CryptoNET: Generic Security Framework for Cloud Computing Environments

Abdul Ghafoor Abbasi

Doctoral Dissertation in
Communication Systems
School of Information and Communication Technologies (ICT)
KTH - Stockholm, Sweden, 2011

TRITA - ICT - COS -1102
ISSN: 1653-6347
ISRN: KTH/COS--11/02--SE
ISBN: 978-91-7415-964-6

KTH, School of Information and
Communication Technology
SE-16440 Stockholm
SWEDEN

Academic dissertation for the Degree of Doctor of Philosophy in Communication Systems at Kungliga Tekniska Högskolan to be publicly defended on May 16, 2011 at 13:00 in Sal D, Forum, Isafjordsgatan 39 Kista.

Acknowledgements

The research presented in this doctoral dissertation has been performed at the Royal Institute of Technology, Stockholm, Sweden. Activities towards the completion of this thesis span four years of research and many people have contributed and guided me to the full understanding of the presented work. I wish to express my deep appreciation and gratitude to all of them in my humble acknowledgments.

First and foremost, I offer my sincerest gratitude to my supervisor, Prof. Sead Muftic, who mentored and guided me throughout my research with his patience, expert knowledge and invaluable suggestions. His encouragement and support from the initial to the final step enabled me to develop an understanding of the subject and finish my thesis. One simply could not wish for a better or a friendlier supervisor. I am indebted to him more than he knows.

I gratefully acknowledge support by Dr. Arshad Ali and Dr. Farooq Ahmed for providing me the opportunity to study at the Royal Institute of Technology (KTH). I also wish to express my deep appreciation to all the faculty members and administrative staff of School of ICT and DSV department for their support and valuable assistance in the completion of my thesis.

I am as ever especially indebted to my parents for their love and inseparable support and prayers throughout my life. Words fail me to express my appreciation towards my parents whose dedication, love and persistent confidence in me, has taken the loads off my shoulders. I couldn’t have achieved all this in the absence of their prayers. I simply cannot thank my parents enough. I also wish to thank my brothers and sisters for their continuous support and love during my studies. I would like to thank my wife Nazia Afzal and my son Muhammad Shahwaiz Abbasi for their patience, love, understanding and bearing with my late night hours during my studies.

I also would like to acknowledge National University of Sciences and Technology (NUST), Pakistan, Royal Institute of Technology (KTH), Sweden and Higher Education Commission (HEC), Pakistan for providing financial support for my PhD studies and research publications.

I also feel blessed by having wonderful colleagues, M. Awais Shibli, Chenchen Yuan, and Gernot Schmölzer, Matei Ciobanu Morogan, Feng Zhang, Hao, A. Haseeb, Ammar, M. Qaisar and Adnan Lateef for creating a very friendly environment in Stockholm and providing me with technical help and support wherever I needed. Lastly, I offer my regards and blessings to all my friends and everyone who supported me in any respect during the completion of this thesis.

Abdul Ghafoor Abbasi
Stockholm, Sweden

Dedication

To my parents

M. Riaz Abbasi

and

Shahnaz Akhtar Bibi

Abstract

The area of this research is security in distributed environment such as cloud computing and network applications. Specific focus was design and implementation of high assurance network environment, comprising various secure and security-enhanced applications. “High Assurance” means that

- our system is ***guaranteed*** to be secure,
- it is ***verifiable*** to provide the complete set of security services,
- we prove that it always ***functions correctly***, and
- we justify our claim that it ***can not be compromised*** without user neglect and/or consent.

We do not know of any equivalent research results or even commercial security systems with such properties. Based on that, we claim several significant research and also development contributions to the state–of–art of computer networks security.

In the last two decades there were many activities and contributions to protect data, messages and other resources in computer networks, to provide privacy of users, reliability, availability and integrity of resources, and to provide other security properties for network environments and applications. Governments, international organizations, private companies and individuals are investing a great deal of time, efforts and budgets to install and use various security products and solutions. However, in spite of all these needs, activities, on-going efforts, and all current solutions, it is general belief that the security in today networks and applications is not adequate.

At the moment there are two general approaches to network application’s security. One approach is to enforce *isolation* of users, network resources, and applications. In this category we have solutions like firewalls, intrusion–detection systems, port scanners, spam filters, virus detection and elimination tools, etc. The goal is to protect resources and applications by isolation after their installation in the operational environment. The second approach is to apply methodology, tools and security solutions already in the process of creating network applications. This approach includes methodologies for secure software design, ready–made security modules and libraries, rules for software development process, and formal and strict testing procedures. The goal is to create secure applications even before their operational deployment. Current experience clearly shows that both approaches failed to provide an adequate level of security, where users would be guaranteed to deploy and use secure, reliable and trusted network applications.

Therefore, in the current situation, it is obvious that a new approach and a new thinking towards creating strongly protected and guaranteed secure network environments and applications are needed. Therefore, in our research we have taken an approach completely different from the two mentioned above. Our first principle is to use *cryptographic protection of* ***all*** *application resources.* Based on this principle, in our system data in local files and database tables are encrypted, messages and control parameters are encrypted, and even software modules are encrypted. The principle is that

if **all resources** of an application are always encrypted, i.e. “enveloped in a cryptographic shield”, then

- its software modules are not vulnerable to malware and viruses,
- its data are not vulnerable to illegal reading and theft,
- all messages exchanged in a networking environment are strongly protected, and
- all other resources of an application are also strongly protected.

Thus, we strongly protect applications and their resources **before** they are installed, **after** they are deployed, and also **all the time** during their use.

Furthermore, our methodology to create such systems and to apply total cryptographic protection was based on the *design of security components in the form of generic security objects.* First, each of those objects – data object or functional object, is itself encrypted. If an object is a data object, representing a file, database table, communication message, etc., its encryption means that its data are protected all the time. If an object is a functional object, like cryptographic mechanisms, encapsulation module, etc., this principle means that its code cannot be damaged by malware. Protected functional objects are decrypted only on the fly, before being loaded into main memory for execution. Each of our objects is complete in terms of its content (data objects) and its functionality (functional objects), each supports multiple functional alternatives, they all provide transparent handling of security credentials and management of security attributes, and they are easy to integrate with individual applications. In addition, each object is designed and implemented using well-established security standards and technologies, so the complete system, created as a combination of those objects, is itself compliant with security standards and, therefore, interoperable with exiting security systems.

By applying our methodology, we first designed enabling components for our security system. They are collections of simple and composite objects that also mutually interact in order to provide various security services. The enabling components of our system are: Security Provider, Security Protocols, Generic Security Server, Security SDKs, and Secure Execution Environment. They are all mainly engine components of our security system and they provide the same set of cryptographic and network security services to all other security–enhanced applications.

Furthermore, for our individual security objects and also for larger security systems, in order to prove their structural and functional correctness, we applied *deductive scheme* for verification and validation of security systems. We used the following principle: “*if individual objects are verified and proven to be secure, if their instantiation, combination and operations are secure, and if protocols between them are secure, then the complete system, created from such objects, is also verifiably secure*”. Data and attributes of each object are protected and secure, and they can only be accessed by authenticated and authorized users in a secure way. This means that structural security properties of objects, upon their installation, can be verified. In addition, each object is maintained and manipulated within our secure environment so each object is protected and secure in all its states, even after its closing state, because the original objects are encrypted and their data and states stored in a database or in files are also protected.

Formal validation of our approach and our methodology is performed using *Threat Model*. We analyzed our generic security objects individually and identified various potential threats for their data, attributes, actions, and various states. We also evaluated behavior of each object against potential threats and established that our approach provides better protection than some alternative solutions against various threats mentioned. In addition, we applied threat model to our composite generic security objects and secure network applications and we proved that deductive approach provides better methodology for designing and developing secure network applications. We also quantitatively evaluated the performance of our generic security objects and found that the system developed using our methodology performs cryptographic functions efficiently.

We have also solved some additional important aspects required for the full scope of security services for network applications and cloud environment: manipulation and management of cryptographic keys, execution of encrypted software, and even secure and controlled collaboration of our encrypted applications in cloud computing environments. During our research we have created the set of development tools and also a development methodology which can be used to create cryptographically protected applications. The same resources and tools are also used as a run–time supporting environment for execution of our secure applications. Such total cryptographic protection system for design, development and run–time of secure network applications we call CryptoNET system. CrytpoNET security system is structured in the form of components categorized in three groups: Integrated Secure Workstation, Secure Application Servers, and Security Management Infrastructure Servers. Furthermore, our enabling components provide the same set of security services to all components of the CryptoNET system.

Integrated Secure Workstation is designed and implemented in the form of a collaborative secure environment for users. It protects local IT resources, messages and operations for multiple applications. It comprises four most commonly used PC applications as client components: Secure Station Manager (equivalent to Windows Explorer), Secure E-Mail Client, Secure Web Browser, and Secure Documents Manager. These four client components for their security extensions use functions and credentials of the enabling components in order to provide standard security services (authentication, confidentiality, integrity and access control) and also additional, extended security services, such as transparent handling of certificates, use of smart cards, Strong Authentication protocol, Security Assertion Markup Language (SAML) based Single-Sign-On protocol, secure sessions, and other security functions.

Secure Application Servers are components of our secure network applications: Secure E-Mail Server, Secure Web Server, Secure Library Server, and Secure Software Distribution Server. These servers provide application-specific services to client components. Some of the common security services provided by Secure Application Servers to client components are Single-Sign-On protocol, secure communication, and user authorization. In our system application servers are installed in a domain but it can be installed in a cloud environment as services. Secure Application Servers are designed and implemented using the concept and implementation of the Generic Security Server. It provides extended security functions using our engine components. So by adopting this approach, the same sets of security services are available to each application server.

Security Management Infrastructure Servers provide domain level and infrastructure level services to the components of the CryptoNET architecture. They are standard security servers, known as cloud security infrastructure, deployed as services in our domain level could environment.

CryptoNET system is complete in terms of functions and security services that it provides. It is internally integrated, so that the same cryptographic engines are used by all applications. And finally, it is completely transparent to users – it applies its security services without expecting any special interventions by users. In this thesis, we developed and evaluated secure network applications of our CryptoNET system and applied Threat Model to their validation and analysis. We found that deductive scheme of using our generic security objects is effective for verification and testing of secure, protected and verifiable secure network applications.

Based on all these theoretical research and practical development results, we believe that our CryptoNET system is completely and verifiably secure and, therefore, represents a significant contribution to the current state-of-the-art of computer network security.

Scientific Contributions

The following are research papers published as journal articles or conference paper:

[1] **Abdul Ghafoor**, Sead Muftic, and Gernot Schmölzer, "CryptoNET: Design and Implementation of the Secure E-mail System", published in the Proceeding of International Workshop on Security and Communication Networks (IWSCN-2009), pp.75-80, Trondheim, Norway, **May 20-22, 2009**.

[2] **Abdul Ghafoor,** Sead Muftic, Gernot Schmölzer, "CryptoNET: Secure Federation Protocol and Authorization Policies for SMI", published in the Proceeding of The International Conference on Risks and Security of Internet and Systems 2009 (CRiSIS-2009), pp. 19-26, Toulouse, France, **October 19-22, 2009.**

[3] **Abdul Ghafoor**, Sead Muftic, Gernot Schmölzer, "A Model and Design of a Security Provider for Java Applications" published in the proceeding of The International Conference for Internet Technology and Secured Transactions (ICITST-2009), pp. 794-800, London, UK, **November 9-12, 2009**.

[4] **Abdul Ghafoor**, Sead Muftic, "CryptoNET: Integrated Secure Workstation", published in the International Journal of Advanced Science and Technology, (IJAST), SERSC, pp. 1-10, Vol 12, **November 2009**.

[5] **Abdul Ghafoor**, Sead Muftic "CryptoNET: Software Protection and Secure Execution Environment", published in the International Journal of Computer Science and Network Security (IJCSNS), pp. 19-26, Vol 10, **February 2010**.

[6] **Abdul Ghafoor**, Sead Muftic, "Web Contents Protection, Secure Execution and Authorized Distribution", published in the Proceeding of The 5th IEEE International Multi-Conference on Computing in the Global Information Technology (ICCGI-2010), pp. 157-162, Valencia, Spain, **September 20-25, 2010**.

[7] **Abdul Ghafoor**, Sead Muftic, and Gernot Schmölzer, "CryptoNET: A Model of Generic Security Provider"', published in the International Journal of Internet Technology and Secured Transactions, Vol. 2, Nos. 3/4, pp.321–335, **November 2010**.

[8] **Abdul Ghafoor**, Sead Muftic, "CryptoNET: Security Management Protocols", published in the Proceeding of The 9th WSEAS International Conference on Data Networks, Communication, Computers (DNCOCO-2010), Faro, Portugal, pp. 15-20, **November 3-5, 2010**.

[9] **Abdul Ghafoor**, Sead Muftic, Shahzad Ahmed Mumtaz, "Security Extensions of Windows Environment based on FIPS 201 (PIV) Smart Card", published in the Proceeding of The World Congress of Internet Security, London, UK, pp. 100-106 **February 21-23, 2011**.

Table of Contents

Part I: Generic Components and Their Validation

Part II: Security Architecture and Applications

Part III: Overview of Significant Contributions and Future Research

Table of Figures

List of Tables

1. Introduction

In this chapter we give a brief introduction to security principles, concepts and issues for network applications and cloud computing environments. We highlight motivations for scientific objectives of our research by emphasizing deficiencies in current approach and by describing our security framework for cloud computing environments and network applications based on standard security technologies and protocols.

Our primary focus is protection of IT resources, operations and communications in network applications based on innovative and new approach to security. Our framework is based on a simple principle that all resources and operations of network applications are maintained in cryptographically protected form and used in a cryptographically protected environment.

1.1. Concept and Principles

It is general opinion today that security for network applications is their very important feature and property. Its scope includes protection of data, messages, software modules and other resources, privacy of users, reliability, availability and integrity of resources and other properties. In the last 20 – 25 years there are many contributions in the area of computer networks security: standards, research projects, conference and journal papers and commercial products. Governments, companies, banks and other users of network services invest great deal of time, effort and budgets installing and using various security products and solutions.

However, in spite of all these activities, on-going efforts and current solutions, it is general belief that security in today networks and applications is not adequate. We are daily witnessing various problems – infection of computers by malware, distribution of E–mail spam, phishing of Web pages, penetrations by hackers, software bugs, stolen industrial secrets and credit cards, disclosure of sensitive documents, and so on.

All these interests and current problematic situations justify efforts and activities towards creating effective security solutions for network applications and environments. At the moment, there are two general approaches to network applications security:

- One approach is based on *isolation*, that is protecting them by isolating operational environments at their periphery using firewalls, port scanners, intrusion–detection tools, spam and phishing filters, “demilitarized zones”, E-mail spam filters, etc. and also using virus/malware scanners, virus signatures, encrypted disk files, etc.
- Another approach is called *software security* which is based on methodology to create secure, robust and protected applications, bug–free and not vulnerable to attacks, by using well–established methodology for design of applications, software tools for their development, and testing methodology and environments for their debugging and testing.

Although both approaches give some degree of security and protection, current situation in open networks indicates that in principle none of the current approaches is effective and does not produce secure, reliable and protected network applications and cloud environments. This means that current, mainly single point–solutions and approaches, reactive to emerging problems, have limited scope and effectiveness. This indicates that the two current approaches, one based on solving individual problems ("point–solutions") in a reactive mode and the other based on conceptual methodology for design and development of secure software, so far have not produced satisfactory results and are not capable to create the ultimate solution – secure and reliable network environment and its applications.

Therefore, in the current situation, new approach and new thinking towards creating strongly and guaranteed secure network environments and applications are needed. This new approach was the focus and the concept of our research. The background, motivation and the essence of our new approach is the following:

Most of secure applications today were usually developed first with their basic functionality, and security was added later, if at all, as an add–on extension or as additional, optional feature. If some already developed and operational application is to be enhanced with security, then the usual approach today is to invoke application programming interfaces (APIs) of some crypto library [1] [2] or use some, so called, crypto services provider [3][4]. However, security tools and libraries today are not broadly available, sometimes not fully functional, and usually very complicated to use. Furthermore, in this process security functions are usually applied only to resources and functions of a specific application. In addition, if an application offers some security services, then end-user has to configure various options and parameters prior to use of these security services. The procedures for that are usually inconvenient, especially for non-technical users. Finally, those applications are protected by additional external modules, like firewalls and virus scanners, after their installation and deployment.

Such add–on security extensions of applications, analysis of consequences and damages after network penetrations, recovery after destruction of resources, analysis of vulnerabilities of software modules for infection and other "post–factum" methods so far have shown their weaknesses and inadequate protection effects.

Therefore, in our research we have taken completely different approach. Our first principle is *to apply cryptography to protection of all application resources*: data in local files and database tables, messages, control parameters, and even software. The principle is that if an application is completely encrypted, "enveloped in a cryptographic shield", then (a) its software modules are not vulnerable to malware and viruses, (b) its data are not vulnerable to illegal reading and theft, (c) its messages exchanged in a networking environment are strongly protected, and (d) all other resources of an application are also strongly protected. Thus, we protect applications and their resources before they are installed and deployed.

We have also solved some additional important aspects of this approach: manipulation and management of cryptographic keys, execution of encrypted software, and even collaboration of encrypted applications in a distributed environment and cloud computing environment. During our research we have created a set of development tools and also development methodology which can be used to create such cryptographically protected applications. The same resources and tools are also used as the run–time supporting environment for execution of secure applications. Such total cryptographic protection system for design, development and run–time support of secure network applications we call ***CryptoNET*** system.

CryptoNET is an integrated secure collaborative environment comprising the most popular standalone and distributed applications and associated security protocols. We have created several client components, such as Secure Station Manager (equivalent to Windows Explorer), Secure E-Mail Client, Secure Documents Manager (security extensions of Open Office), and Secure Web Browser. In addition to those workstation components, we also designed and implemented corresponding servers: Secure E-Mail (SEM) Server, Secure Library Server, Secure Web Server and Secure Software Distribution Server. Security protocols between clients and servers are Strong Authentication, SAML–based Single-Sign-On, Secure Sessions, and some application–specific security protocols. All our applications and security protocols use functions and credentials of our single Generic Security Provider, which also transparently uses FIPS 201 Personal Identity Verification (PIV) [5] smart cards, if they are configured and attached. The components of our CryptoNET environment may also be connected to our cloud security infrastructure, so standard network security protocols, such as certification protocol [6], SAML authorization protocol, secure sessions, etc., are also supported in a large-scale network environment.

Our CryptoNET system is **complete** in terms of functions and security services that it provides, it is **integrated** so that the same cryptographic engines are used by all applications, and finally, it is completely **transparent** to users – it applies its security services without expecting any special interventions by users. With all the principles and resources used for design of applications and their components, CryptoNET can also be used as a practical environment for future research, design and development of a generic security framework, what represents major goal of our research.

1.2. Research Focus

Our research focus was to design a security system for network applications and cloud environments that will completely protect their resources, attributes, messages, software modules and execution environment against various attacks. Some of the most common attacks are described in Appendix A.

Our methodology to create complex security systems and to apply total cryptographic protection is based on the design of security components in the form of generic security objects. Those objects can be data object, functional object or composite object. Each object is completely secure means its data attributes are protected, its functions can only be accessible to authenticated and authorized users, and its executable binaries are also protected. Furthermore, for our individual security objects and larger security systems, in order to prove their structural and functional correctness, we apply deductive scheme for verification and validation of security systems. So, verification of complex security systems is based on the principle that is: if individual objects are verified and proven to be secure, if their instantiation, combination and operations are secure, and if protocols between them are secure, then the complete system, created from such objects, is also verifiably secure.

