode"/>
                <xs:element name="Type"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="ContractBreakAction"/>
    </xs:sequence>
    <xs:attribute name="Date" type="xs:dateTime" use="required"/>
    <xs:attribute name="Id" type="xs:string" use="required"/>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure A-2 Contract schema

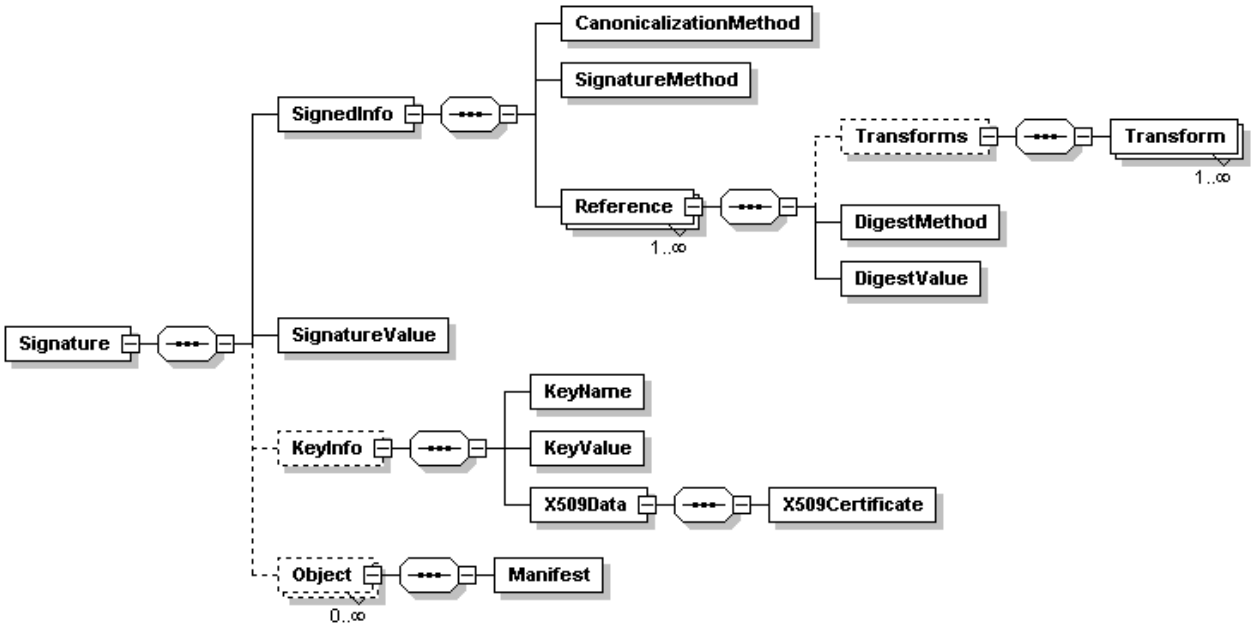

Figure A-3 Detailed signature schema diagram

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Signature">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="SignedInfo">
                <xs:complexType>
                  <xs:sequence>
                     <xs:element name="CanonicalizationMethod"/>
                     <xs:element name="SignatureMethod">
                        <xs:complexType>
                          <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
                        </xs:complexType>
                     </xs:element>
                     <xs:element name="Reference" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="Transforms" minOccurs="0">
                               <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="Transform" maxOccurs="unbounded"/>
                                  </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="DigestMethod"/>
                            <xs:element name="DigestValue"/>
                          </xs:sequence>
                        </xs:complexType>
```

```
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="Id" type="xs:ID" use="optional"/>
            </xs:complexType>        </xs:element>
        <xs:element name="SignatureValue"/>
        <xs:element name="KeyInfo" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="KeyName"/>
                    <xs:element name="KeyValue"/>
                    <xs:element name="X509Data">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="X509Certificate"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Object" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="Manifest">
                        <xs:complexType>
                            <xs:attribute name="Id" type="xs:ID" use="optional"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="Id" type="xs:ID" use="optional"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:ID" use="optional"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure A-4 Signature schema

APPENDIX B
SAMPLE XML DOCUMENTS

B.1 Registration

When a user enters the personal details in the GUI, PersonalDetails.java and Registration.java generate the following output file (register.xml) in the "Distinguished Name" format.

```
<?xml version="1.0" encoding="UTF-8"?>
<DistinguishedName>
  <Name>sahib</Name>
  <Unit>cise</Unit>
  <Organization>uf</Organization>
  <City>gainesville</City>
  <State>fl</State>
  <Country>usa</Country>
</DistinguishedName>
```