Our security system, CryptoNET, is based on our generic security objects, well-established secure technologies and security standards. Therefore, the components designed based on this methodology will also be generic and compliant with security standards. The core components of our security system are Security Provider, Secure Execution Environment, and Security Protocols. They contain security engines of our security system, where each component provides the same set of tested security services. These components are complete with respect to their functionality, so developers can use these components to extend their applications with security features.

1.2.1. Enabling Components

Security Provider provides security services to various components. The Provider is designed using the concept of generic security objects. Each generic object encapsulates security functions and attributes of some security service. The Provider transparently handles security credentials and hardware tokens, which are easy to integrate with other components.

Security protocols component comprises various network security protocols that provide authentication, authorization, secure communication, and identity verification services. These protocols are based on generic security objects, security standards and well-established security technologies. Some of protocols are FIPS 196 strong authentication, Single-Sign-On, SAML authorization, and secure sessions protocol. These protocols also use Security Provider for various software-based or FIPS 201 (PIV) smart cards-based cryptographic functions.

Generic Security Server is another complex object which provides core components for implementation of Secure Application Servers supporting standard and extended security functions. All security functions are based on well-established security standards, technologies and protocols. Furthermore, several components, actions and libraries are available in this template in the form of Eclipse plug-ins in order to provide easy management of Secure Application Server, extendable with customized security functions, and several implemented actions for administration.

In order to use our secure applications, we also designed extended secure execution environment. It uses cryptographic services provided by the Security Provider. The extended environment executes encrypted and verifiable components in a cryptographically protected run-time environment. In addition, we provide the solution to generate and encapsulate protected software modules. The extended secure execution environment supports various network security protocols which are designed as a security protocols component of our security system.

1.2.2. Secure Network Applications

Based on above four enabling technologies, we also designed several secure network applications. Some of them are client components and some are application servers. Client components are part of our Integrated Secure Workstation (ISW). The ISW uses single Security Provider which provides extended security functions and features. Client components provide application-specific security functions and features. This approach provides the same set of cryptographic functions and security protocols across multiple applications. ISW may also be connected to various servers of our cloud security infrastructure, so it supports security protocols, certification protocols, SAML protocols, strong authentication protocol, etc.

Application servers are components of our network applications. These are Secure E-Mail Server, Secure Mail Infrastructure (SMI) Server, Secure Web Server, Secure Library Server, and Secure Software Distribution Server. These servers are deployed in a domain as services in order to provide application-specific services to the client components. For example, Secure E-Mail Server provides secure communication with Secure E-Mail Client, handling of secure Emails, secure management of address books and cryptographic keys, and confirmation messages. Application servers provide security functions and secure communications using our designed engine components, so by adopting this approach, the same set of security services are available with each of the four servers.

1.3. Research Methodology

Research methodology is an activity which provides a systematic process to conduct research in various disciplines. It defines the procedures for data collection, investigation, analysis, and interpretation. In this research activity, we adopted Design Science research methodology which is more suitable for the artifact development scientific research projects [103]. This research methodology follows certain steps in order to make scientific contributions to information sciences. These steps are: problems identification, design, artifact development, and then evaluation. In the problem identification phase, we critically analyzed current security systems, solutions, methodologies and techniques and then formulated our hypothesis. In the design phase, we used the concepts of generic security objects and component-based software engineering methodology to define the core and application-level generic objects and components of our security system. To test the hypothesis, we developed cloud security infrastructure and network applications based on engine components. In the evaluation phase, we used attack resistant techniques and quantitative analysis to evaluate the performance of our generic security objects.

1.4. Overview of Research Contributions and Results

This section presents the summary of our research contributions and results.

1.4.1. Enabling Components

Generic Security Provider

Generic Security Provider is an engine component in our system. It provides a comprehensive set of security services, mechanisms, encapsulation methods, and security protocols for other components of our security system and for secure applications. The Provider is structured in four layers; each layer provides services to the upper layer and the top layer provides services to applications. Security services reflect requirements derived from a wide range of applications: from small desktop applications to large distributed enterprise environments. Starting from an abstract model, we describe design and implementation of an instance of the provider comprising various generic security modules: symmetric key cryptography, asymmetric key cryptography, hashing, encapsulation, certificates management, creation and verification of signatures, and various network security protocols. We describe the properties, extensibility, flexibility, abstraction, and compatibility of the Security Provider which is implemented using Java (see Chapter 4).

Generic Security Protocols

Generic Security Protocols play an important role for implementing security services in distributed applications and cloud computing environments. In this contribution, we designed several security protocols. They are based on the concepts of generic security objects and on a modular approach. The objects of security protocols are complete in terms of their functionality, so each object provides features to client and server applications. The protocols are designed using on well-established secure technologies and standards in order to make their host components interoperable with other components. Some of our authentication protocols are specifically designed for a specific operating system, while other protocols are platform independent and generic. Therefore, they can be integrated with any application for secure communication, authorization, key distribution, Single-Sign-On and strong authentication. These protocols are based on our Generic Security Provider in order to perform cryptographic functions and communications with smart cards. In addition, these protocols are generic what makes them easy to use by developers for building secure cloud computing applications. The complete description of Generic Security Protocols is described in Chapter 5.

Generic Security Server

Generic Security Server is also engine component which provides basic structure to implement secure application servers (see Chapter 6). It is designed as a template which provides complete set of standard security and administration functions along with a number of extended security functions and features. These functions are based on well-established security standards and services. It provides basic structure for developers in order to develop customized Secure Application Servers. We already implemented several initialization and management functions and several administrative actions. We also included APIs and libraries for cryptographic functions and security protocols in order to provide the same set of security services for all instances of Secure Application Servers. The structure of our Generic Security Server is flexible and it is available in the form of Eclipse-plug-ins, which is easy to extend according to customization requirements of each application.

Secure SDK

Secure SDK is a set of various security components which are protected using strong encryption techniques (see Chapter 7). For protection of software modules, we designed a solution using strong encryption techniques. This solution comprises Secure Software Distribution Server and Web Server in order to generate and distribute protected software modules only to authorized users. Our solution encapsulates these modules in the form of specially designed eXtensible Markup Language (XML) file which represents general syntax of protected software modules. Secure SDK and encapsulation of software modules are based on well-established secure technologies and standards, like FIPS 201 (PIV) smart cards, FIPS 196 strong authentication, and authorization policies based on eXtensible Access Control Markup Language (XACML).

Secure Execution Environment

Secure Execution Environment is also key component of our system. It executes protected software modules in controlled environment. In our design, all software modules and all other components of the CryptoNET are encrypted in order to protect them against reverse engineering, illegal tempering, program-based attacks, BORE (Break-Once-Run-Everywhere) attack, and unauthorized use of software. We extended standard execution environment with special security features and functions. Our extended Secure Execution Environment supports standard security services and network security protocols. These are: transparent handling of certificates, use of FIPS-201 compliant smart cards, Single-Sign-On protocol, strong authentication protocol, and secure sessions.

1.4.2. Integrated Client for Secure Applications

In the most of current applications security is usually provided individually. This means that various applications use their own security mechanisms and services, applied only to their own resources and functions. Furthermore, procedures to configure security parameters are usually inconvenient and complicated for non-technical users. As an alternative to this approach, we have designed and implemented Secure Workstation, which represents an integrated security environment and protects local IT resources, messages and operations across multiple applications. It comprises several components, i.e. four most commonly used PC applications: Secure Station Manager (equivalent to Windows Explorer), Secure E-Mail Client, Secure Documents System, and Secure Web Browser. These four components for their security extensions use functions and credentials of our enabling components, Security Provider and Security Protocols. With this approach, we provide standard security services (authentication, confidentiality, integrity and access control) and also additional, extended security services, such as transparent handling of certificates, use of smart cards, strong authentication protocol, SAML-based Single-Sign-On, secure sessions, and other security functions, to all applications with the same set of security modules and parameters.

1.4.3. Secure Network Applications

CryptoNET: Secure E-mail System

This section describes the design and implementation of a secure, high assurance and very reliable E-mail system. The system handles standard E-mail security services – signing and encryption of E-mail letters and, in addition, provides a number of extended and innovative security features. These new features are: transparent handling of certificates, strong authentication between Secure E-Mail Client and Secure E-Mail Server, archiving and recovery of encrypted address books, simple and secure handling of cryptographic keys, security sessions management, tracking of E-mail letters using confirmation messages, elimination of spam messages, prevention of fraudulent and infected attachments, and usage of smart cards. The system is based on the concepts of proxy architecture that makes it compatible with existing E-mail infrastructure. We also used XACML–based authorization policies at the sending and receiving Secure E-Mail (SEM) Servers in order to provide complete protection against spam. In our system, these policies are enforced by the Policy Enforcement Point (PEP), a component of the SEM server. In order to interconnect Secure E-mail systems deployed in individual domains, we introduced new infrastructure-level servers in order to develop trust between domains, exchange SEM registration information, and certify and verify domain names.

CryptoNET: Secure Web System

Our Secure Web System represents the design and implementation of a comprehensive system for protection of Web content stored at Web Servers, for execution of protected Web pages, and for their distribution to authorized users. We introduced additional components and added extended security features to a standard Web Server in order to provide confidentiality and integrity of Web content. We also designed and implemented an extended secure execution environment for Java Web Server, which is capable to process and execute different types of encrypted and digitally signed Web pages encapsulated in *PKCS7SignedAndEnvelopedData* format. This system follows component-based architecture what makes it compatible with the exiting Web infrastructure.

CryptoNET: Secure Documents System

We designed a Secure Documents System in order to protect documents in local and collaborative environment. Our Secure Documents System comprises a set of security functions, features and components functioning as security extensions of the Open Office. The extended security features are: protection of documents in local environments, distribution of secure documents to group members, group key management, enforcement of section level XACML policies for access control, smart card-based cryptographic functions, and transparent handling of security credentials. The design of the system is based on our generic security objects and plug-in architecture, what makes it easy to extend and integrate with existing document systems. In addition, Secure Documents System is linked to our cloud security infrastructure which provides security services in global environments by using certificates and SAML technologies.

1.4.4. Cloud Security Infrastructure

Cloud security infrastructure is an environment in which several standard security components are deployed as services. These components are: Local Certification

Authority, Policy PKI Server, Top PKI Server, Identity Management System (IDMS), XACML Policy Server, and Strong Authentication Server.

1.5. Organization of The Thesis

This thesis is organized in four parts, each part comprises several chapters. Part I comprises two chapters, Part II comprises six chapters, Part III comprises five chapters, and the last part has one chapter.

Chapter 2 describes security requirements and limitations of various applications by highlighting the existing research efforts and gaps in existing solutions. Based on Chapter 2, in Chapter 3 we describe our formal approach for designing generic security objects. We also describe the structure and various types of generic security objects. In Chapter 4 we describe the model and design of Security Provider, a key component of our framework. In Chapter 5 we describe various security protocols for authentication, authorization, secure communication, and key distribution. In Chapter 6 we describe our design of a Generic Security Server template for implementing customized secure application servers with extended security features. In Chapter 7 we explain the design of software modules protection and their execution in a secure execution environment. In Chapter 8 we evaluated and validated individual generic security objects and our formal approach using threat model.

Using our enabling components, we designed several network applications described in Part III. In Chapter 9 we describe the design of our CryptoNET System and its layered model. We described Integrated Secure Workstation in Chapter 10, Secure E-mail System in Chapter 11, Secure Web System in Chapter 12, and Secure Documents System in Chapter 13.

In Chapter 14 we give conclusions of our research and describe significance of our contributions.

1.6. Summary

Currently two main approaches are used for network applications security. One is isolation and the second is software security. These techniques do not provide an adequate level of security against ever increasing threats. Contrary to these approaches, we designed and implemented a security system for network applications which cryptographically protects and manipulates IT resources, attributes, messages, software modules, and the overall execution environment. With this approach, we believe that if all the components and environments are shielded with cryptographic protections, then the overall system will be secure. The design and implementation of our security system is based on generic security objects and components, based on well-established security standards and technologies. We designed some core security components in order to provide the same set of security functions to other components of the system. Our components are complete in terms of their functionality, they are easy to integrate with various applications, and they transparently handle all their configurations and security credentials.

2. Related Work

In this chapter we overview and analyze existing security solutions, products and architectures currently available for protection of resources and messages in network applications. Based on the analysis, we identify the shortcomings and problems with existing solutions.

2.1. Overview and Analysis of Existing Solutions

In this section we analyze existing security systems, solutions and products. We structured this section into subsections based on the components of our security system, explained in Chapter 1. We also analyze security functions and features of various network applications available in our CryptoNET system.

2.1.1. Security Providers (Libraries and Middleware)

Security providers are usually crypto libraries or middleware modules exporting their functionality through the set of Application Programming Interfaces (APIs).

Generic Security Services Application Programming Interface (GSS-API)

One of the first efforts to standardize cryptographic security platform was Generic Security Services Application Programming Interface (GSS-API) standard [7]. GSS-API itself does not provide security services, it is only a framework that offers security services to callers in a generic fashion, supported by a range of underlying mechanisms and technologies, such as Kerberos or public key cryptography.

Microsoft Security Support Provider Interface (SSPI)

One of the implementations of GSS-API was by Microsoft in the form of Security Support Provider Interface (SSPI) [8]. SSPI is a set of interfaces between transport level applications and network security service providers and it is commonly known as Security Support Provider (SSP) or Cryptographic Service Provider (CSP) [9][10]. CSP is collection of providers: Microsoft Base Cryptographic Provider, Microsoft Enhanced Cryptographic Provider, Microsoft DSS Cryptographic Provider, Microsoft Base DSS and Difie-Hellman Cryptographic Provider, and Schannel Cryptographic Provider. Some of these providers are available with Windows 2000 and later versions and some enhanced are only available at selected locations due to export restriction policies. All these providers are proprietary solutions, so they can not be used in open source projects and even for extensibility. Furthermore, Microsoft CSP is platform-dependent provider and is digitally signed only by Microsoft.

Java Security Architecture

Sun Microsystems developed its own security provider [11]. Initially, its purpose was to sign and verify Java applets. Later, Sun Microsystems introduced Java Extensible Security Architecture (JSA) based on a set of Application Programming Interfaces (APIs), tools and security mechanisms. It includes a set of frameworks to provide security services to Java applications. These frameworks are Java Cryptography Architecture, Java Cryptographic Extension, Java Certification Path, Java Authentication and Authorization Services, and Java Secure Socket Extensions. Design of JSA framework is generic and extendable, so some third parties, like IBMJSSE [12], IAIK-JCE [13], J/Crypto [14], VIA-JCP [15], OC4J [16] implemented these interfaces with extended security features. Moreover, JSA provides cryptographic services transparently to applications by invoking underlying available security provider(s). JSA follows incremental and replaceable component model in order to add other security providers.

CrypTool

CrypTool is open source crypto library [17]. The aim of this project is to provide a platform for e-learning of cryptography and cryptanalysis in a modular and easy-to-use way. Currently, the team of CrypTool is working on JCrypTool and CrypTool 2.0. CrypTool 2.0 is based on C++ programming language, while JCrypTool is based on Java and Eclipse. JCrypTool provides security features using modular plug-in approach. It is structured in the form of plug-ins, which are structured based on their functionality e.g. logging, core engine, data objects, etc. The objective of this separation is to provide flexibility and extensibility to end-users.

Other Providers

Some commercial companies developed also their own security providers, but they are mainly client-oriented, like NSP’s (Network Security Provider) [18]. NSP adopted a layered approach in order to protect networks from viruses, worms and intruders. Similarly, Entrust Entelligence® Security Provider uses digital identities to clearly identify users, machines to access network, and other resources, either local or through VPN [19]. This solution provides encryption and digital signing cryptographic techniques in order to protect access to sensitive proprietary information either stored locally or in transit.

Analysis of Security Providers

Existing security providers are very limited in functionality and usage, because they were developed using conventional programming approach. These are modeled for specific tools and technologies. Most of them are very complicated and difficult for developers to use. Furthermore, categorization of cryptographic services was not addressed according to the functionality of each service, using modular plug-in approach. Based on above shortcomings, we modeled and designed our Generic Security Provider which is generic, modular, easy to understand and complete in terms of functionality.

2.1.2. Security Management Protocols

Client-server paradigm is widely used in distributed applications. Various security protocols for authentication, authorization and secure communications have been designed and developed. For authentication, the most popular protocol is Password Authentication Protocol. This protocol is based on a use of a secret password, which is known only by the end-user and the server. This protocol is considered weak protocol, because a password can be cracked by using various techniques, for example, guessing password, dictionary attack, and brute force attack. A comparatively secure protocol is Challenge Handshake Authentication Protocol. In this protocol, random numbers are exchanged between a client and a server for authentication. This protocol is also not secure, because it does not provide source authentication and replay attack can be used for impersonation. A modified version of this protocol uses asymmetric-key cryptographic functions. Secure Shell (SSH) and Strong Authentication protocol are examples of such protocols. SSH uses self generated asymmetric keys, while Strong Authentication is based on X.509 certificates. Another authentication protocol is Extensible Authentication Protocol (EAP), described in RFC 3748 [36]. EAP is used for authentication of wireless LANs and most of operating systems support it. The extended version of EAP is Protected Extensible Authentication Protocol (PEAP) [37], which is based on TLS for certificates-based authentication and secure communications.

Secure Socket Layer (SSL) protocol is widely deployed in most of commercial products for secure communications between clients and servers. SSL uses X.509 certificates and hash functions for confidentiality and integrity of messages. Most of companies integrated this protocol in their products with authentication and authorization protocols. OpenSSL [1] is one of them. This library provides the set of cryptographic functions and security protocols. OpenSSL is an open source implementation of SSL, which can be used with other applications for secure communications. Another product is eToken [38]. It provides USB smart-card-based strong user authentication and password management for enterprises. This solution is compliant with industry regulations and internal security policies. This solution also provides SSO services where a user can store more than one account information in a single token.

Lexar® JumpDrive SAFE S3000 [39] is another products which provides hardware based encryption and smart card based authentication for multiple operating systems. In this solution, smart card securely generates and stores cryptographic keys which is eventually used to encrypt and decrypt user’s stored data.

ASECard for Windows Smart Card Starter Kit [40] is smart card-based solution for Windows Login. This product provides various security services to applications like MS Outlook, IIS SSL, and Adobe Acrobat. AuthLite [41] uses USB as a storage device to store security credentials. It provides strong account protection without any specific driver and hardware.

The SafesITe PIV client module installed on user machine securely performs strong authentication, encryption, decryption, and generates smart card-based digital signature for application data. Gemalto in collaboration with IBM also developed solution for web based Single-Sign-On protocol based on smart cards for physical and logical access control. This product supports public key cryptography and is fully compliant with FIPS-201 and Europe Identification Authentication Signature standards.

Smart Card Alliance [42] is an organization which provides platform to different member's organizations for smart card manufacturing, middleware development and smart card-based applications. The core objective of Smart Card Alliance is to promote smart card technologies for identification, payment and other user applications to ensure user privacy, data security and integrity.

Analysis of Security Management Protocols

By reviewing various existing solutions and commercial products, we established that most of security products are designed using some proprietary technology. Therefore, these products cannot be extended with new security features. We also found that various solutions support security functions applicable only to specific resources of individual applications. In addition, these applications need intervention of an end-user for configuration and integration of hardware tokens and security credentials. Most of them are based on some proprietary technology, so each product uses a different type of personalized smart cards for security functions and protocols, which cannot be integrated with other applications to extend them with security functions.

2.1.3. Generic Security Server

Currently available most important APIs and libraries for rapid development of application servers are available in the form of Eclipse plug-ins. They provide basic structure and functions like start server, strop server, publishing, targeting projects, adding and removing modules, etc. This is a generic framework, so new servers can also be added. It further provides servers' view, wizards, editor framework, etc. Some of the core packages of Eclipse generic server are *org.eclipse.jst.server.generic.core*, *org.eclipse.jst.server.generic.serverdefinition* and *org.eclipse.jst.server.generic.ui*. For implementing security services, application developers use third party Security Provider, APIs and security libraries. Another concept of generic application server is described in [102]. It addressed scalability issues and provides solutions to handle multiple clients and requests. Therefore, it provides session management and client authentication. Each default message contains a header and a block of request specific data. The header includes packet size and a request type identifier, while data contains the actual information. In addition, the header may include security information, such as an authentication token, checksum, etc.

IBM provided the concept of a generic server which is managed in the WebSphere Application Servers administrative domain. WebSphere Application Server provides features to define a generic server as an instance of application server within the WebSphere Application Server administration domain.

In current situation, the most important deployed generalized server is Web server [55]. It is a container for Web objects which are accessible to clients using HTTP protocol. Web server provides various services to Web modules. It provides transportation facilities, SSL based security features, basic access control services, and local and remote administration.

Another application server is Lotus Domino [90]. It is a generalized server, but it expands services horizontally. Lotus Domino provides Web services, mail and newsgroup services. It supports Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP3), Internet Message Access Protocol (IMAP), Network News Transfer Protocol (NNTP), Lightweight Directory Access Protocol (LDAP) and Domino Internet Inter-ORB Protocol (DIIOP). This server provides access control using Access Control List (ACL). This mechanism determines whether or not users have access to the database and which levels of permissions are granted to users. This server provides transport level security services to all the components using SSL protocol. Furthermore, it provides username and password-based authentication.

Analysis of Generic Security Server

We analyzed some of general most popular servers and their behaviors. We found that some of existing products use third party security components to extend them with security services. They are very complicated to integrate and it is inconvenient to configure their security parameters. If security is already implemented, then it is only applied to individual resources of specific resources. In addition, we analyzed that they follow different standards, methodologies and proprietary solutions. Therefore, they may not be interoperable in global environments with other solutions. We also found that existing APIs and libraries provide only basic structure for implementing application servers. They do not provide certain security management functions and features, because most of developers give more importance to functional requirements, while less importance is given to security functions. Therefore, current implementations do not provide certain security libraries in order to implement extended security functions.

2.1.4. Secure SDK and Execution Environment

In this section we overview and analyze security functions and features of some existing products, applications, proposed solutions, and industry software protection standards. Most of the software protection solutions can not effectively combat major attacks. We structured software protection systems in three aspects and analyzed security functions and requirements of each approach. These aspects are: (a) protection of software modules, (b) secure software distribution, and (c) controlled execution environment.

Protection of Software Modules

Software Protection Initiatives (SPI) group [20] initiated a process to develop strategies and technologies to protect sensitive code, like engineering, scientific, modeling and simulation software. SPI focused on availability, authentication, confidentiality, integrity and non-repudiation services in order to protect value-added software. One of the first solutions against reverse engineering and illegal modifications of software executable modules was explained by Kent in 1980 [21]. Kent defined both cryptographic and physical temper-resistance techniques for software protection. Obfuscation is another technique, which automatically transforms the original code into equivalent obfuscated code, discussed in [22] [23] [24]. In the early 1990s this technique was used to protect software from viruses, but with some modifications it is being used to protect binary code from reverse engineering and illegal modifications. This technique does not require special execution environment on a host platform.

Currently, the most important method for protection of software modules is verification of software against viruses. Some solutions, like [25], provide protection of software modules using asymmetric cryptography. This approach allocates Cryptographic Function Area (CFA) to store private key and software encryption key. The binaries server generates software encryption key which is seeded by the fingerprints (the identity of a host machine). Similarly, UltraProtect [26] uses asymmetric key to protect software executables against piracy and illegal distribution. A hybrid software protection technique, described in [27] [28], protects software modules against reverse engineering. This technique embeds a plaintext decryptor in an encrypted program, but the plaintext decryptor is obfuscated using code obfuscation technique. The role of descriptor is to decrypt executable binaries.

Secure Software Distribution

Currently, open source and free software distribution community is only concerned with integrity of software modules. With this approach, software owner generates hash value of executable modules and uploads it to Internet with static hash value [29]. Client downloads software and generates its hash value to compare with the published hash value for integrity assurance. This mechanism ensures the integrity of software guaranteeing that it was not altered during downloading phase. Similarly, vendors of commercial products may sign software modules which are verified by the client during the installation phase [30]. These two techniques do not provide integrity or resistance against software tempering of executable modules after deployment phase. Software distribution technique explained in “Secure Code Distribution” [31] verifies integrity of software (Applets) after downloading and verifies signature before execution. Applet developer signs the code using private key which is verified by the secure class loader embedded in the JVM. Similarly, J2ME based applications for mobile phones and Point of Sale (PoS) applications for PoS devices must be signed before loading into devices. The run-time environment of devices verifies applications in the installation phase. Furthermore, the paper [31] also mentioned that S/MIME can be used to securely distribute software.

Controlled Execution Environment

Currently, a well known software execution environment is Java Virtual Machine which verifies signed Java applets before execution. Trusted Computing Group [32] provided hardware-based solution, known as Trusted Platform Module, which is a combination of different components to protect local resources, like files, software modules, keys, etc. Another hardware-based secure execution environment is described in [33] which use cryptographic functions in a low cost memory chip. Microsoft is working on the concept of “Next-Generation Secure Computing Base (NGSCB)” [34] which relies on hardware technology to provide a number of security-related features, like fast random number generation, secure cryptographic co-processor, and the ability to protect cryptographic keys to make them impossible to retrieve. The goal of this approach is to execute software in a secure environment. Apple is also working on incorporating a Trusted Platform Module (TPM) into their Apple Macintosh line of computers for the integrity and confidentiality of software modules [35].

Analysis of Secure Execution Environments

By reviewing various existing software protection and secure execution environments, we established that most of techniques are only used to ensure the integrity of software modules. Execution environments can only verify it before execution. Current solutions do not protect software modules using strong encryption techniques against reverse engineering and BORE attacks. Furthermore, middleware platforms do not support execution of protected software modules, enveloped in a standard cryptographic format. We also found that the current software protection standards and techniques are focused on security services and functions, but we did not find any standard to describe the format of protected software modules.

2.1.5. Integrated Secure Workstation

In this section we analyze security features and principles of some of the most popular and widely used PC applications. With respect to security, we classify various PC applications in three groups:

- Security Applications that provide protection of PC resources against intruders, malicious code, theft, destruction, etc. Popular such applications are McAfee [43], Norton [44] or Symantec [45];
- Proprietary products, open source or commercial, that provide mainly encryption and/or access control to local resources. Examples of such products are eCryptf [46], Ubuntu File Browser [47], AxCrypt [48] or Crypt Manager [49];
- Standard PC applications, available on every desktop, with some security extensions: Web browsers (with SSL), E-mail clients (with S/MIME), and applications handling files and documents (with possibilities for encryption or creation of digital signatures). Examples are security extensions of E-mail clients to send/receive signed/encrypted E-mails, SSL for browsers, or digital signing of PDF documents in Adobe Acrobat.

Security Applications

End-user installs anti-virus software (security applications) at a workstation to protect PCs from viruses, worms and malicious code. Examples are McAfee [43], Norton [44] and Symantec [45]. These tools use signatures or pattern-based database to detect malicious code. In order to effectively detect ever increasing threats, signature or patterns database must be updated regularly.

Protection of Local Resources

File or directory encryption functions, if available in file browsers, use symmetric key cryptography. These applications store symmetric keys either in the same folder or file they protect or in a separate encrypted private directory [50]. Some commercial products, like McAfee and Symantec, provide Endpoint Encryption Suites, which automatically encrypt files and devices using AES-256 symmetric key algorithm. In addition, this type of products sometimes also provides local access control and key management functions for sharing information in collaborative environments. Another example, eCryptfs [46], provides security features like encryption of files, key management and access policies. This product stores cryptographic metadata in the header of each file, so that encrypted files can be copied between hosts without keeping track of the cryptographic keys. In general, currently available commercial and open-source products do not provide strong and comprehensive security using advanced security functions, such as asymmetric key cryptography, support of certificates, cryptographic encapsulation technique (PKCS#7), or strong authentication protocol.

Security of Standard PC applications

File Browser, E-mail client, Web Browser and Document Management software are standard PC applications. Most of E-mail clients, like MS Outlook, Eudora or Thunderbird provide end-to-end security for E–mail letters using S/MIME. These applications do not provide enhanced security features, like protection of their address books, key-management, transparent handling of certificates, use of smart cards, strong authentication protocol, Single-Sign-On, and protection against spam. Thus, E–mail is usually used to transfer malicious content, spam, viruses, etc.

Browsers are another application with serious security weaknesses and privacy threats. Current browsers do not protect browsing history, cookies, passwords, and data filled in Web forms. Furthermore, some browsers automatically download ActiveX controls from Web servers [51], which are major source of vulnerabilities, viruses and worms. Eavesdropping, man-in-the-middle, spyware, malicious scripts are additional threats in most of the current browsers. Moreover, the integration of smart cards and strong authentication are not properly addressed.

Document processing applications are also part of standard PC applications. For example: MS Office [52] and OpenOffice [50][53]. Both provide features to encrypt documents using symmetric keys. Keys are stored internally in the protected file. This represents security weakness, since an attacker can discover the key by applying dictionary or brute-force attacks. Document Security Systems [54], a commercial product, provides security functions like: illegal scanning, copying, digital imaging, protection of personal identification, authentication and authorization. Jakarta Slide [55] and JLibrary [56] provide security functions and services like security locks, constraints on documents, authentication and authorization. Protection of documents using advanced cryptographic techniques was explained in [57]. That research addressed security issues of documents stored at a local station and shared in group environments. In addition, the solution structures documents in sections accessible only by authorized group members. The enforcement of authorization policies and protection of sections are achieved by using Role–Based Access Control and symmetric key cryptography. The system was implemented as an extension of OpenOffice using XACML [58], XACML Policy Server, and Policy Enforcement Point (PEP) Server.

Analysis of Secure Client Applications

It may be emphasized that all examples of current security features and applications are (a) limited in scope, (b) available only locally in individual applications, (c) applicable only to resources of specific applications, (d) not extendable or replaceable with stronger solutions, and finally, (e) complicated to set-up and use.

2.1.6. Secure E-mail System

E-mail security is normally based on signature and encryption of E-mail contents using S/MIME standard, while transmission level security is achieved using TLS/SSL [59]. Most of E-mail security solutions are based on the concept mentioned in [59]. For example in [60], new E-mail architecture is proposed based on web services, but it also provides security services using TLS/SSL. Furthermore, this solution provides prevention from viruses using virus scanning and spam mails protection by enforcing E-mail acceptance policy. Intelligent E-mail Management System [61] is another solution that is based on inclusion of "intelligent" code to identify malicious E-mails and protection of inboxes from spam mail. This code is actually interpreted by client and server to manage rules and policies for sending and receiving E-mails.

eSecure Mail system [62] provides security solution to secure E-mails using deployment of eSecureMail software at gateways. This system ensures that each E-mail entering into local network is coming from legitimate sender and does not contain any viruses. They are using header and contents filtering mechanisms to eliminate spam mails and anti-viruses were used to protect network from viruses. Furthermore, Gateway software is responsible to handle security issues. The author of the paper [63] describes another certificates–based solution to check the authenticity of users. In this case, recipient presents his/her certificate prior to fetching E-mail or downloading attachments.

Smart card is considered a good option to protect user credentials. Solutions proposed in [64] [65] present E-mail software integrated with smart card used to store secret user credentials. According to a recent survey, most of users still send and receive E-mail without using security features, because they are unable to configure security settings. One such solution is "Attribute-Based Usefully Secure E-mail System" [66] that introduces even additional burden on users to define their own attributes for receiving and sending E-mails.

Current E-mail systems protect E-mail letters using signed and enveloped MIME and use SSL/TLS for secure communication purposes. But some software does not support protection of E-mail letters from source machine, because they provide cryptographic services at gateway level. Moreover, some systems provide security features as optional and assume that end-user have sufficient technical knowledge about security configuration. Most of current spam solutions work at a Gateway or E-mail server level, but this feature is usually not available for individual users. However, in [67] it is mentioned that none of these methods are 100% effective against spam.

Analysis of E-mail Security

Our analysis established that current E-mail systems protect E-mail letters using signed and enveloped MIME standard and use SSL/TLS for secure communication. But some software does not support protection of E-mail letters from a source machine, because they provide cryptographic services at a gateway level. Moreover, some systems provide security features as optional and assume that end-users have sufficient technical knowledge about security configuration. Current systems do not provide transparent handling of certificates, strong authentication, protection of address books, management of protected address books, efficient handling of attachments, and confirmation E-mails. Most of currently anti-spam solutions work at a Gateway or E-mail server level, but this feature is usually not available for individual users. We also found that trust between sender and receiving domains is not properly established using some cryptographic functions. In addition, current E-mail clients do not protect entries in address books and do not provide effective key management.

2.1.7. Secure Web System

In this section, we analyzed some of the solutions which are being used for protection and authorization of Web content. Existing solutions are categorized as follows:

Web Shields

Attackers primarily target workstations for exploitation and spreading malicious code by devising various sophisticated techniques. These techniques [68] can be categorized in two groups: pull-based and push-based. The purpose of both techniques is to download and execute malicious code on workstations via E-mails or insecure Web contents. Drive-by-download attack is one of them. It uses pull-based techniques to download malware binaries [69]. It uses HTTP protocol as a carrier and Web mobile components for hiding and obfuscation purposes. Examples of Web mobile components are ActiveX [70], Java Applets [71], Flash scripts [72], plug-ins, etc. Some of these spyware exploit network connection of compromised workstation with the attacker to reveal weaknesses of the targeted for further exploitation. Most effective tools to combat against such types of attacks are Web shields, which are normally bundled with antivirus software. Some examples are Symantec Web Security Monitoring [73], Norton Internet Security [74], AVG [75], Avast [76], etc. These tools use virus/threat pattern and signature database to effectively detect the latest viruses and threats.

Intrusion Detection Systems

Various Intrusion Detection Systems (IDS) are developed to monitor network traffic and system activities. The purpose of such software is to detect malicious activities or policy violations in workstations or in a network, which are eventually reported to the administrator of a management station, but these systems do not prevent workstations from various Web threats. SNORT [77] is an example of such an IDS, which is an open source cross-platform lightweight network intrusion detection tool used for network traffic monitoring in order to detect suspicious network activities. It has rules-based logging to perform content pattern matching and to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, etc. Another IDS, nCircle [78], provides security risk and compliance management solutions. Reflex Security's Intrusion Prevention™ [79] solutions provide end-to-end enterprise network protection. Reflex

IPS applies packet inspection with signature, anomaly and rate-based algorithms to inspect and control network traffic flows. This detection methodology already proved to produce either high rate of false positives or false negatives. Nessus™ [80] is another vulnerability scanner that provides a couple of good features, like efficient discovery of vulnerabilities, network configuration and auditing, asset profiling, etc. However, the major problem with Nessus is that it requires significant involvement of security administrators.

Protection and Authorization of Web Content

Most of Web sites provide SSL-based connections (HTTPS) in order to protect communication channels. SSL protocol uses certificate-based security solution for authentication and content protection. Normally, SSL-enabled Web sites dynamically download certificates into the client browser. However, in some cases, user may select a wrong option and browser overrides certificates verification, what increases the probability of man-in-the-middle attack over HTTPS. The same weakness of SSL is pointed out by T. Burg [81], which can be prevented by properly handling and verification of certificates. Along with Web contents protection, some Web servers implement access control mechanisms in order to restrict provision of Web contents to authorization users. Password-based access control and Access Control Lists are representative examples of such mechanisms.

Analysis of Web Security

After reviewing existing security products and solutions, we found that dynamic code loading, Web contents modification, and hacking of legitimate Web sites are major security threats in current Web systems. Currently available Web shields and Intrusion Detection Systems require a regular update of signature database. Thus they do not provide an adequate level of protection to workstations against ever increasing Web threats. In addition, these tools do not ensure the integrity and confidentiality of Web contents downloaded to a workstation. SSL provides confidentiality and integrity at a message level, but Web pages stored on a Web server are in a clear text. Thus, there is a possibility that the attacker may illegally insert malicious scripts in those Web pages after gaining illegal access to a Web server. In order to prevent such illegal access, some access control products and technologies are available, such as firewalls, access control systems, etc. However, they are compromised as well [82]. Furthermore, execution environment of current Web servers do not support processing of encrypted Web pages.

2.1.8. Secure Documents System

The concept of storing data in an encrypted form was suggested in 1990s by Blaze92 [83] and designed as the Cryptographic File System (CFS) for UNIX operating system. Later, it was extended by Cattaneo97 [84], Zadok98 [85], and Hughes99 [86]. The system encrypts every file before storing it in a local repository or before sending it to remote servers. It decrypts the requested file before presenting it either to the client or to the intended server. Furthermore, the system provides security for communication channel between the sending host and networked file server. CFS transparently manages protection of documents based on symmetric keys, while access control is handled by applying key-level access control mechanisms. The author proposed a secure document distribution system based on Public-Key Cryptography [87] and Secure Socket Layer (SSL) [88] for secure distribution of documents.

Oracle Beehive is one of the products which provides unified single platform for all communication and collaboration services and documents sharing. This product uses user name /password-based authentication, access control lists for authorization, Secure Hypertext Transfer Protocol (HTTPS) [89], and SSL for secure communication. This product also provides audit capabilities, so that system administrator processes audit trail on log files generated by the system during day-to-day actions.

IBM Lotus Notes [90] is another popular tool for collaborative and team work. Similar to Oracle's Beehive, it is also a combination of different applications and document sharing is one of them. This product supports certification protocol, username/password authentication, Single-Sign-On, SSL with support of AES, and it is complaint with FIPS 140-2 [91] standard. It uses access control lists for access control services. This product uses .ID file to store user credentials which are protected by user's password. In addition, this product applies different levels of encryption according to the sensitivity of documents stored in a local workstation.

Protection of documents using advanced cryptographic techniques was explained in [92] [93]. That research addressed security issues of documents stored at a local station and shared in group environments. In addition, the solution structures documents in sections accessible only by authorized group members. The enforcement of authorization policies and protection of sections are achieved by using Role–Based Access Control and symmetric key cryptography. The system was implemented as an extension of OpenOffice using XACML [93], XACML Policy Server, and PEP.

Another major example is Xerox DocuShare [94]. This solution implements security features like user name/password authentication, SSL to secure communication, different level of access policies to access contents, different roles to access encrypted contents, etc. Adobe [95] solution for trusted document sharing brings a new product for exchange of documents in government organizations. This product uses asymmetric key cryptography to encrypt and digitally sign documents and distribute them securely in a grouped environment.

Analysis of Document Management Systems

We analyzed and found that most of the current products and solutions focus on user authentication and access control of documents in a distributed environment. Some of solutions provide cryptographic protections of documents. In most of available solutions, security functions and features are available only locally for individual application resources. Some products are very complicated to setup and very difficult for end-user to understand, because each application has different settings and interfaces. Currently available document management applications do not provide end-to-end security. In addition, existing applications provide document-level security, but other than [92] and [93] do not provide sections level security and access control.

2.2. Summary

In this review and analysis section, our strategy was to investigate security features, functions, design and development methodologies of various components of secure distributed applications and cloud computing environments. As concluded, it may be emphasized that all examples of current security features and applications are (a) limited in scope, (b) available only locally for individual applications, (c) applicable only to resources of specific applications, (d) not extendable or replaceable with stronger solutions, (e) complicated to set-up and use, (f) based on various models and methodologies, (g) based on proprietary technologies and finally, (h) some are based on non-standard formats and protocols.