B.2 Contract Generation

B.2.1 Resource Description

The first step of generating a contract is to specify the resources. Resource description is entered using a GUI. Following XML file (resource.xml) is generated initially.

```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceDescription>
   <Resource>
      <Feature name= "CPUType" value= "Pentium">
    </Resource>
    <Resource>
      <Feature name= "PrimaryMemory" value= "256">
    </Resource>
    <Resource>
      <Feature name= "SecondaryMemory" value= "1000">
    </Resource>
    <Resource>
      <Feature name= "OS" value= "Solaris">
    </Resource>
    <Resource>
      <Feature name= "Grid" value= "OCEAN">
```

```
    </Resource>
    <Resource>
      <Feature name= “Price” value= “2”>
    </Resource>
</ResourceDescription>
```

B.2.2 Contract Signing

The XML document below is an example of a signed contract.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SignedContract>
  <Contract Id= “C123” Date= “03252003”>
   <ResourceDescription>
    <Resource>
        <Feature name= “CPUType” value= “Pentium”>
    </Resource>
    <Resource>
        <Feature name= “PrimaryMemory” value= “256”>
    </Resource>
    <Resource>
        <Feature name= “SecondaryMemory” value= “1000”>
    </Resource>
    <Resource>
        <Feature name= “OS” value= “Solaris”>
    </Resource>
    <Resource>
        <Feature name= “Grid” value= “OCEAN”>
    </Resource>
    <Resource>
        <Feature name= “Price” value= “2”>
    </Resource>
   </ResourceDescription>
  </Contract>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
    <ds:Reference URI="#xpointer(//Contract)">
     <ds:Transforms>
   <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
     </ds:Transforms>
     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
     <ds:DigestValue>O34xCdxTjQHhN1rMZCufBvSmCjE=</ds:DigestValue>
    </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue>MC0CFQCIZS/FvfpizjUVUIm7aA/c39EAZZodKft3mZXNzA=</ds:SignatureValue>
   <ds:KeyInfo>
    <ds:KeyName>Public key of certificate</ds:KeyName>
    <ds:KeyValue>
     <ds:RSAKeyValue>
      <ds:Modulus>zMwAw9ZmukeP6Ae3IIKsWsVKya32J5Qc7xT2tdCwRdauwPkFKtEbFj+yFQlTIc4x8M
Qd8uDzNjHqm7RV63LYjTbL2dEojKI5Lme3mFqwm2ESFv6ClVRuiPxp5iRprwaVQUYFOpZbgB UjD9WsremvLagE999+pzuHa5Zvc=
      </ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
     </ds:RSAKeyValue>
    </ds:KeyValue>
    <ds:X509Data>
     <ds:X509Certificate>
        MIIDKDCCAtKgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBjTELMAkGA1UEBhMCVVMxCzAJBgNV
        BAgTAkZMMRQwEgYDVQQHEwtHYWluZXN2aWxsZTELMAkGA1UEChMCVUYxDTALBgNVBAsTBENJ
        U0UxGzAZBgNVBAMTElNhaGliIFNpbmdoIFdhZGh3YTEiMCAGCSqGSIb3DQEJARYTc3Npbmdo
        QGNpc2UudWZsLmVkdTAeFw0wMzAxMjQxODM5NTJaFw0wNDAxMjQxODM5NTJaMEoxCzAJBgNV
        BAYTAlVTMQswCQYDVQQIEwJGTDELMAkGA1UEChMCVUYxDTALBgNVBAsTBENJU0UxEjAQBgNV
        BAMTCWxvY2FsaG9zdDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAzMwAw9ZmukeP6Ae3
        IIKsWsVKya32J5QXZCjXQc7xT2tdCwRdauwPkFKtEbFj+yFQlTIc4x8MQd8uDzNjHqm7RV63
        LYjTbL2dEojKI5Lme3mFqwm2ESFv6ClVRuiPxp5iRprwaVQUYFOpZbgBUjD9WsremvLagE99
        9+pzuHa5ZvcCAwEAAaOCARkwggEVMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9wZW5T
        U0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBRlmKOCiunJOQcg0aEBb+y+5RjF
        sTCBugYDVR0jBIGyMIGvgBRrzsFln+bU9fICtrYx+xumHSHPIKGBk6SBkDCBjTELMAkGA1UE
        BhMCVVMxCzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtHYWluZXN2aWxsZTELMAkGA1UEChMCVUYx
        DTALBgNVBAsTBENJU0UxGzAZBgNVBAMTElNhaGliIFNpbmdoIFdhZGh3YTEiMCAGCSqGSIb3
        DQEJARYTc3NpbmdoQGNpc2UudWZsLmVkdYIBADANBgkqhkiG9w0BAQQFAANBAFle1sDvcS6a
        eYOdHEQvmS0D/IC2/+3iH6cwLPESab2Ed8CfyGMyYDE7W1p+lKyTNDuVn5HlUuafOVpSTu8R a34=
     </ds:X509Certificate>
    </ds:X509Data>
```

```
   </ds:KeyInfo>
  </ds:Signature>
</SignedContract>
```

A co-signed contract has an additional signature element as a sibling to the existing signature element.

APPENDIX C
MISCELLANEOUS

The following is an example of a Certificate Signing Request (CSR) generated on a Windows XP platform using keytool.

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBoDCCAQkCAQAwYDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtHYWluZXN2
aWxsZTELMAkGA1UEChMCVUYxDTALBgNVBAsTBENJU0UxEjAQBgNVBAMTCWxvY2FsaG9zdDCBnzAN
BgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAzMwAw9ZmukeP6Ae3IIKsWsVKya32J5QXZCjXQc7xT2td
CwRdauwPkFKtEbFj+yFQlTIc4x8MQd8uDzNjHqm7RV63LYjTbL2dEojKI5Lme3mFqwm2ESFv6ClV
RuiPxp5iRprwaVQUYFOpZbgBUjD9WsremvLagE999+pzuHa5ZvcCAwEAAaAAMA0GCSqGSIb3DQEB
BAUAA4GBAMnMY1hBzIy2Plw5BadIuz1Vn5VkBQXV7/q7FYG2OF5dGU0gaW6mj8sOgrn7MQ2Iiscv
/nXFlbVAexohmB9LWbpN/p7OD/lI74AQ/hXl+adJ8vvNs5K9aG3QM1gZ8fyraH8FdWTNXOezzx3u
6lbzXJxmjW9V821fAh+hdqynWs0P
-----END NEW CERTIFICATE REQUEST-----
```

The following is a sample certificate for the above CSR signed by the homemade Certificate Authority using OpenSSL.

```
-----BEGIN CERTIFICATE-----
MIIDKDCCAtKgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBjTELMAkGA1UEBhMCVVMx
CzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtHYWluZXN2aWxsZTELMAkGA1UEChMCVUYx
DTALBgNVBAsTBENJU0UxGzAZBgNVBAMTElNhaGliIFNpbmdoIFdhZGh3YTEiMCAG
CSqGSIb3DQEJARYTc3NpbmdoQGNpc2UudWZsLmVkdTAeFw0wMzAxMjQxODM5NTJa
Fw0wNDAxMjQxODM5NTJaMEoxCzAJBgNVBAYTAlVTMQswCQYDVQQIEwJGTDELMAkG
A1UEChMCVUYxDTALBgNVBAsTBENJU0UxEjAQBgNVBAMTCWxvY2FsaG9zdDCBnzAN
BgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAzMwAw9ZmukeP6Ae3IIKsWsVKya32J5QX
ZCjXQc7xT2tdCwRdauwPkFKtEbFj+yFQlTIc4x8MQd8uDzNjHqm7RV63LYjTbL2d
EojKI5Lme3mFqwm2ESFv6ClVRuiPxp5iRprwaVQUYFOpZbgBUjD9WsremvLagE99
9+pzuHa5ZvcCAwEAAaOCARkwggEVMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8W
HU9wZW5TU0wgR2VuZXJhdGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBRlmKOCiunJ
OQcg0aEBb+y+5RjFsTCBugYDVR0jBIGyMIGvgBRrzsFln+bU9fICtrYx+xumHSHP
IKGBk6SBkDCBjTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkZMMRQwEgYDVQQHEwtH
YWluZXN2aWxsZTELMAkGA1UEChMCVUYxDTALBgNVBAsTBENJU0UxGzAZBgNVBAMT
ElNhaGliIFNpbmdoIFdhZGh3YTEiMCAGCSqGSIb3DQEJARYTc3NpbmdoQGNpc2Uu
dWZsLmVkdYIBADANBgkqhkiG9w0BAQQFAANBAFle1sDvcS6aeYOdHEQvmS0D/IC2
/+3iH6cwLPESab2Ed8CfyGMyYDE7W1p+lKyTNDuVn5HlUuafOVpSTu8Ra34=
-----END CERTIFICATE-----
```