These shortcomings of the current security solutions represented motivations, framework, and scope of our research activities and results. As explained further in this thesis, we have successfully solved most of the stated shortcomings and deficiencies. Thus, we claim that our system and methodology represents significant contribution to the state-of-the-art of computer networks and applications security.

Part I: Generic Components and Their Validation

3. Generic Security Objects

In this Chapter we describe the concept of generic security objects. They support various alternatives and options to provide a complete set of security functions and features. We categorized our generic security objects in three types: Entity object, Functional object, and Composite object. We protect object source code, its instantiation, usage, attributes, actions, states and communication with other objects. So each of our generic security objects is protected in all aspects.

3.1. Approach and Methodology

An object is a basic concept of a modern object-oriented software engineering methodology. An object may contain internal data available to other objects by exposing them through public interfaces. In general, some objects may contain only data and support only `get()` and `set()` public interfaces to manipulate those data. Such objects are called *`data`* or *`entity objects`*. The best examples are database template classes in Java, representing entities in DB tools. Other types of objects perform also some functions. Therefore, they are called *`functional objects`*. The best example may be objects representing crypto algorithms, with public interfaces such as *`encrypt()`* or *`decrypt()`*. We extended standard concept and design of objects and we created the concept of generic security objects for developing secure applications. Like standard objects, they also contain data and functions, but data attributes are protected and functions are only accessible to authenticated and authorized objects and users. Furthermore, our objects support multiple alternatives, so that selection of different options and variations is simplified. Individual generic security objects provide complete set of functions related to some specific aspect of security. For example, *`Symmetric Key`* object supports various symmetric key algorithms along with different key sizes, but it also provides complete set of functions which are required for standard symmetric key cryptography. It also provides various symmetric key management functions like save, load and create symmetric key. These functions can be invoked only by authenticated and authorized users or other objects in order to securely store, retrieve or create *`Symmetric Key`* object. For storing symmetric key, it either uses standard key-file or it can be stored in a smart card, which is transparently handled by this generic object.

Each generic security object is protected in all of states. Its attributes are encrypted and hashed, so it enforces confidentiality and integrity of object’s internal data. Various actions of an object can be performed only by authenticated and authorized objects or users in a secure way. These actions are: instantiation, usage, method calling, and its interactions with other objects. Each generic security object is also protected when it is stored in a local file system. Furthermore, each object protects all of its states which are securely maintained and manipulated in a secure execution environment.

In object-oriented software engineering methodology, an object may be combined with other objects. Similarly, each of our generic security objects may also be linked or interconnected with other generic security objects in order to perform complex security functions. The link between objects is defined as `security protocol`. For example `DigitalSignature` is a complex generic security object which instantiates `Hash` object to compute hash value while private key of `Asymmetric Key` object is used to encrypt hash values in order to generate digital signature. The `Asymmetric Key` instance provides access to private key after authenticating and authorizing caller object. Therefore, security protocol provides secure way for accessing its methods by other objects.

In our system, each generic security object is individually tested and verifiable. So based on our deductive scheme when a security system is designed and implemented using such objects, it is also secure, tested and verifiable.

3.2. Design of Generic Security Objects

A generic security object is the basic component of our security system which supports multiple alternatives and options. Such objects are easy to understand, they are individually verifiable, they are based on well-established security standards and technologies, and they transparently handle security credentials, if required.

Multiple alternatives of each object are handled by designing and implementing several types of constructors for each object. They are carefully designed according to the requirements of various secure applications. Furthermore, we considered authentication and authorization parameters, which are passed as arguments for instantiation of each generic security object. We also implemented various cryptographic functions and most of functions accept security parameters along with operational data. Security parameters can be password, if software-based cryptographic functions are used. Otherwise, our generic security objects accept PIN, if they are using smart card-based cryptographic functions. In the implementation of our generic security objects, the name of an object and its functions are very close to natural language. They are very easy to understand. Furthermore, they are structured in the form of high–level objects, so that understanding of their concepts, functionality, invocation, and use is very simple and easy for developers.

The operations of our generic security objects are based on well-established security standards. If some security service has various standards, then our objects provide flexible procedure to incorporate all the standards. Based on our approach and methodology, attributes of generic security objects are protected, their methods are accessible only to authorized users and their states are secure, so each generic security object is secure, tested, and verifiable.

Using our approach and methodology, we designed generic security objects for cryptographic functions, cryptographic encapsulation techniques, security protocols, and transparent handling of security tokens.

3.2.1. Types of Generic Security Objects

Some objects contain and maintain data in the form of attributes, while some objects perform actions with those attributes. Similarly, each generic security object is designed for specific roles and responsibilities in order to provide extended security functions and features in open, distributed and cloud computing environments. Based on roles and construction of each generic security object, we categorized them in three groups: (1) entity objects, (2) functional objects, and (3) composite objects.

Entity Objects

Entity objects contain only data and provide various functions to expose attributes of the object. For example, `DistinguishedName` is a complex generic security object which supports alternative X.500 attributes and their values. It is generic in nature, so it contains various attributes depending on requirements of applications and supports various formats. This object provides data to various other generic security objects used to uniquely identify users, applications, and resources like `Certificate` or `AsymmetricKey`. An object `DistinguishedName` can be instantiated using one of the following constructors with different attributes:

```
DistinguishedName()
DistinguishedName(String countryName, String stateOrProvinceName, String localityName, String organizationName, String organizationalUnitName, String commonName, String emailAddress, String urlAddress)
DistinguishedName(String dn)
DistinguishedName(byte dn[])
```

Functional Objects

The second type is functional objects, as shown in Figure 3.1, which contains data and various functions to provide features related to a specific security aspect. For example, `SymmetricKey` is a functional object. It contains symmetric key as data and provides various functions for symmetric key cryptography. This object can be instantiated using one of the following constructors:

```
SymmetricKey()
SymmetricKey(String key)
SymmetricKey(byte[] key)
SymmetricKey(String seed, int hashAlgo, String salt, int skAlgo)
SymmetricKey(String workingDir, String password, String alias)
```

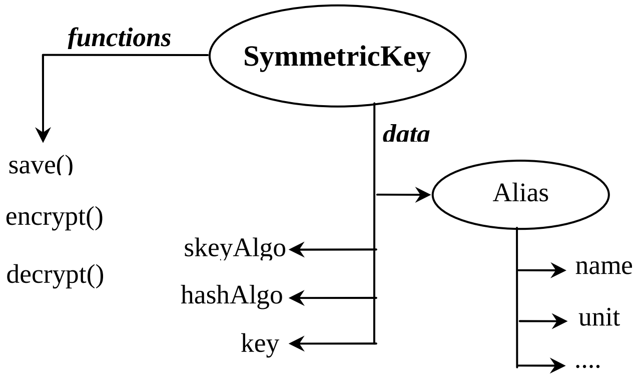


Figure 3.1. `SymmetricKey` Object with Data and Operations

An instance of the *SymmetricKey* object provides various functions related to symmetric key cryptography and encapsulates all functions and features of symmetric key cryptography. Some of its public methods are: *encrypt()*, *decrypt()*, *save()*, *saveWithPassword()*, and *toConvertXXX()*. These actions for invocation require a password as an authentication token.

Composite Objects

The third type of generic security objects are composite objects. Composite objects may contain entity objects and/or functional objects. In some cases a composite object may also be part of some other composite object in order to provide tested and verifiable functionality. *DigitalSignature* object is one of them. This object uses an instance of the *Hash* object and the *AsymmetricKey* object.

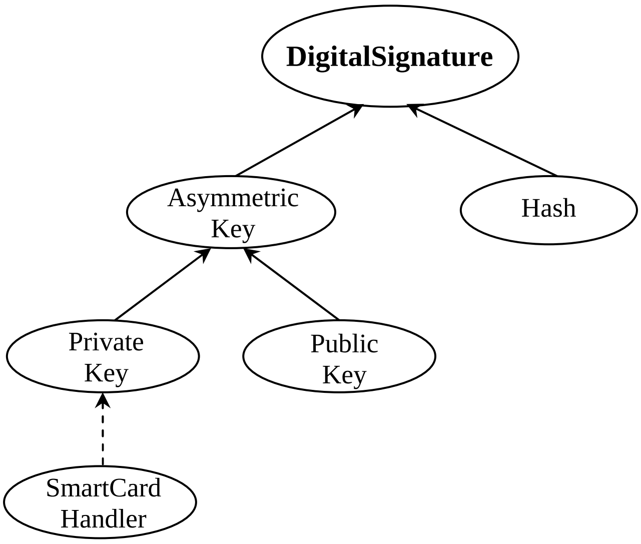


Figure 3.2. Composite *DigitalSignature* Generic Security Object and its Decomposition

The *Hash* object is used to compute hash value and *PrivateKey* of the *AsymmetricKey* object is used to encrypt that hash value to generate digital signature. As shown in Figure 3.2, *AsymmetricKey* object comprises instances of *PrivateKey* and *PublicKey* objects while *PrivateKey* object calls functions of the *SmartCardHandler* object in order to perform smart card-based cryptographic functions. For verification of a digital signature, it also uses *PublicKey* object. *DigitalSignature* object can be instantiated using *Signature(int algo)* constructor*.* Some of its most useful functions are shown in the following code:

```
createDigitalSignature(String password, AsymmetricKey keyRf, int hashAlg, String data)
createDigitalSignature(String workigDir, String password, DistinguishedName dn, int hashAlg, String data)
verifySignature(signedData AsymmetricKey keyRf, int hashAlg,String data)
```

The above functions are accessible only to authorized users, where password is used as authentication token. If *DigitalSignature* object is using smart card-based cryptographic functions, then this password is treated as PIN to open smart card.

The complete set of generic security objects, designed for Generic Security Framework for cloud computing environments, is described in subsequent chapters.

3.2.2. Actions of Generic Security Objects (Security Protocols)

Each generic security object performs some actions while some actions are also performed on data contained in generic security objects. We described the concept of actions in generic security object using examples. One example is *SymmetricKey*. The *SymmetricKey* object can be instantiated using one of the constructors described in Section 3.2.1. When a user or object wants to perform *save* operation, then the object has to provide password in order to open key-file or it must provide PIN to authenticate user to open the smart card before storing symmetric key in a smart card. Similarly, when user performs *generateSignedData()* action of the *PKCS7* object, then authenticated user or object provides its *DistinguishedName* and password. The purpose of password is to access *PrivateKey* of *AsymmetricKey* in order to digitally sign input data. If smart card is attached, then the user or object provides PIN in order to open smart card. After that, the instance of *PKCS7* object transparently sends hashed value to smart card to encrypt it using private key stored in a smart card. Each action of generic security object is performed only by authenticated and authorized user or object which contains required security credentials. Various actions of complex generic security objects, performed in a secure way, are described in Chapter 4.

3.2.3. States of Generic Security Objects

The state of an object is defined as the values of its attributes protected in each of our generic security objects. Normally, in a standard software engineering methodology, object state starts from *Init (Instantiate)* and ends at an *Exit* state. Lifecycle of generic security objects may contain various states and during transitions from one state to another state, we take various security measures. We considered security of each object during its instantiation, usage, function invocation, and its communication with other generic security objects. For example, in Figure 3.3, we describe various states of a *Symmetric Key* object and each state is secured using well-established security standard. In the Pre-Init state our Secure Execution Environment, described in Chapter 7, fetches protected class files from local-file-system and then, verifies signature and decrypts it in order to extract class contents. This state ensures that the class file is not altered for malicious purposes. In the *Instantiate* state, security system permits only authorized users, which have valid credentials, to instantiate the *Symmetric Key* object. During execution, the *Symmetric Key* instance may change its state. For example, in Figure 3.3, *State-1* loads symmetric key from smart card or key-file. For loading symmetric key, it uses security credentials of a current user to identify the key from the key-file and to decrypt of protected symmetric key. If smart card is connected, symmetric key instance uses PIN to open a smart card in order to fetch stored symmetric key. In each state, it performs actions in order to transition to a new state after proper authentication and authorization. In addition, it protects sensitive attributes in each state after transition. For example, in `State III`, it uses password to protect symmetric key before saving it in a Key-file. When an instance transitions into `Exit` state, it deletes its temporary data and external libraries (like dlls etc.) from local files.

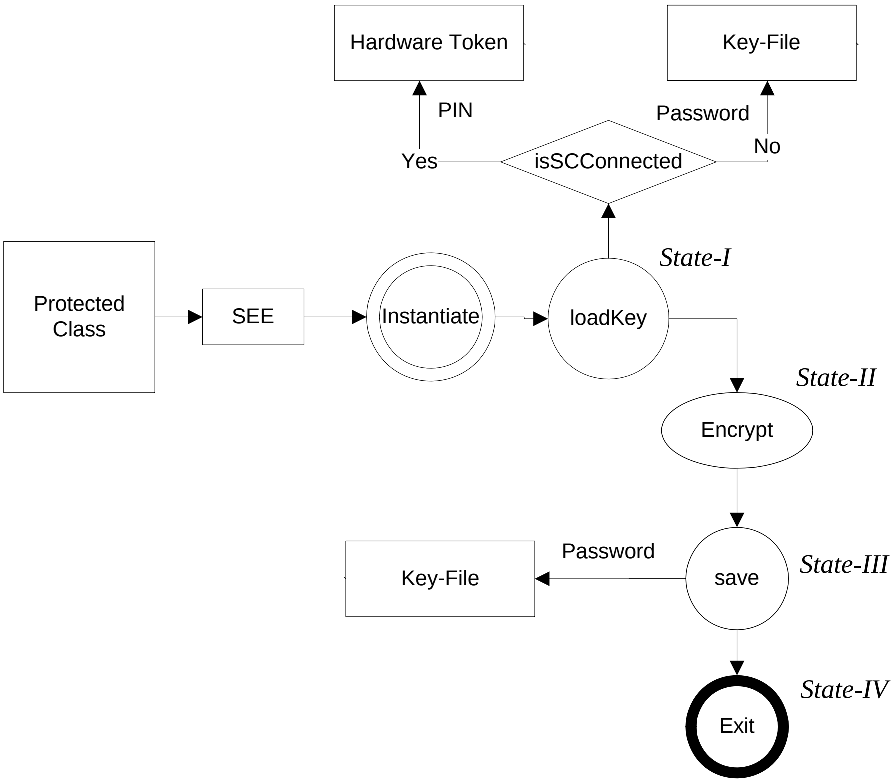


Figure 3.3. Various States of The `SymmetricKey` Object

Similarly, all the designed generic security objects, described in Chapters 4, 5, and 6, perform their actions in a secure way and protect their data and states.

3.3. Features of Generic Security Objects

The following are the main features of generic security objects:

- They encapsulates multiple alternatives, so that selection of different options and variations is simplified;
- They are protected and secure in all aspects: their attributes, methods, actions and states.
- They are complete in terms of their functions which are only accessible to authenticated and authorized users.
- Each generic security object is structured in the form of a high–level object, so that understanding of their concepts, functionality, invocation, and use is very simple and easy;

- They handle transparently their security credentials and hardware tokens.
- Each generic security object performs its actions based on well-established security standards.
- Generic security objects are used to create a uniform and consistent security system.
- The instances of secure, evaluated, and verified generic security objects produced secure, tested and verifiable security system.

3.4. Summary

Generic security objects are basic units of our security system. Each object is generic, since they support various alternatives and options. Attributes of each object are protected using well-established security standards. Objects are accessible through their public functions and interfaces only to authenticated and authorized users. Each generic security object protects its states which are maintained and manipulated in a secure environment. We designed several generic security objects and each of them has well-defined role. Based on the role, a generic security object can be either entity object, functional object or composite object. Entity objects contain data, functional objects implement security functions, while composite objects are combination of entity and functional objects. All actions of generic security objects, like their instantiation, use, invocation of methods, and communications between objects are secure (Security Protocol). Therefore, all objects are protected, all their users are authenticated, and all their actions are performed in a secure way.

4. Generic Security Provider

Generic Security Provider provides a comprehensive set of security services, mechanisms, encapsulation methods, and security protocols for Java, C, C++, and .NET applications. The Provider is structured in four layers; each layer provides services to the upper layer and the top layer provide services to applications. The services reflect security requirements derived from a wide range of applications; from small desktop applications to large distributed enterprise environments. Based on the abstract model, we describe design and implementation of an instance of the Provider comprising various generic security modules: symmetric key cryptography, asymmetric key cryptography, hashing, encapsulation, certificates management, creation and verification of signatures, and various network security protocols. We also designed security applets which provide various FIPS 201 (PIV) smart cards based cryptographic functions and security services. The design of Security Provider describes its properties for extensibility, flexibility, abstraction, and compatibility.

4.1. Overview and Features of Generic Security Provider

Generic Security Provider is an enabling component of our security system. It provides security services using standard security technologies. The model of our Security Provider is based on a layered architecture. It may be used as a reference model for implementation of security provider using any programming language. We describe conceptually related security services of each layer.

Based on the model, we also designed and implemented one instance of our generic Security Provider for various applications. Several versions of Security Provider are implemented using Java, C, C++ and .Net technologies. The design of Security Provider is based on generic security objects, described in the previous Chapter. In this Chapter we describe implementation of our Security Provider with examples in the form of Eclipse plug-ins using Java technology. Other then support to standard security mechanisms and services, the distinctive features and properties of our Security Provider are the following:

- It is structured as a combination of high–level generic security objects, so that understanding of their concepts, functionality, invocation, and use is very simple and easy;
- Each object included in the Security Provider is generic, i.e. it encapsulates multiple alternatives, so that selection of different options and variations is simplified;
- Java Security Provider uses alternative supporting security technologies: software modules, smart cards, or various security tokens, which can be easily configured and switched, even in run–time environments;

- The Provider is structured in the form of several Eclipse plug–ins, it also contains sample code and the full documentation; and
- All the modules included in the Security Provider are encrypted, thus not vulnerable to viruses, worms, malicious code, illegal use or any other forms of threats and problems.

The Provider is combined with our security protocols: strong authentication protocol, Secure Sessions, SAML authorization and several security management modules. Most of these protocols support dynamic behavior and can be used both for client and server components. Security Provider provides security services to the components of the CryptoNET system (described in Chapter 9) in order to support the same set of functions and features for multiple applications.

In addition, our Security Provider transparently uses smart cards for various cryptographic functions and for management of security protocols, if smart cards are installed. We designed security applet for FIPS 201 (PIV) smart cards and Security Provider uses these credentials for smart card-based cryptographic services without extracting private keys from a smart card. Furthermore, attributes of security applets are accessible to various applications using Security Provider.

4.2. Model of The Generic Security Provider

Security Provider is based on modular approach and uses the concept of generic security objects. It is structured in four layers. The layered model of the Security Provider is shown in Figure 4.1. Each layer has distinct role and responsibility, what helps to clearly identify its functions and services. Furthermore, each layer has interfaces in order to provide services to lower layers. The following are the layers:

- Abstraction Layer
- Encapsulation Layer
- Cryptographic Engines Layer
- Resource Management Layer

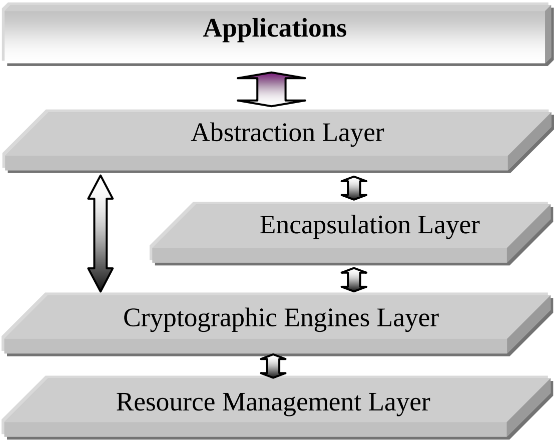


Figure 4.1. Layered Model of Security Provider

Applications are located at the top of the Security Provider and they invoke the required security functions and features. They are not part of the Security Provider and may be developed based on user's requirements. As mentioned above, Security Provider is implemented in the form of plug-in modules. Therefore, in order to use them an application simply includes the required Security Provider plug-ins in its working space. The developer of security applications may use functions and objects of the Security Provider just with the basic knowledge of the implementation details and the structure of its complex objects. Some of our security applications are developed using services of the Security Provider: Secure E-Mail Client, which invokes security functions of the Security Provider to generate S/MIME messages to protect the contents of E-mail letters, Secure Document System uses *PKCS7* object to protect and encapsulate documents in the *PKCS7SignedAndEnvelopedData* format using the required credentials, and so on.