APPENDIX D
LIST OF URLS

1. http://www.openevidence.org (last accessed 04/03/03)
2. http://www.betrust.com (last accessed 04/03/03)
3. http://www.abanet.org/scitech/ec/isc (last accessed 04/03/03)
4. http://www.w3.org/TR/2002/CR-soap12-part0-20021219 (last accessed 04/03/03)
5. http://www.griphyn.org (last accessed 04/03/03)
6. http://www.globus.org (last accessed 04/03/03)
7. http://www.tryllian.com (last accessed 04/03/03)
8. http://www.beowulf.org (last accessed 04/03/03)
9. http://setiathome.ssl.berkeley.edu (last accessed 04/03/03)
10. http://distributed.net (last accessed 04/03/03)
11. http://www.gnutella.com (last accessed 04/03/03)
12. http://wwws.sun.com/software/gridware (last accessed 04/03/03)
13. http://www.cs.wisc.edu/condor (last accessed 04/03/03)
14. http://www.cs.virginia.edu/~legion (last accessed 04/03/03)
15. http://www.ubero.com (last accessed 04/03/03)
16. Sun Microsystems, *Java Development Kit*, www.java.sun.com (last accessed 04/03/03)
17. OpenSSL, http://www.openssl.org (last accessed 04/03/03)
18. ITU-T, X.500 naming standard, http://java.sun.com/products/jndi/tutorial/ldap/models/x500.html (last accessed 04/03/03)
19. Altova, Inc., http://www.altova.com (last accessed 04/03/03)

20. http://www.paypal.com (last accessed 04/03/03)

21. http://www.e-gold.com (last accessed 04/03/03)

22. http://www.ofx.net (last accessed 04/03/03)

LIST OF REFERENCES

[1] American Bar Association, *Digital Signature Guidelines*, Chicago, August 1, 1996

[2] American Bar Association, *PKI Assessment Guidelines*, Chicago, June 18, 2001

[3] Ben-Or M., Goldreich O., Micali S. and Rivest R.L., *A Fair Protocol for Signing Contracts*, IEEE Transactions on Information Theory, 36(1): 40-46, January 1990

[4] Bock J., *Make Your XML Secure*, .Net magazine, November 2002 issue, http://www.fawcette.com/dotnetmag/2002_11/magazine/columns/security/ (last accessed 04/08/03)

[5] Brown P.W., *Digital Signatures: Are They Legal for Electronic Commerce*, IEEE Communications Magazine, 32(9): 76-80, September 1994

[6] Buyya R. and Vazhkudai S., *Compute Power Market: Towards a Market-oriented Grid*, Proceedings of First IEEE/ACM International Symposium on Cluster computing and the Grid, CCGRID2001, 574-581, May 2001

[7] Chakravarthula S.K.N., *Trading System Design and Implementation in OCEAN*, Master's Thesis, University of Florida, December 2002

[8] Chawla N., *Registration And Authentication Protocol for the OCEAN*, Master's Thesis, University of Florida, August 2002

[9] Chokkareddy C., *Automated Negotiations in OCEAN*, Master's Thesis, University of Florida, August 2002

[10] Dournaee B., *XML Security*, McGraw-Hill/Osborne Media, Berkeley, CA

[11] Eastlake D., *Secure XML*, Addison Wesley, Boston, MA

[12] Frank M.P., *Open Computation Exchange and Auctioning Network*, http://www.cise.ufl.edu/research/ocean/ (last accessed 04/03/03)

[13] Foster I., Kesselman C. and Tuecke S., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, The International Journal of Supercomputer Applications, 15(3), 2001

[14] Govindaramanujam S., Harrison C., Jansen E., Nallan S.K., Singh S., Frank M.P., *Locating Suitable Resources in OCEAN*, October 6, 2002 Paper accepted at the Poster session at High Performance Computing Conference, Bangalore, India,

http://www.cise.ufl.edu/research/ocean/new/documents/papers/paper_v2.pdf (last accessed 04.08/03)