4.2.1. Abstraction Layer

Abstraction layer is the top layer of the Security Provider. It provides interfaces for generic applications and security protocols. The interfaces are user-friendly and complete in terms of their functionality. They hide implementation details of generic security objects and support the functionality of implemented objects and protocols without changing applications. Furthermore, components and concrete objects at this layer are designed and managed as non-replaceable components and can't be added at run-time. This feature protects against tempering of components from viruses and worms.

As mentioned earlier, this layer provides interfaces to applications, so it also contains log-in module which provides authentication of resources. Login module provides this feature by interacting with Cryptographic Engines Layer. This module is generic, so it acquires authentication credentials based on the configuration of upper layer applications. If some secure application is configured to use hardware tokens, then it uses PIN and/or fingerprints. Otherwise it uses username and password which is default authentication protocol for our Security Provider.

Abstraction Layer interacts with Encapsulation Layer and Cryptographic Engines Layer for accessing generic security objects and for security packaging functionalities.

4.2.2. Encapsulation Layer

Encapsulation layer envelopes data and security credentials and thus creates complex security objects. For this, it uses cryptographic engine objects, user credentials, and encoding techniques. All enveloped security objects are based on various security standards. For example, *PKCS7* signed and/or enveloped object follows standard defined in PKCS#7. Similarly, certificates, S/MIME, SAML, resource registration messages, security protocols and other cryptographic messages are packaged according to relevant standards. Encoding and decoding schemes, used in encapsulation process, are also provided by this layer. These encoding and decoding schemes are BASE64, ASN.1, DER, and HexNotations. Furthermore, applications invoke these objects through the Abstraction Layer in order to use them in application-level security protocols.

4.2.3. Cryptographic Engines Layer

Cryptographic Engines Layer implements various cryptographic algorithms and techniques, which include: hashing, signature generation and verification, encryption and decryption, secret key generation, key pair generation, random number generation, and key distribution. This layer acts as an engine of the Security Provider, so accuracy, efficiency and strength of the Security Provider is highly dependent on this layer.

This layer plays important role in extensibility and inclusion of new algorithms. Therefore, generic security objects and interfaces for each security service are carefully designed.

Cryptography may sometimes be a subject of control or export restrictions. So, this layer is also used to enforce export restriction policies for specific algorithms, according to the export or import restrictions for a specific country. Security Provider restrictions are enforced through enforcement of object-level authorization policies and only authorized objects are allowed to be loaded into memory after getting locality information from host computer. This layer interacts with other layers through interfaces used to provide access to cryptographic engine objects. For example, it interacts with Resource Management Layer for fetching or storing user credentials using local storage or hardware tokens.

4.2.4. Resource Management Layer

Resource Management Layer manages security resources used by the Security Provider. Management of these resources is more important than management of general purpose resources, since the overall security of applications depends on them. These security resources are: database with certificates, private and symmetric key files, policy files, configuration properties, and hardware devices (like smart cards or hardware tokens, along with their drivers). The structure and format of these resources is based on relevant standards in order to make them compatible with other implementations. Implementation of interfaces between hardware and cryptographic objects is also managed by this layer.

Hardware devices require drivers in order to make them compatible with Security Provider. This layer manages these drivers as replaceable components and it manages the context and functionality of each device. This functionality enables Security Provider to integrate devices at run-time and to make them operational in order to support running applications.

Errors reporting and exceptions handling along with a logging mechanism in this layer assist applications for debugging processes. These errors and exceptions propagate to the Abstraction Layer which passes them to applications for displaying them in the form of user-friendly messages.

4.3. Generic Security Provider for Java Applications

Design of Security Provider plug-ins for Java applications, based on our abstract model, is shown in Figure 4.2. This figure shows generic security components of the Java Security Provider and interactions between them. These components are categorized based on their services and functionalities. Each component in this design uses an interface to interact with other components. As mentioned above, the model and design of Security Provider follows plug-in architecture, so in our implementation, groups of components are designed as Eclipse plug-ins. Furthermore, this concept also provides well-defined extension points for accessibility and extensibility of Java Security Provider.

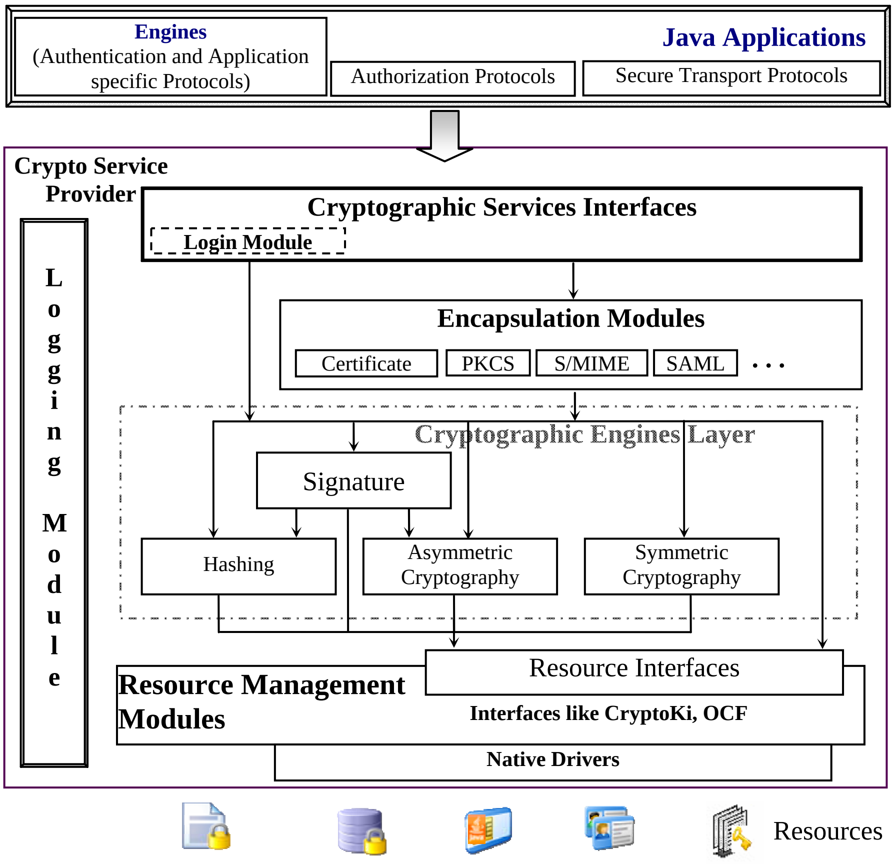


Figure 4.2. The Components of Java Security Provider and Their Mutual Interactions

Security Provider provides transparent handling of security credentials and hardware tokens. We designed security applets for cryptographic functions and management of security attributes of various security protocols. Most of the generic security objects of the Provider transparently use smart cards for cryptographic functions, if smart cards are installed.

4.3.1. Applets for Security Extensions

Security Provider uses two security applets for FIPS 201 (PIV) smart cards. These applets contain security credentials which may be used by the components of the CryptoNET System in order to provide various security services. These security applets store identity information, cardholder’s security credentials and attributes for a secure session protocol. The purpose and model of each applet is specified in the following subsections:

PIV Applet

The Security Provider uses security credentials which are already managed by the PIV applet for various cryptographic functions. This applet contains the following attributes:

- PIV Authentication Credentials: PIV Authentication Private Key, PIV Authentication Public Key and PIV Authentication Certificate. PIV Authentication Private Key is used to digitally sign messages which are verified by the authenticator using PIV Authentication Public Key. PIV Authentication Certificate is a X.509 certificate and can be verified by the authenticator from certificate issuer authority.
- Key Exchange Protocol Credentials: These security credentials comprise Key Exchange Private Key, Key Exchange Public Key and Key Exchange Certificate. The purpose of these credentials is to establish secure session and to exchange session key between secure application and corresponding secure server. In addition, these credentials are also used for protection of symmetric keys. Key Exchange Certificate is a X.509 certificate. In our system, we are using these credentials for SSL protocol in our Secure Web System and for protection of E-mail letters.
- Digital Signature Credentials: This attribute contains Digital Signature Private Key, Digital Signature Public Key and X.509 Digital Signature Certificate. The purpose of Digital Signature Private Key is to digitally sign messages, exchanged between secure application and application server which are verified by the recipient using Digital Signature Public Key of the sender. In our system, these credentials are also used to digitally sign local files, E-mail letters, and SSL messages.
- Card Management Credentials: The purpose of these credentials is to authenticate card management authority in order to perform card management functions, like setting PIN, generating security credentials for card owner, etc.
- Security option file: This file may contain access control policies of security applets.

Security Applet

The purpose of this applet is to manage security credentials which are required by applications for authentication, authorization and secure communication. This applet stores user name, password, domain name, SAML ticket, symmetric key and attributes of a secure session protocol. The roles of these applets are described in the following chapters.

4.3.2. Symmetric Key Object

Symmetric Key component provides secret key cryptography services. It contains generic objects to generate, manipulate, and protect secret keys. Symmetric key encryption and decryption techniques are implemented in this component. Some of supported Symmetric Key algorithms are AES, Blowfish, Cast5, DES, TripleDES, RC4, and Twofish. These algorithms can be used in different modes to encrypt and decrypt data. As mentioned above, this model and design follows the concept of generic security objects, so during the implementation various alternatives are considered like keysize, algorithm and cipher-mode.

In the following Java code, a customized instance of `ISymmetricKey` is generated, with parameters: seed, hash algorithm, salt, and name of symmetric cipher with mode. Its function calls are: `encrypt(data)` for encryption and `decrypt(encrypted_data)` for decryption using `sk` object. Similarly, for protecting key, `getProtected(...)` function can be used which encrypts a key using password and stores it in a working directory, in a designated keyfile or in a hardware token.

```
ISymmetricKey sk = sp.createSymmetricKey( "seed",Hash.NID_sha1,
                                "salt", ISymmetricKey.DES_CBC);
byte[] encrypted_data = sk.encrypt(data);
byte[] cleartext = sk.decrypt(encdata);
sk.getProtected("pwd_PIN");
```

This module also uses Resource Management component for hardware based symmetric cryptographic services.

4.3.3. Asymmetric Keys Object

Asymmetric Key component implements functions supporting public key cryptography. It is structured in the form of a generic security object (`AsymmetricKeys`) which generates public and private keys based on object arguments. These arguments are key size and/or algorithm id.

Both, Public and Private Key objects support functions to encrypt and decrypt data. Like Symmetric Key component, functions of Asymmetric Key object are accessible by other components through interface that provides access to public and private key's attributes and their methods. For example, the following code creates an instance of `IAsymmetricKeys` and then invokes encrypt function using `privateKey`. Furthermore, the same instance can be used to protect private key using `protectWithPassword(…)` function.

```
IAsymmetricKeys ask =  sp.createAsymmetricKey(1024);
byte[] d = ask.privateKey().encrypt(data);
ask.privateKey().protectWithPassword("password");
```

Similarly, the following code creates public key `pk` using the function `getPublicKey(…)` of `ask` and then `pk` object is used to encrypt and decrypt data using public key. `pk` also implements `getValue(…)` function to return the value of public key.

```
PublicKey pk = ask.getPublicKey ();
byte b[] = pk.encrypt(data);
byte clear[] = pk.decrypt(b);
byte [] kValue= pk.getValue();
```

This component interacts with the Resource Management module, if a smart card or cryptographic hardware device is attached for encryption or decryption. Furthermore, if cryptographic hardware token is connected, it generates Asymmetric Key using hardware token and does not permit to extract private key.

4.3.4. Hashing Object

Hashing component provides functions and features to generate message digests for integrity of messages. This module contains implementation of various standard hashing algorithms. Some of these are Message Digest-2 (MD2), Message Digest-5 (MD5), HMAC, SHA-1, and SHA-256. Hashing services are available to other components through interfaces. For example, the following Java code creates a reference of *IHash* interface which implements functions like *compute(…)* for computation of hash value and *verify(…)* for comparing hash values using SHA1 algorithm.

```
IHash h = sp.createHash(IHash.NID_SHA1);
byte [] hash = h.compute(data);
boolean result= h.verify(hash, data);
```

4.3.5. Digital Signature Object

Signature component implements algorithms required for generation and verification of signatures. Implementation of *ISignature* interface supports alternatives and combines different types of digital signature generation techniques that act according to input given as arguments by end–user or application. Some of these are Digital Signature Algorithm (DSA), RSA, etc. This component uses other components of Cryptographic Engines Layer to generate and verify digital signature. For example, the following Java code creates an instance *sg* of *ISignature* class which uses RSA algorithm to generate and verify digital signature. By using *sg*, the developer may invoke *createDigitalSignature(…)* method to generate signature using specified hashing algorithm and *keyRef*. The developer may call *verifySignature(…)* method to verify signature using required arguments.

```
ISignature sg=sp.createSignature(ISignature.RSA);
byte [] sign= sg.createDigitalSignature(password, keyRf, hashAlg,
data);
boolean result = sg.verifySignature(signedData, keyRef,hashAlg);
```

Signature module also checks the presence of various hardware devices attached to the Java Security Provider. If a required hardware device is attached, then it invokes Resource Management module for signature generation and verification using that hardware device.

4.3.6. Encapsulation Objects

Packaging and enveloping services are implemented by several encapsulation components. These components envelope data and relevant information into PKCS#7, PKCS#10, S/MIME, SAML messages, certificate, keys packages, Kerberos tickets and domain specific secure messages. Encapsulated objects are implemented using the concept of generic security objects, so these encapsulated objects provide the complete set of functions and methods. Furthermore, the name of interfaces and functions are developer's friendly. For example, in the following Java code the developer creates an instance of *ISmime* and then invokes *createSmimeSignedData (…)* method and *createSmimeEnvelopedData (…)* method. These methods get input from user and then interact with other components to generate signed and enveloped S/MIME messages. In this code the *createSmimeSignedData(…)* method generates signed data in standard S/MIME format. It stores the result in a file which is created in the same directory.

```
ISmime smime = sp.createSmime();
smime.createSmimeSignedData( sender, password, inputFilePath);
smime.createSmimeEnvelopedData(sender,recipients,password,
 inputFilePath);
```

Implementation of encoding and decoding schemes is also part of this component. These schemes are BASE64, ASN.1, DER and HexNotation. They are also developed using generic objects model and provide interfaces to other objects. Encapsulation module interacts with other components, like Symmetric Key, Asymmetric Key, and Signature, to invoke cryptographic functions. Furthermore, it can directly contact Resource Management component to interact with hardware devices or certificate–related functions. The interactions with other components are performed through interfaces.

4.3.7. Resource Management (Resource Interfaces)

Resource Management component is a facilitator component in the system and performs functions explained in the Section 4.2.4. Resource Management Module contains Resource Interface to interact with resources using resource specific interface. It provides implementation of various security functions which may invoke other modules either to perform cryptographic functions using cryptographic hardware devices or storage of security credentials in files or hardware tokens. This module encapsulates functionality of implemented functions and features. Some of the functions are: *createCertificate(…)*, *requestCertificate(…)*, *getDigitalSignature(…)*, *encrypt(…)*, *decrypt(…)*, *generateSCSymmetricKey(…)*. These functions are accessible to Cryptographic Engines Layer's modules using our designed interfaces. For example, Java code described in Section 4.3.5 is used to generate signature. This function invokes functions of Hashing module to compute message digest and passes it to the Resource Interface which calls *getDigitalSignature(…)* method. This method invokes hardware specific interface or library and sends message

(Application Protocol Data Unit) to a hardware token for encryption of hash value using the referenced private key.

Resource Management module provides context for different security devices and resources attached to a computer for execution of various cryptographic functions and for storage of security credentials. This layer also supports standard interfaces required to communicate with specific hardware devices. These interfaces are Cryptographic Token Interface Standard (PKCS#11), Open Card Framework (OCF), and Personal Identity Verification (PIV). Resource Management module integrates drivers and interfaces as replaceable components, because this features helps the Provider to use new devices attached at a run-time.

4.3.8. Logging

Logging component plays vital role for managing the log about various activities and operations of each object. Each log entry provides information about object, operation, nature of activity, class name, logging date and time. For example, in the following Java code, *LogFactory* creates an object of *Log* which encrypts log entry before storing it into log file. Log messages are grouped based on the type of message. Each group provides specific information about event which are implemented in separate methods. For example: *log.info(…)* method stores information message, *log.error(…)* method writes errors along with exception.

```
Log logger_ = LogFactory.getLog(className);
logger_.info("Some message");
logger_.error("Some message", exp);
```

Log file is stored on a local hard disk in the encrypted format. Upon user request, logging module decrypts log file and displays log information in a user–friendly form. This helps to find anomalies and to analyze the behavior of the Java Security Provider.

4.3.9. Abstraction Module (Cryptographic Services Interfaces)

Abstraction module provides cryptographic service interfaces offered by different components in order to access implementation of each component. The names of interfaces and methods are user–friendly and provide complete functionally of a particular component.

Abstraction module provides *SecurityProvider* interface which is used to get an instance of different components. For example, in the following Java code *SecurityProvider* creates an instance of the *Certificate* object named *cert*. Then by using *cert*, different methods like *save(…)* or *request(…)* can be called. Furthermore, *ICertificate* interface provides also functions to get *PublicKey* and *PrivateKey*. These objects can be further used for encryption and decryption.

```
SecurityProvider sp= new SecurityProvider();
ICertificate cert = sp.createCertifiacte();
```

```
ASN1ncoding asnCert = cert.request(subject);
cert.save(asnCert);
byte [] enc = cert.getPrivateKey("pwd").encrypt(data);
byte [] data = cert.getPublicKey().encrypt(enc);
```

To perform some security functions, applications may use cryptographic services of hardware tokens. For that, *Login* module is used to authenticate resources stored on a local system in files, database or in hardware tokens. Implementation of the *Login* module is also based on the concept of generic security objects, so it supports PIN-based authentication, if smart cards or hardware tokens are connected. Otherwise, it uses username/password-based authentication.

Abstraction module invokes logging, symmetric key, signature and resource management components for cryptographic services and maintenance of log files. In addition, this module contains a generalized higher level exception *SecurityProviderException*, which is extended by other components for run-time errors.

4.4. Eclipse Packages (Plug-Ins)

Java Security Provider is organized in the form of the following Eclipse plug-ins.

- *com.cryptonet.securityprovider*
- *com.cryptonet.securityprovider.symmetrickey*
- *com.cryptonet.securityprovider.asymmetrickey*
- *com.cryptonet.securityprovider.signature*
- *com.cryptonet.securityprovider.ecapsulation*
- *com.cryptonet.securityprovider. logging*
- *com.cryptonet.securityprovider.engines*
- *com.cryptonet.securityprovider.resources*

Each plug-in may contain various packages which group logically correlated objects. For example, *com.cryptonet.securityprovider.engines* contains *com.cryptonet.securityprovider.engines.strongauthenticatio,* *com.cryptonet.securityprovider.engines.securesession,* etc. *com.cryptonet.securityprovider.strongauthentication* package contains only those objects which are required for strong authentication. This packaging structure provides extensibility feature and it can be extended with new security functions and protocols.

It is important to emphasize that all modules of the Java Security Provider are digitally signed and encrypted. Therefore, they are loaded by a special Secure Class Loader which verifies digital signature and decrypts modules at execution time (see Chapter 7). Furthermore, Java Security Provider loads available plug-ins at start-up and does not load any other security plug-in(s) after verification of digital signature. This feature protects security modules from viruses, worms, malicious code, illegal use or any other forms of threats and problems.

4.5. Summary

The Security Provider for Java Applications is complete in terms of functionality which provides security services, mechanisms and protocols. The interfaces and methods of the Security Provider are user friendly and generic. Furthermore, the packages of the Provider are self-protected and loaded by the special Secure Class Loader which verifies digital signature and performs decryption of packages. The implementation of Generic Security Provider is platform independent, so it supports Windows and Linux operating systems.

5. Generic Security Protocols

In this chapter we describe several network security protocols used by various components of the CryptoNET system. The protocols are based on the concept of generic security objects, well-established security standards, and technologies. Distinctive features of our security protocols are: (1) they are complete in terms of their functionality, (2) they are easy to integrate with applications, (3) they transparently handle security credentials and protocol-specific attributes using FIPS 201 (PIV) smart cards, and (4) they are based on generic security objects. These protocols are: initial user authentication protocol, remote user authentication protocol, Single-Sign-On protocol, SAML authorization protocol, secure sessions, and key distribution protocol. Security protocols use Security Provider as a collection of cryptographic engines implemented either in software or using FIPS 201 (PIV) smart cards supporting protocols' attributes by security applets.

5.1. Overview and Features of Generic Security Protocols

Generic security protocols are enabling components of our security system that provide network security services to various components of the CryptoNET system. These protocols are: initial user authentication protocol, remote user authentication protocol, Single-Sign-On protocol, SAML authorization protocol, secure sessions, and key distribution protocol.

Design of the protocols is based on the concepts of generic security objects and modular approach. Each protocol is complete in terms of its functionality, each is easy to integrate with other components, and each transparently handles security credentials and attributes. In addition, they provide the same set of secure network services to all components of the CryptoNET system.

Security Protocols use Security Provider as a collection of cryptographic engines. Security Protocols described in this chapter provide user authentication, secure communication, user authorization, and key distribution services to the components of the CryptoNET system. For example, initial user authentication protocol provides user authentication using FIPS 201 (PIV) smart card. Remote user authentication provides remote authentication based on FIPS 196 strong authentication protocol. Single-Sign-On authentication protocol provides user authentication at application servers using SAML tickets. Secure communication protects messages exchanged between clients and application servers. Key distribution services support distribution of keys between clients and servers and also in group environments. The following are the main features of our security protocols:

- They are all based on generic security objects and modular, so the same protocols provide the same set of security services to all components of the CryptoNET System;

- They are all fully compliant to well-established security standards, like FIPS 196, SAML, GSAKMP, etc.;
- They use the same Security Provider in order to provide the same set of cryptographic services;
- Security protocols are easy to understand, so developers can easily integrate them with their applications.

5.2. Design of Security Protocols

Design of security protocols is based on modular approach and each module is implemented using the concept of generic security objects. Security protocols are initial user authentication using FIPS 201 (PIV) smart cards, FIPS 196 based strong authentication protocol, Single-Sign-On, secure session, SAML authorization, and key distribution protocol. In our system we used security protocols to provide network security services to various components of the CryptoNET system.

5.2.1. Local User Authentication Protocol

Local user authentication protocol is designed as a generic login module. It supports username/password-based authentication, IDMS-based authentication, and certificates-based authentication. Upon starting the system, the workstation automatically checks installation environment and its configuration and then selects the appropriate protocol. If it is configured for username/password-based authentication, our system acquires PIN and/or PIN plus fingerprint from a user in order to activate a smart card. It fetches username and password from the Security Applet (see Chapter 4) and presents them to the login module of the native operating system. Login module consults user accounts database for authentication. The main problem with this type of authentication is that username and password stored in the smart card must be mapped to the operating system’s user accounts database. The change of a password requires administrative privileges and also requires a change of the password in a smart card. To solve this problem, we store a symmetric-key in a smart card instead of a password and we keep encrypted password in the IDMS. The purpose of the symmetric-key is to encrypt a password before storing it in the IDMS. When user performs IDMS-based authentication, our local login module fetches encrypted password from the IDMS and decrypts it using symmetric key. After that, it presents username and password to the operating system’s login module. Our generic local user authentication module also supports certificate-based authentication. In that protocol, our login module fetches PIV authentication certificate from a FIPS-201 (PIV) based smart card and presents it to the windows login module for domain level authentication.

5.2.2. Remote User Authentication Protocol

In our system remote user authentication is performed using mutual Strong Authentication protocol. It is an extension of the FIPS-196 strong authentication protocol. Its extended security functions are verification of certificates by the Local Certificate Authority (LCA) Server and verification of identities by the IDMS Server. As mentioned above, Security Protocols use Security Provider for software or smart card-based cryptographic functions. So our mutual Strong Authentication protocol also uses PIV credentials and smart card-based cryptographic functions.

In our system client initiates mutual strong authentication protocol with the Strong Authentication (SA) Server and sends PIV authentication certificate to the SA Server instead of the *Hello* message, as specified in the FIPS 196 standard:

$$\mathit{Client} \rightarrow \mathit{SA\ Server:}\quad \mathit{Cert}_{PIV\text{-}a}$$

SA Server receives the certificate and verifies it by sending it to the LCA Server. In addition, it also verifies the distinguished name of the user using IDMS Server. Upon successful verification, SA Server generates random number R_s and sends it to the client. Otherwise, if verification fails, it informs the client and stops conversation with the client.

$$\mathit{Server} \rightarrow \mathit{Client:}\ R_s$$

Client receives R_s and signs it using private key corresponding to the PIV authentication certificate. The following cryptographic functions are used to generate signature of the R_s:

$$h = H\ (R_s) \quad \dots\dots \quad (5.1)$$
$$S(R_s) = E\ (h,\ private\ key) \quad \dots\dots \quad (5.2)$$

In these equations H is a hash function and h is the output of the hash function. E is an encryption function which encrypts h using private key corresponding to the PIV authentication certificate. In the next step, client generates a random number R_c and returns it with $S(R_s)$ to the SA Server:

$$\mathit{Client} \rightarrow \mathit{Server:}\ \{S(R_s),\ R_c\}$$

SA Server receives the message and verifies client's signature using the following cryptographic functions:

$$h = H\ (R_s)$$
$$h` = D\ (S(R_s),\ public\ key) \quad \dots\dots \quad (5.3)$$

In equation (5.3), SA Server uses public key, extracted from the PIV authentication certificate of the client, for verification of the signed challenge $(S(Rs))$.

If h is equal to $h`$, SA Server sends digitally signed R_c and its digital signature certificate to the client. Cryptographic functions are the same as explained in Equations (5.1) and (5.2):

$$\mathit{Server} \rightarrow \mathit{Client:}\quad \{S(R_c),\ \mathit{Cert}_{sa}\}$$

Client receives signed random number and verifies its digital signature using Equation (5.1) and Equation (5.3). But, in this case it uses public key extracted from the digital signature certificate of the SA Server. In addition, it also verifies digital signature certificate from the LCA Server and the identity of the SA Server using IDMS Server.

$$Client \rightarrow Server: S(R_c),\ Cert_{sa}$$

Upon successful authentication, SA Server creates connection with the XACML Policy Server and sends the identity of the client (distinguished name) requesting SAML Ticket. XACML Policy Server validates client's identity using IDMS Server and generates SAML Ticket. SAML Ticket contains ticket-id, identity of the client, timestamp, and IP address of the XACML Policy Server. XACML Policy Server also digitally signs SAML Ticket (ST) using its own private key corresponding to its digital signature certificate. It sends signed ST to the SA Server which then sends it back to the client. Client receives ST and stores it in the security applet in a smart card.

5.2.3. Single-Sign-On Protocol

When client establishes connection with some Secure Application Server, that Server initiates Single-Sign-On protocol. Upon receiving the initiation message, client fetches ST from a smart card and digitally signs it using private key corresponding to the digital signature certificate. It sends ST to the Policy Enforcement Point (proxy associated with the application server) along with digital signature certificate:

$$Client \rightarrow PEP :: Request(ST_s,\ Cert_{DSc}) \ldots\ldots\ldots\ldots \quad (5.4)$$

The PEP component also signs ST and concatenates to it multi-party signature ST_{ss}. The PEP component encapsulates ST_{ss} in the *SAMLAuthenticationRequest* message and sends it to the XACML Policy Server for validation:

$$PEP \rightarrow XACMLPolicyServer::SAMLAuthenticationRequest(ST, ST_{ss}, Cert_{DSc}) \ldots \quad (5.5)$$

XACML Policy Server verifies both signatures. Successful verification of signatures proves that SAML Ticket was received from the PEP and presented by the owner of the SAML Ticket, which provides source authentication. After this, XACML Policy Server consults SAML-Ticket database, in order to validate ST. If *OK*, it sends *SAMLAuthenticationResponse* message to the PEP component, as shown in Equation (5.6), which contains authentication decision:

$$XACML\ Policy\ Server \rightarrow PEP :: SAMLAuthenticationResponse\ (Permit/Deny) \ldots\ldots\ldots \quad (5.6)$$

If the decision is *Deny*, PEP informs the client and terminates the connection without any further correspondence. If it is *Permit*, PEP component establishes secure session with the client.

5.2.4. Secure Sessions

In our system secure session is established after a Single-Sign-On protocol is successfully completed. Secure Application Server requests *KeyExchange* certificate from a client, as shown in (5.7).

Secure Application Server → Client:: Request(KeyExchangeCert$_c$) ....(5.7)

The purpose of the KeyExchange certificate is to securely exchange session-key and session-id between a client and Secure Application Servers. To manage secure sessions' attributes at the application server, PEP creates an active session object for the specific client in a session's container. Each object in the session container contains the identity of the authenticated client, session key, and session id.

Upon receiving of certificate request, the client fetches KeyExchange certificate from a smart card and sends it back to the Secure Application Server, as shown in (5.8).

Client → Secure Application Server:: Response(KeyExchange Cert$_c$) ....(5.8)

Since Single-Sign-On protocol is capable to authenticate clients in a cloud computing environment, there is still a possibility that the attacker may launch replay or impersonation attacks by presenting valid SAML Ticket. To counter such attacks, Secure Application Server receives *KeyExchange* certificate and compares its distinguished name with the identity stored in the session container. In addition, PEP also verifies the certificate chain. Upon successful verification, Secure Application Server generates a session-symmetric-key and session id which is digitally signed by using private key corresponding to its own digital signature certificate and enveloped using public key corresponding to the *KeyExchange* certificate of the client. It then sends session key exchange message to the client, as shown in (5.9).

Secure Application Server → Client :: P(SK, SID), KeyExchange Cert$_{as}$ ....(5.9)

Client receives the message and verifies the signature. Upon successful verification, it opens the envelope using private key corresponding to the *KeyExchange* certificate in order to extract session-symmetric-key and session id. Client stores both session attributes in a smart card, if it is installed. Otherwise, it stores them in a key-file. Client uses session-symmetric-key and smart card-based cryptographic functions to create secure messages in the standard format – *PKCS#7SignedAndEnvelopedData* [17]. The purpose of session-id is to enable the secure application and secure application server to perform secure asynchronous communication.

5.2.5. Authorization Protocol

Authorization policies in our security system are based on the XACML standard [101]. We adopted Role-Based Access Control model, so an authorized person (for example Security Administrator (SA_d)), creates a group and defines access level for each group member along with his/her role and permitted actions. SA_d generates a Policy Token which includes `Target` object used to identify the role of each group member in a group. `Target` contains the name of a group member, the name of a resource, and actions permitted to perform by a group member with the specified resource. In addition, SA_d can also specify `Policy` and `Rules` objects, if required. SA_d saves newly created policy in an XACML policy file.

When an authenticated group member requests an access to a specific resource, it fetches SAML Ticket from a smart card and sends it to the PEP component, along with the name of the requested resource. The PEP component creates *SAMLAuthorizationRequest* message and sends it to the XACML Policy Server. Since, the PEP component and XACML Policy server has required certificates, so the PEP protects the contents of *SAMLAuthorizationRequest* using XML Security standards (XML Signature standard and XML Encryption standard). The XACML Policy Server receives the authorization request and verifies signature, and then extracts its contents. The XACML Policy Server consults XACML policy file and generates *SAMLAuthorizationResponse* message, which contains authorization decision. *SAMLAuthorizationResponse* is sent back to the PEP component in order to enforce authorization policy.

5.2.6. Key Management Protocol

Some of secure applications in the CryptoNET system operate in a collaborative environment and use key exchange protocol to exchange group-key between group members. For this purpose, we designed Generic Key Distribution (GKD) component compliant with the GSAKMP standard [15]. GKD performs key-related functions like key creation, key distribution, and rekeying. GKD supports both Push and Pull-based operations to distribute shared-key. In addition, it works with Secure Application Server in order to perform key-distribution functions. This module works with application servers as a component, so it uses PEP component of a host application server for enforcement of authorization policies for shared-keys.

When a group member requests a group-key, he/she performs Single-Sign-On, and establishes secure session with Secure Application Server, as described in Sections 5.2.3 and 5.2.4. After that, group member fetches SAML ticket from a smart card and sends it to the PEP associated with the Secure Application Server. PEP enforces authorization policies based on procedure described in Section 5.2.5. After successful authorization, GKD sends group-key to the authorized group member using secure communication channel.

5.3. Summary

We designed Security Protocols for authentication, secure communication and authorization between various components of CryptoNET system. Our protocols are based on the concept of generic security objects, well-established security standards and technologies. They transparently handle security credentials and handle protocol-specific attributes using FIPS 201 (PIV) smart cards. In addition, the same attributes can be used by our designed protocols. Our generic security protocols are initial user authentication protocol, remote user authentication protocol, Single-Sign-On, SAML authorization protocol, secure sessions, and key distribution protocol. Our security protocols use Security Provider as a collection of cryptographic engines implemented either in software or using FIPS 201 (PIV) smart cards managing protocols' attributes by security applets.

6. Generic Security Server

In this chapter we describe the design of Generic Security Server which represents basic structure for developers for implementation of various specific Secure Application Servers, supporting as default various standard and extended security functions. All security functions are based on well-established security standards, technologies and protocols. Furthermore, several components, actions and libraries are readily available in the form of Eclipse plug-ins which provide Secure Application Server easy to manage, extended with customized functions, and several actions for its administration.

6.1. Overview and Features of Generic Security Server

Generic Security Server is a template for developers to develop various specific Secure Application Servers. It provides the basic structure for implementing customized secure application servers with extended security features, functions and actions. In the design of our Generic Security Server we have taken completely different approach. In our design, security services are an integral part of each application server in order to provide the same set of security features to all application servers in a domain. These common security services are protection of messages, strong authentication, XACML based authorization, and transparent handling of security credentials. Second, we used generic security objects for modeling of our Generic Security Server. Generic security objects provide complete functionality, support various alternatives and they are secure, tested and individually verifiable. Third, we “vertically” expanded functions of application servers to add new security features. These functions are generic local and remote user’s authentication, transparent handling of security credentials, secure server administration, secure session management, scalability of servers, enforcement of authorization policies, and encrypted AuditLog management.

We designed and implemented our Generic Security Server using Eclipse plug-in architecture which further improved the design, development and deployment of our security server using Java technology. The approach was to use software modules, structured in the form of plug-ins, as basic building blocks to create specific secure application servers. Therefore, not only that better template of Generic Security Server was created for other secure application servers, but they are also available in the Eclipse environment in the form of Eclipse generic security plug–ins.

We believe that this contribution will play significant role for a rapid development of secure application servers, which are based on well-established security standards and technologies. In addition, by using template of our Generic Security Server to develop other secure application servers, the developer has to implement only specific functional requirements of each server. Other then support to standard functions and extended security services, the distinctive features and properties of our Generic Security Server are the following:

- Generic Security Server is complete in terms of standard functions and extended security features;
- Generic Security Server is based on well-established security standards and services such as PKCS#7, X.509 Certificates, SAML Protocols, Single-Sign-On, etc.;
- It supports rapid development of other, specific secure application servers with the same set of security services for each of those application server;
- It provides secure local and remote administration;
- It supports SAML-based Single-Sign-On protocol;
- It enforces authorization policies using XACML standard; and
- It is implemented as an Eclipse plug-in, so it is available in the Eclipse environment for rapid development of customized secure application servers.

6.2. Design of The Generic Security Server

Generic Security Server is designed as a complex generic security object which is a combination of several secure, tested and verifiable individual generic security objects. They are shown in Figure 6.1. Extended security functions and features of Generic Security Server are categorized in three groups. They are: Initialization and Management Functions, Administration Functions, and Client functions. Some of Client functions are already implemented and the developer can extend Client function list using our crypto APIs and libraries. In order to design our Generic Security Server, we extended current design of application servers with generic and extended security features.

All Initialization and Management functions are implemented in the main process (main thread). This group provides certificate management, secure session management, databases connection management, AuditLog management, activation of local and remote user authentication, Single-Sign-On component, and transparent handling of security credentials. We implemented some of management functions in the form of *`Action`*`s`, while some of components are initialized in order to provide services. Our implemented *`Actions`* for administration are Start-server, Login-server, Stop-server, View AuditLog, View Certificate, List Current Users, and Settings.

In the current implementations, client thread receives and processes messages from both, server administrator and clients. If these messages are from the administrator, then it is called administrator thread. Otherwise, it is called client thread. To distinguish between administrative messages and client messages, we used XACML based authorization policies in order to define access of each user to perform authorized action. In order to implement client specific functions, we provided security APIs and libraries for developers. In our design, secure communication and authentication security services are implemented in the client thread, while the required services for clients are implemented using our crypto APIs and libraries.

Various generic security objects of Generic Security Server are shown in Figure 6.1. These are Generic Login Module, Certificate Management, Policy Enforcement Point, AuditLog Management, Security Protocols, Security Provider and Database object. We described operational, functional and implementation details of our generic security object in the following sections.

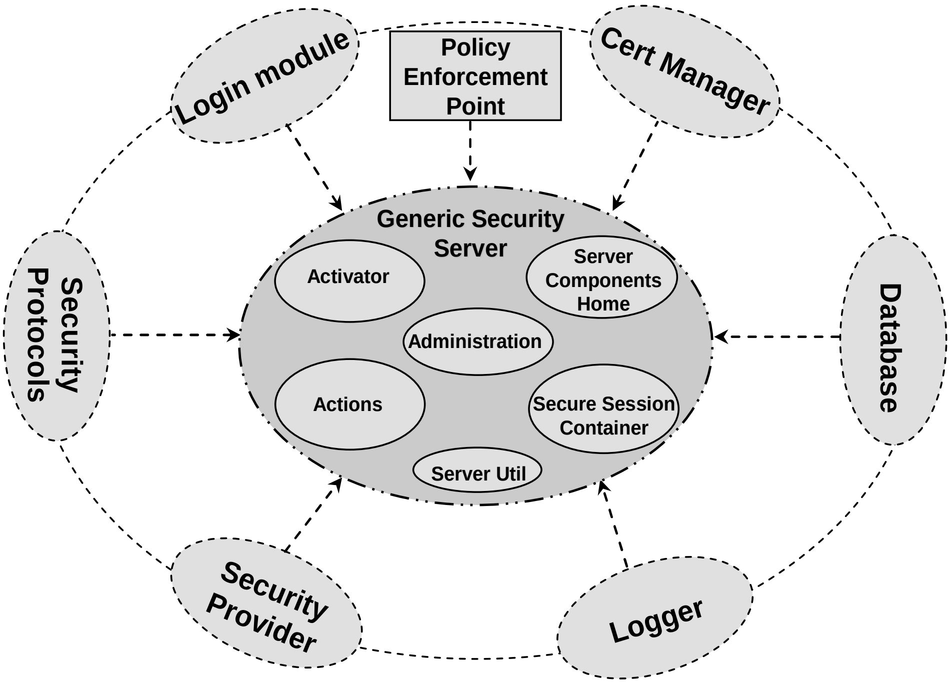


Figure 6.1. Components of The Generic Security Server

6.2.1. Initialization and Management Functions

Generic Security Server supports various initialization and management functions to initialize and manage its components and also various other actions needed to administer secure application servers and their security services. Those functions are Certificate Management, Secure Session Management, Database Connection Management, and AuditLog Management.