[15] Han W.Y., Park C.S., Young Lim S.Y., Kang J.H., *An XML Digital Signature for Internet e-business Applications*, Info-tech and Info-net, 2001. Proceedings. ICII 2001 - Beijing. 2001 International Conference

[16] Karlinger G., *XML Digital Signatures: An Implementation in Java*, Masters Thesis, Graz University of Technology, Germany, October 2000

[17] Kesterson II H.L., *Digital Signatures-Whom Do You trust?* Aerospace Conference, 1997, Proceedings, IEEE Volume 4: 1-8, February 1997

[18] Mactaggart M., *An Introduction to XML Encryption and XML Signature*, September 2001, http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html (last accessed 04/08/03)

[19] McLaughlin B., *Java & XML*, O'Reilly & Associates, Sebastopol, CA

[20] Micali S., *Computationally Sound Proofs*, In Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Science, 20-22, 436-453, November 1994

[21] Miyazawa T., Kushida T., *An Advanced Internet XML/EDI Model Based on Secure XML Documents*, Seventh International Conference on Parallel and Distrubuted Systems, 295-300, October 2000

[22] Oaks S., *Java Security*, O'Reilly & Associates, Sebastopol, CA

[23] Park H.Y., *Peer List Update Manager Implementation in OCEAN*, Master's Thesis, University of Florida, 2002

[24] Pfleeger, C. P., *Security in Computing*, Prentice-Hall, Englewood Cliffs, NJ

[25] Scheibelhofer K., *Signing XML Documents and the Concept of What You See Is What You Sign*, Masters Thesis, January 2001

[26] Simon E., Madsen P., Adams C., *An Introduction to XML digital Signatures*, August 2001, http://www.xml.com/pub/a/2001/08/08/xmldsig.html (last accessed 04/08/03)

[27] Simpson J.E., *Xpath and Xpointer*, O'Reilly & Associates, Sebastopol, CA

[28] Sun Microsystems, *Java Secure Sockets Extension (JSSE)*, http://java.sun.com/products/jsse (last accessed 04/03/03)

[29] Takase T., Uramoto N., *XML Digital Signature System Independent of Existing Applications*, IEEE Proceedings of the 2002 Symposium on Applicaions and the Internet (SAINT'02w), 150-157

[30] Tobias M.J., *Resource And Requirement Schemas Applied To Auctioning In A Computational Market*, Master's Thesis, University of Florida, December 2001

[31] Wadhwa S.S., *Securing Transactions in OCEAN*, Summary Report, 2002, http://www.cise.ufl.edu/research/ocean/sahib/SecureTransactions.pdf (last accessed 04/08/03)

[32] W3C, *Extensible Markup Language (XML)*, 1997-2000, http://www.w3.org/XML/Schema (last accessed 04/03/03)

[33] W3C, *XML-Signature Syntax and Processing*, http://www.w3.org/TR/xmldsig-core/, W3C Recommendation 12 February 2002 (last accessed 04/03/03)

[34] W3C, *XML Key Management Specification (XKMS2.0)*, http://www.w3.org/TR/xkms2, W3C Working Draft, 18 March 2002 (last accessed 04/03/03)

[35] W3C, *Canonical XML version 1.0*, http://www.w3.org/TR/xml-c14n, W3C Recommendation 15 March 2001 (last accessed 04/03/03)

[36] W3C, *Exclusive XML Canonicalization Version 1.0*, http://www.w3.org/TR/xml-exc-c14n (last accessed 04/03/03)

BIOGRAPHICAL SKETCH

Sahib Singh Wadhwa was born in Ludhiana, Punjab, India on February 3, 1977. He attended Thapar Institute of Engineering and Technology at Patiala, Punjab, India, where he received a Bachelor of Engineering in Computer Science & Engineering in May 1999. In July 1999, he joined HCL Infosystems Ltd. at Noida, India as a Customer Support Engineer. He served HCL Infosystems Ltd. for over one year working at its branches at Secunderabad, Mumbai and Noida. In January 2001, Sahib Singh entered the graduate program in the CISE department at the University of Florida. During his master's research he was a graduate assistant and attended school full time. His research interests include network protocols, steganography and security in distributed systems. His other interests include sports, music, history and poetry.