At the startup, our Generic Security Server loads database components, Security Provider and all required Security Protocols. It creates connection with the IDMS and Certification Authority Server. Certificate Management component transparently fetches application servers' identity from the IDMS Server and then it checks local certificate database to ensure the presence of the required certificates. If certificates do not exist, Certificate Management module acquires two new certificates from the LCA Server. They are: digital signature and key exchange/non-repudiation certificates. If the LCA Server is not configured, then Certificate Management module generates transparently these two self-signed certificates for Generic Security Server. In addition, our Generic Security Server uses its own password to protect private keys in the private-key-file.

Our Generic Security Server initializes database component and creates the pool of database connections used to store and retrieve server's and application specific data. In order to create connection to the DB server, it transparently fetches DB-URL, port number, username and encrypted password from the configuration file. After that, it decrypts password and then creates connection with the database. Furthermore, it creates database connection pool for execution of application specific database functions.

Session management is an important function of our Generic Security Server. It maintains information about each session. It creates a session container at the startup which contains all session attributes. These attributes are used for secure communication and unique identification of clients. Attributes of the session object are SAML Ticket, Distinguished Name, Session ID, and Session Symmetric Key. For protecting session symmetric key, we encrypt it using public key corresponding to key-exchange certificate. The session object is also a generic security object which is accessible only to the session owner in order to access and add application specific attributes.

AuditLog management is also an important aspect of the Generic Security Server. It manages protected logs containing entries describing various activities and operations performed by administrators and clients. Each log entry provides information about the session owner, object, operation, nature of activity, class name, logging date, and time. *`AuditLog`* object transparently creates symmetric key which is used to encrypt Log entries before storing them into the log file. Our Generic Security Server protects symmetric key using public-key corresponding to the key exchange certificate of the server and stores it in the key-file.

During the initialization phase, Generic Security Server also instantiates Policy Enforcement Point (PEP) to provide Single-Sign-On and XACML-based authorization services. Furthermore, it provides various features like validation of SAML Ticket, evaluation of authorization policies from XACML Policy Server and enforcement of authorization policies.

6.2.2. Administrative Actions

Generic Security Server provides several *`Actions`* which are required for its administration. Server administrator can perform these functions from a remote computer or they can be performed locally. If the administrator administers Generic Security Server locally, then the server only verifies that the user is authorized to administer secure server. If the administrator performs theses actions remotely, then in addition to the above security functions, our Generic Security Server establishes also secure session with the Administrative Station.

We defined some of default and most common security functions in the form of *`Actions`*. In order to invoke those actions, our Generic Security Server enforces transparently authorization policies and permits only authorized roles to perform these *`Actions`*. If a user does not belong to an authorized group, then our Generic Security Server does not allow him/her to activate administrative actions.

The default actions available in the Generic Security Server are standard application server operations, certificate management functions, audit log management, and client management actions. Operating Actions are *`StartServer`*, *`StopServer`*, and *`LoginServer`*. *`StartServer`* action starts the Server. During this action it checks servers' certificates; if they exist, then the action continues; otherwise, it transparently downloads the required certificates from the LCA Server. After successful completion of this Action, it starts accepting requests from users. *`LoginServer`* verifies the credentials of the Administrator using IDMS and then checks the role of the user. If he/she is authorized, then it permits him/her to perform other actions; otherwise it denies him/her to perform further actions. For certificate management, we provided several actions. They are *AcquireServerCertificates*, *ListServerCertificates*, *ViewCertificate*, *VerifyCertificate* and *DeleteCertificate*. These functions are dependent on a specific role and any user that has an authorized role can perform these actions, because Generic Security Server uses its own password to protect servers' security credentials. *AcquireServerCertificates* action is used to acquire new certificates for the Generic Security Server from the LCA Server. *ListServerCertificates* action displays certificates stored in the certificate database, while *ViewCertificate* action displays attributes of the selected certificate. *VerifyCertificate* action verifies the selected certificate. To delete a certificate, *DeleteCertificate* action is used for this purpose.

Generic Security Server provides also several actions to manage AuditLog. They are *ListServerLog*, *ArchiveServerLog* and *DearchiveServerLog*. *ListServerLog* action fetches encrypted log entries from the log file and decrypts them. After that, it displays log entries. *ArchiveServerLog* action compresses log file and then encrypts it before storing it in the archive file with a new password. The *DearchiveServerLog* decrypts archive file and then decompresses it before listing its entries for the administrator. In addition to the above actions, our Generic Security Server provides *ViewCurrentUsersAction* used to display the list of current users and their identity attributes.

6.2.3. APIs and Libraries for Extended Security Functions

In addition to the above described security functions, our Generic Security Server template provides APIs and libraries for cryptographic functions and security protocols. Developers may use these APIs and libraries to implement extended security functions according to the security requirements of their customized application servers.

In order to provide cryptographic functions, we integrated our Security Provider so developers can instantiate specific generic security objects for specific security aspects. Our Security Provider is based on well-established cryptographic standards and also provides the same set of cryptographic functions to all instances of Generic Security Servers. Security Provider can be instantiated using the following code:

```
SecurityProvider sp = ServerComponentsHome.
getSecurityProvider();
ISymmetricKey sk = sp.createSymmetricKey();
```

Our libraries provide an instance of an active database connection in a client thread in order to store and retrieve application specific data to/from database. In the following example, application servers' developer can create an instance of the *EntityManager* object by calling *getDbComponent()* function of the *ServerComponentsHome* object. By using object *entityMgr*, developer may implement application servers' specific functional requirements related to persistent storage. In the following code *createQuery()* is a function of the *entityMgr* which is used to fetch the results from a database:

```
EntityManager entityMgr = ServerComponentsHome.
getDbComponent().getEntityManagerFactory().createEntityManager();
Query query = entityMgr.createQuery(queryStr);
List resultData = query.getResultList();
```

The developer of some secure application server can access attributes of the current user's session using *ServerComponentsHome.getSecureSession()* function, as shown in the following code. This function returns an instance of the *SecureSession* with the local name *ss*. It contains the state and values of various properties of the session of the current user. In addition, this object has also functions to send and receive secure messages. In the *send()* function, it uses *SymmetricKey* object to encrypt message data using session key of the current user. It further signs encrypted data using private key corresponding to the digital signature certificate before transmitting. In the *receive()* function it receives data, transparently verifies its signature, and then decrypts data using session key before instantiating *msg* object. The attributes of the *Message* object are *Message-Header*, *SessionID,* and *data*:

```
SecureSession ss= ServerComponentsHome.getSecureSession();
Message msg = new Message();
msg.setData("Hello");
msg.setHeader(Message.AUTHENTICATION);
int result = ss.send(msg);
msg = ss.receive();
String data = msg.getData();
```

In addition, we provide generic security object for enforcement of SAML authorization policies. In the following example the developer calls *getPolicyEnforcementPoint()* method of the *ServerComponentsHome* object in order to create an instance of the *PolicyEnforcementPoint* object which implements *isAuthorized(…)* function. This function transparently fetches IP number and port number of the XACML Policy Server from the IDMS database and creates a connection with it. This function accepts SAML Ticket (users' identity), resource name and action. This function transparently digitally signs SAML Ticket and then encapsulates resource name and action in the *SAMLAuthorizationRequest* object before sending it to the XACML Policy Server. The same function receives the response *SAMLAuthorizationResponse* object from the XACML Policy Server and processes it. It finally returns the decision and the developer may implement logic accordingly in order to provide access to resources.

```
PolicyEnforcementPoint  pep = ServerComponentsHome.getPolicy
                                                    EnforcementPoint();
String decision = pep.isAuthorized(sAMLTicket, resource, action);
```

In our Generic Security Server we also provide an instance of the *LogFactory* object in order to log various actions of clients and administrators. In the following example *LogFactory* creates an instance *logger_* of the *Log* object. In this instance, we implement *error(…)* and *info(…)* functions for storing error messages and information messages respectively. This object transparently encrypts an entry of the audit log and stores it in the log file:

```
Log logger_ = LogFactory.getLog(this.getClass());
logger_.error(errorMessage);
logger_.info(infoMessage);
```

6.3. Eclipse Packages (Plug-Ins)

Generic Security Server is organized in the form of the following Eclipse plug-ins:

- *com.cryptonet.gss*
- *com.cryptonet.gss.sessionmanagement*
- *com.cryptonet.gss.server*
- *com.cryptonet.gss.actions*
- *com.cryptonet.gss.engines*
- *com.cryptonet.gss.components*
- *com.cryptonet.gss.utils*

This packaging structure provides extensibility feature and it can be extended with new security functions and protocols. Furthermore, all modules of the Generic Security Server are digitally signed and encrypted. They are loaded by the special Secure Class Loader which verifies digital signature and decrypts modules at the execution time.

6.4. Summary

We designed a template of the Generic Security Server which provides complete set of standard functions along with the list of extended security functions and features. These functions are based on well-established security standards and services. The server provides the structure for developers to develop their own components, functions, and protocols for their customized Secure Application Servers. We already implemented several initialization and management functions and several administrative actions. We also included APIs and libraries for cryptographic functions and security protocols in order to provide the same set of security services for all instances of Secure Application Servers. The structure of our Generic Security Server is flexible and is available in the form of Eclipse-plug-ins, which are easy to extend according to the customized requirements of each specific distributed application.

7. Secure Execution Environment

In this chapter we describe our solution for protection of software modules and their execution in a secure execution environment. The solution is based on strong encryption techniques and well established security technologies. We use Hudson server for generation of binaries and Web server for software publication. We introduced a new component, Software Protection Module, which is responsible to protect binaries and their secure distribution using Web server. In this system, protected software modules are packaged in XML files which specify general syntax and provide meta information about software modules. We also extended existing standard execution environment with Secure Software Loader which loads protected software modules for execution.

7.1. Overview of Software Protection and Secure Execution Environment

Secure Execution Environment and Software Protection describes a comprehensive solution for encryption of software modules and their secure execution. It provides software confidentiality, tempering resistance, and even protection against illegal coping and distribution of software modules. Software protection system comprises: Concurrent Versioning System (CVS), Hudson Server, Software Protection Module, and Web Server for software distribution. Software protection is performed by our Software Protection Module which encapsulates software modules in XML files. Each XML file describes general format of protected software modules and supports several cryptographic syntax and standards. Protected software modules are published using Web Server. In addition, we designed and implemented an extended secure execution environment, which is not only capable to execute encrypted and signed software modules, but also supports various security management procedures and protocols, such as certification protocol, secure session handling, use of smart cards, and Single-Sign-On protocol. We believe that our system is an effective solution against reverse engineering, software tempering, and BORE attacks, because it protects software modules using authentication, access control, integrity, and confidentiality. In particular, we also solved a critical issue of secure execution of such protected software modules.

As mentioned in chapter 4, for the development of our software modules we used Eclipse Plug-in Architecture. Therefore, we implemented this solution for Eclipse Plugins and extended existing Plugin Loader (Standard Eclipse OGSI framework Class Loader) to load and execute our protected software modules for the standard Java Virtual Machine (JVM).

7.2. Secure SDKs

For management of source code in our development environment, we used CVS [99]. CVS is combined with Hudson Server [100], as shown in Figure 7.1. Hudson Server generates executable binaries from the source code. Product Manager initiates the process of generation of binaries and Hudson server generates the binaries. Hudson server stores newly generated binaries in a local file system and instantiates Software Protection Component.

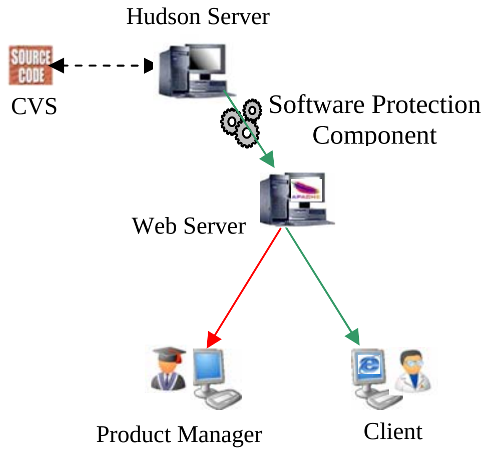


Figure 7.1. Components of Software Protection and Distribution System

Software Protection Component (SPC) encrypts and signs compiled software modules. In this system, SPC is administrated by a Security Administrator (in this case a Product Manager) who acquires digital signature certificate and key exchange certificate from the LCA server. SPC fetches software binaries from the local file system and encapsulates them in the *PKCS7SignedAndEnvelopedData* format.

SPC packages protected software modules in an XML file, which describes the general syntax of protected software modules. The structure of XML file supports several different cryptographic syntax and standards. In this scenario, we used *PKCS7SignedAndEnvelopedData* as a cryptographic syntax. Security Administrator uses his/her security credentials to generate *PKCS7SignedAndEnvelopedData*. After that, he/she generates XML file, whose format is shown in Figure 7.2. It contains information about the cryptographic standard used for protection and enveloping of software modules. Description of each element of the XML file is given in Appendix B. After encapsulating protected software modules in XML files, SPC publishes them using Web Server.

```
<SPS>
    <Version>1.0</Version>
    <Content-Type>SIGNED-ENCRYPTED</Content-Type>
    <Encapsulation-Standard>PKCS7</Encapsulation-Standard>
    <SM-Type>Native</SM-Type>
    <SM-Name>Name of Software module</SM-Name>
    <Content-Description>A description</Content-Description>
    <Contents>Signed and Enveloped contents encapsulated in PKCS7</Contents>
</SPS>
```

Figure 7.2. XML file with information about protected software modules and applied security standards

7.3. Secure Execution Environment

Secure execution environment provides support for execution of protected executable software modules. We extended the existing standard Java Class Loader with security features. Secure Class Loader is a key component of our Secure Execution Environment, downloaded by the client from our Web Server in order to extend the existing environment to load protected Plug-in modules. Secure Class Loader interacts also with cloud security infrastructure to verify and decrypt protected software modules. In our system Security Administrator signs Secure Class Loader according to the procedure described in [96]. In addition, Security Administrator packages it with software modules to form a complete installable product.

As mentioned above, the SPC publishes product on the Web Server, so authorized end-users can download it using Web Interface. For authentication and authorization services, we use Web based authentication and authorization solution described in Chapter 12. In this section, we focus only on operations of the Secure Execution Environment.

Authorized users download the product form the Web Server and install it. During system setups the signature is verified in order to ensure that it is downloaded from an authenticated server and it is not tempered before or during the downloading phase. For this solution, we assume that the workstation has basic execution environment to execute verifiable components. Java Virtual Machine (JVM) is one of them.

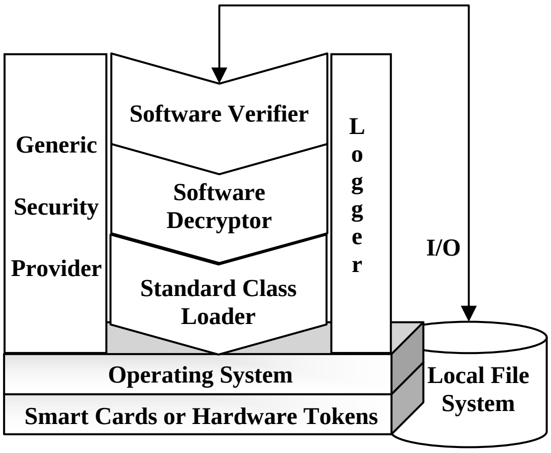


Figure 7.3. Components of Secure Execution Environment

In the execution phase standard Java Virtual Machine verifies the signature of Secure Plugin Loader and then activates it in order to extend existing execution environment with security features.

The Secure Execution Environment comprises five components, as shown in Figure 7.3. These components are: Generic Security Provider, Software Verifier, Software Decryptor, Standard Class Loader, and Logger. Security functions and features of Generic Security Provider are explained in Chapter 4. Software Verifier fetches protected software modules from the Local File System. It reads the header of the XML file and processes it accordingly. As mentioned above, we encapsulate software modules in

`PKCS7SignedandEvenlopedData` format, so it verifies the signature of loaded software modules. If this verification fails, Software Verifier logs the event and stops its execution. Upon successful verification, software modules are passed to the Software Decryptor. Software Decryptor decrypts the value of the *Contents* element of the XML file according to the standards specified in the header of the XML file. The output of this process is executable files. Software Decryptor takes the following actions depending on the value of *SM-Type* element:

- If *SM-Type* is *Native*, then it transfers the module to the Standard Class Loader for loading into memory. Examples are *java classes*, *executable jars, etc.*;
- If *SM-Type* is *Configuration*, then it only uses the required information on-the-fly. Examples are *server configuration* file, *help* files, etc.; or
- If *SM-Type* is *External*, then it saves decrypted modules in the temporary directory. They will be loaded by a specific class loader at the execution time. At closing time, Secure Plugin Loader deletes these temporary files. Examples are *so*, *dlls*, and *exe* files.

7.4. Summary

In this chapter we described the solution for protection of software modules using strong encryption techniques. The solution comprises CVS Server, Hudson Server, Software Protection Component, and Web Server in order to generate and distribute protected software modules to authorized users. Our solution encapsulates these modules in the form of specifically designed XML file which describes the format and procedures of protected software modules. We extended standard execution environment with special security features and functions in order to load protected plug-ins. Our extended Secure Execution Environment supports standard security services and also network security protocols.

8. Formal Validation

In this chapter we consider formal verification and validation of our generic security objects. We evaluated individual generic security objects using threat model and Common Criteria. We qualitatively analyzed behavior of our generic security against various potential attacks and found that, using our methodology and approach, each generic security object provides comprehensive mechanisms in order to protect its code, states, attributes, operations and communication against potential attacks, Therefore, if binaries, instantiation, usage, attributes, states, and communications between objects are protected against potential attacks, then the complete security system designed and implemented using such objects is protected and verifiable.

8.1. Introduction

The basic objective of our security system is that each generic security object should be secure. Furthermore, generic security objects should not have any security breaches and weaknesses that could be exploited by attackers. Various standards, procedures and metrics exist in order to qualitatively and quantitatively measure security of the systems. We used Common Criteria and Threat Model techniques to evaluate security and assurance level of our generic security objects. Common Criteria provide comprehensive procedures for security functional requirements and security assurance requirements, while threat model provides a systematic procedure to identify assets, threats, and their mitigation. So a combination of both techniques provides better evaluation and validation procedure of our security system.

For evaluation of our security system, we identified assets, points, and states of each of our generic security objects that could be potential target of an attacker. We also specified trust boundaries for each generic security object and identified various attributes where an attacker can launch attack(s). After that, we identified various attacks that can be launched against identified assets and operations. For security evaluation and validation of our methodology, we applied these attacks to our designed and implemented generic security objects in order to assess the resistance of our objects against these attacks. We also qualitatively described how our generic security objects resist various attacks.

Since the complexity of distributed applications is increasing, it is very difficult to ensure that the application is secure, trusted and reliable. Our evaluation scheme, which is based on deductive approach, provides effective and better solution for evaluation of complex security systems. In this chapter we also established that if an individual object is secure and resists various attacks, then the security system which is developed using such objects, will provide the same level of security against those attacks.

8.2. Validation and Evaluation Model

We adopted threat modeling, which is a formal approach to assess security and risk of our generic security framework for cloud computing environments. Threat model provides a formal approach to classify potential threats that makes evaluation easy to understand. It also provides mechanism to assign priorities to each threat. By using this scheme, we identified assets and resources of each generic security object, potential attacks, weak points, and their mitigations. Furthermore, we used Common Criteria to specify security functional and assurance requirements of each generic security object.

The most popular attacks are considered in this modeling. They are: Spoofing, Tampering, Information Disclosure, Repudiation, and Elevation of Privilege. In the evaluation of our system, we extended threat list and included brute force attacks, replay attacks, and session hijacking attacks. We identified attacks for each generic security object from the above list and then we described below how our generic security objects protect themselves against each potential attack.

8.3. Formal Evaluation of Generic Security Objects

We analyzed each generic security object individually in this section and qualitatively analyzed its behavior and protection mechanisms against each potential threat.

8.3.1. SymmetricKey Object

Figure 8.1 shows various states and operations of the `SymmetricKey` object.

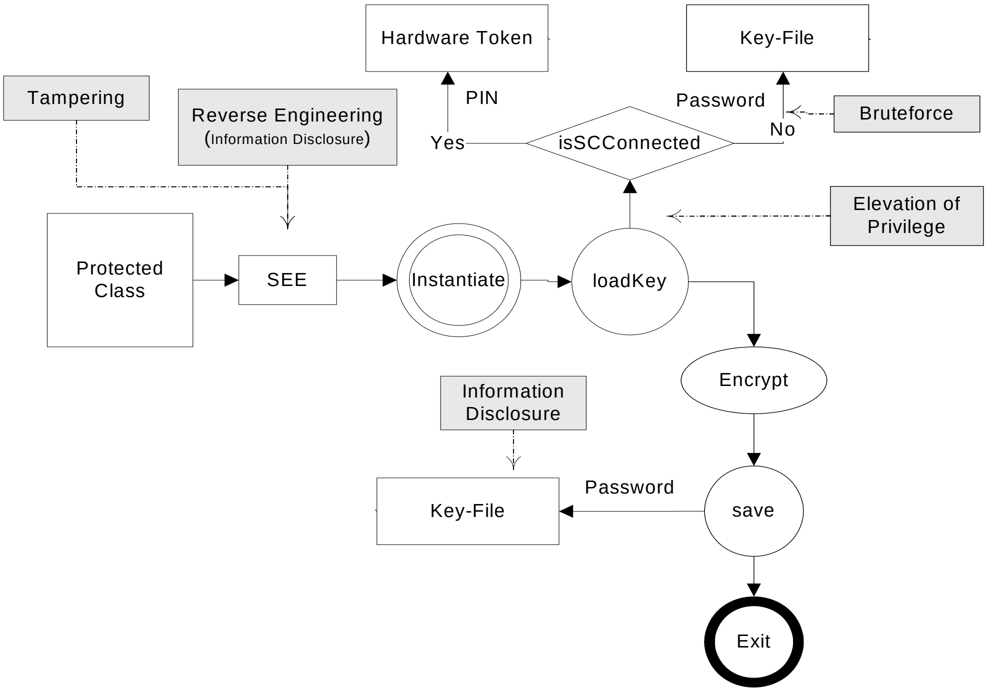


Figure 8.1. Various States and Operations of `SymmetricKey` Object and potential Attacks

We identified various possible areas where an attacker can attack. Furthermore, in Table 8.1 we described protection mechanisms against each attack.

Table 8.1. Evaluation and Validation of `SymmetricKey` Object

No.	Threat	Solution/Mitigation
1	Attacker can tamper binaries to alter logic of *SymmetricKey* object (Tampering).	Binaries of this object are digitally signed by the Product Manager which is verified by the Secure Execution Environment (SEE) before instantiating. During this process any modification can be detected, so our solution is resistant against tampering of binaries.
2	Adversary accesses the binaries for reverse engineering (Information Disclosure)	Our binaries are encrypted using *public-key* corresponding to user's certificate which protects this object against illegal reverse engineering and code analysis. The object can only be decrypted using *private-keys*, which is accessible to authorized user using valid password and *DistinguishedName*.
3	Adversary accesses symmetric key from the key-file (Brute force)	If symmetric key is stored in the key-file protected using password that is low assurance system. A brute force attack can be launched, but if it is configured with a smart card, i.e. high assurance system, then this object automatically blocks smart card after three wrong attempts. This policy protects *SymmetricKey* object against brute force attacks.
4	Adversary accesses symmetric key without authorization (Elevation of Privilege)	Symmetric key is stored in the key-file accessible only to authorized users using valid *password*. If the object is used in the high assurance system then an user provides PIN in order to authenticate on smart card to access Symmetric Key.
5	Adversary reveals symmetric key (Information Disclosure)	Symmetric key is either protected with *password* in the Key-file or it is stored in a smart card which in only accessible to a valid PIN holder.

8.3.2. AsymmetricKey Object

Figure 8.2 shows various states and operations of *AsymmetricKey* object along with various attacks. We describe solutions against various attacks shown in Table 8.2. Table 8.2 shows that our generic security object for asymmetrickey cryptography is completely protected and secure. Furthermore, we also verify that it performs every operation after proper authentication and authorization.

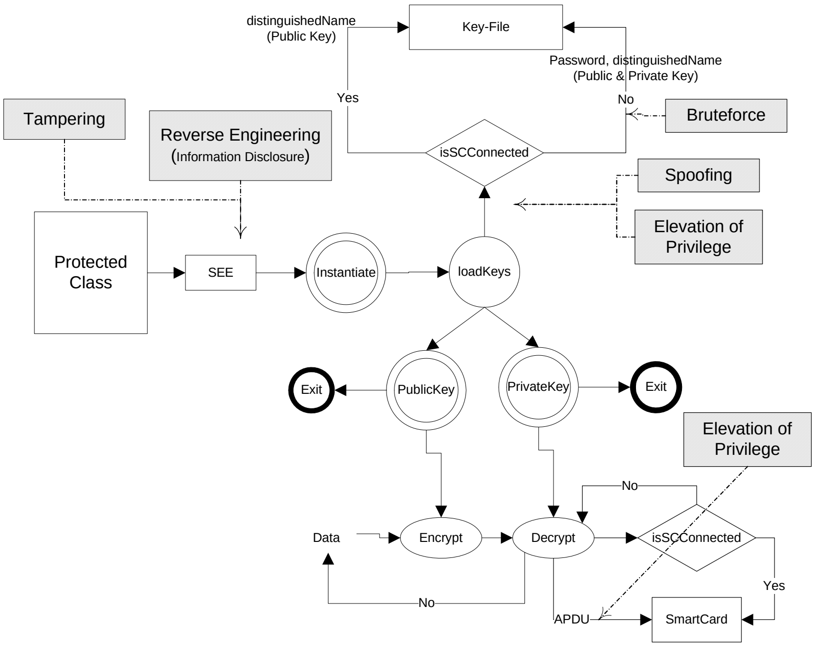


Figure 8.2. States, Operations and Threat Model of The `AsymmetricKey` Object

Table 8.2. Evaluation and Validation of `AsymmetricKey` object

No.	Threat	Solution/Mitigation
1	Attacker can tamper binaries to alter logic of `AsymmetricKey` object (Tampering)	Binary of this object is digitally signed by the Product Manager which is verified by the Secure Execution Environment (SEE) before instantiating. So, this protects binaries of `AsymmetricKey` object against binaries tampering.
2	Adversary accesses the binaries for reverse engineering (Information Disclosure)	Our binaries are encrypted using *public-key* corresponding of user's certificate which protects this object against illegal reverse engineering and code analysis.
3	Adversary accesses private key (Asymmetric key) from key-file (Brute force)	Private key is stored in key-file, in low assurance system, which is protected using password and mapped with public key. While, public key is mapped with `DistinguishedName` and can only be accessible, if user provides valid `DistinguishedName` and password. If `AsymmetricKey` object is using smart card, high assurance policy, then our object does not permit to extract private key from smart card, so in this case brute force attack is not effective.

4	Adversary accesses Asymmetric keys without authorization (Elevation of Privilege)	In the low assurance systems, asymmetric keys are stored in the key-file accessible only to authorized users using `password` and `DistinguishedName`. If a smart card is connected, then user provides PIN in order to get access to the smart card to the use asymmetric keys.
5	Adversary reveals private key which is part of Asymmetric keys (Information Disclosure)	In the low assurance system, private key is stored in Key-file encrypted using password so it is protected. On other hand, if a smart card is connected, then it is stored in a smart card, accessible only to a valid PIN holder.

8.3.3. Digital Signature

Figure 8.3 shows various states, operations and attributes of the `DigitalSignature` object along with various attacks. This is complex object and it instantiates `Hash` object and `AsymmetricKey` objects for various functions of digital signature. We already proved that `AsymmetricKey` object is tested and secure, so in this section we only evaluated `DigitalSignature` object, as described in Table 8.3.

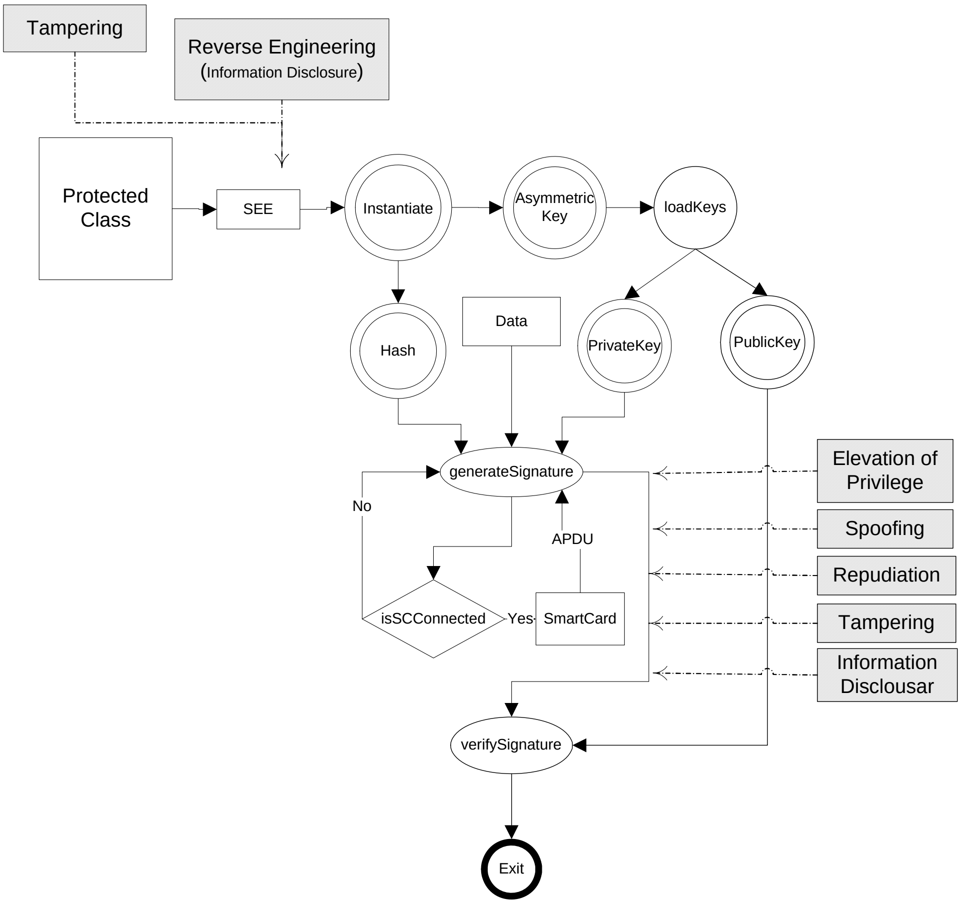


Figure 8.3. Various potential Threats of The `DigitalSignature` Object

Table 8.3. Evaluation and Validation of The `DigitalSignature Key` Object

No.	Threat	Solution/Mitigation
1	Attacker can tamper binaries of the object to alter logic of `DigitalSignature` object (Tampering).	Binary of this object is digitally signed by the Product Manager which is verified by the Secure Execution Environment (SEE) before instantiating. During this process, any modification made by adversary can be detected.
2	Adversary accesses the binaries of `DigitalSignature` object for reverse engineering (Information Disclosure)	Binaries of this object is encrypted using *public-key* corresponding to user's certificate, so binaries of this object are protected against illegal reverse engineering and code analysis.
3	Adversary illegally tries to generate signature (Evaluation of Privilege)	`DigitalSignature` object uses `DistinguishedName` and password in order to access private key stored in the key-file. If `DigitalSignature` performs cryptographic functions using smart card, then a valid PIN is used as authorization token to open smart card in order to encrypt hash value for signature generation.
4	Adversary accesses private key to generate fake digital signature for impersonation (Spoofing)	`DigitalSignature` object uses valid `DistinguishedName` and password as authentication token to extract private key from the key file. If `DigitalSignature` object is used with a smart card, then PIN is used as authentication token to open smart card.
5	Sender denies his/her message (Repudiation)	The sender is authorized and authenticated to generate signature, so he/she cannot repudiate.
6	Adversary alters message (Tampering)	During the digital signature process, a message is hashed using standard `Hash` algorithms. Tampering of a message can be detected using digital signature verification process.
7	Adversary tries to extract actual hashed value in order to generate fake messages (Information Disclosure)	Hashed value is encrypted using private key and only authenticated and authorized user can generate it.

8.3.4. PKCS7 and SMIME objects

In this section we analyze various potential attacks on *PKCS7* object as shown in Figure 8.4 and their mitigations. As *SMIME* is also an extension of the *PKCS7* object, the same evaluation criteria and threat model, as shown in Table 8.4, is used to evaluate *SMIME* object.

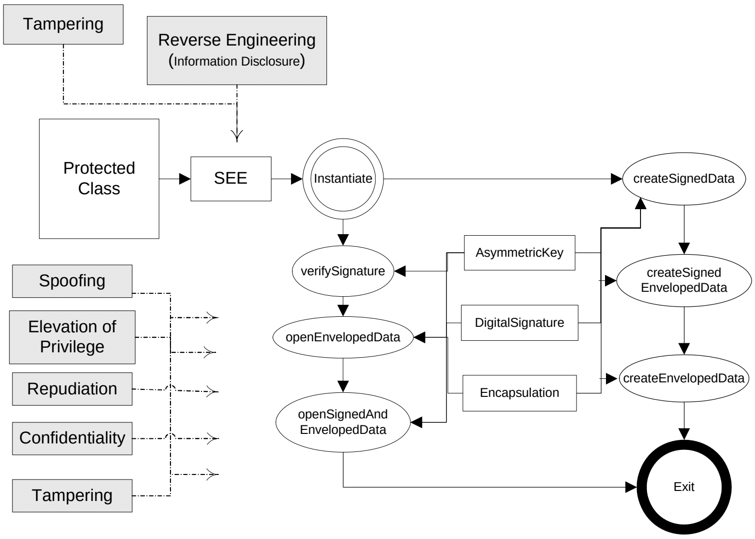


Figure 8.4. Identification of Assets and Operations of The *PKCS7/SMIME* Objects along with Potential Attacks

Table 8.4. Evaluation and Validation of The *PKCS7/SMIME* Object

No.	Threat	Solution/Mitigation
1	Attacker can tamper binaries to alter logic of the *PKCS7* object (Tampering).	Binary of this object is digitally signed by the Product Manager, which is verified by the Secure Execution Environment (SEE) before instantiating. During this process any modification of binaries can be detected.
2	Adversary accesses the binaries of the *PKCS7* object for reverse engineering (Information Disclosure)	Our binaries are encrypted using *public-key* corresponding to user's certificate, what protects this object against illegal reverse engineering and code analysis.
3	Adversary accesses encapsulated data by impersonating as a valid user (Spoofing)	*PKCS7* object uses our *DigitalSignature*, so only valid users with valid identity and password, or with PIN can create signed data and open enveloped data.

4	Adversary accesses security credentials and data from encapsulated data (Evaluation of Privilege)	*PKCS7* object uses our *DigitalSignature* object for creating *PKCS7SignedData* of input data, so the same authorization mechanism is applied to the *PKCS7*/*SMIME* object. In order to open *PKCS7EnvelopedData*, only authorized recipients which have valid private key and identity (*DistinguishedName*) can open it.
5	Sender denies his message (Repudiation)	The message is signed with sender's private key corresponding to the digital signature certificate, so sender cannot repudiate, because he/she is authorized to access private key.
6	Adversary alters message (Tampering)	In order to create *PKCS7SignedData* and *PKCS7SignedAndEnvelopedData*, our *PKCS7* object uses *createSignature* functions of the *DigitalSignature* object in order to provide data integrity. Any modification made of encapsulated data by attacker can be detected in digital signature verification phase. For *PKCS7EnvelopedData*, our system does not guarantee the integrity of data, because in the standard, it is only used for data confidentiality.
7	Adversary tries to extract actual data from the *PKCS7* encapsulated packet (Information Disclosure)	*PKCS7EnvelopedData* and *PKCS7SignedAndEnveloped* data is protected using the public key of the recipient, so only recipients who have valid private key can open it.

8.3.5. Strong Authentication Protocol

In Table 8.5 we describe various attacks and our counter-measures in order to protect our strong authentication protocol.

Table 8.5. Evaluation and Validation of The Strong Authentication Protocol

No.	Threat	Solution/Mitigation
1	Adversary tries to access password (Password compromise)	Our strong authentication protocol uses two-factor authentication. It uses challenge and digital signature to authenticate both parties, what eliminates the threat of password compromising, because it does not use password as authentication token.
2	Adversary tries to impersonate (Masquerade)	Our strong authentication protocol uses private key (stored in the key file) or smart cards to generate digital signature which is

		computationally infeasible for adversary to impersonate as valid user.
3	Adversary can use previous authentication messages (Replay attacks)	Since our strong authentication is based on random numbers challenges generated for every new authentication request, they cannot be reused by attackers as authentication token at a later time. Our strong authentication server simply discards theses messages.
4	Signing of pre-defined data	Our strong authentication protocol uses random number challenges, which is generated for each authentication request. It prevents signing of pre-defined data attack, because signature of pre-defined data is the same for every authentication request.
5	Adversary can present valid certificate (Elevation of Privileges)	Our strong authentication protocol does not only rely on the successful verification of digital signature certificates, but also validates its identity from the IDMS what ensures that valid domain user is performing strong authentication.
6	Adversary can impersonate as Strong Authentication Server (Destination Authentication Attack)	Our strong authentication protocol uses mutual authentication protocol, so the client also verifies the servers' digital signature certificate and asserts its identity from the IDMS. This step prevents our client for performing strong authentication with other strong authentication servers, which do not belongs to user's domain.
7	Adversary can tamper SAML ticket (Tampering)	After successful authentication, our Strong Authentication Server consults XACML Policy Server in order to issue SAML Ticket to a client. SAML Ticket is signed by the XACML Policy Server, so any change in SAML Ticket can be detected in the SAML Ticket verification phase.

8.3.6. Single-Sign-On Protocol

Various attacks on Single-Sign-On protocol and their counter measures are described in Table 8.6.

Table 8.6. Evaluation and Validation of The Single-Sign-On Protocol

No.	Threat	Solution/Mitigation
1	Adversary modifies SAML Ticket (Tampering)	The SAML Ticket is digitally signed by XACML Policy Server, so any modification in

		SAML Ticket can be detected using SAML Ticket verification and validation phase.
2	Adversary impersonates using valid SAML Ticket of another user (Masquerade)	The user signs SAML Ticket which provides source authentication. In addition, with the combination of secure session protocols, our system is strongly protected against impersonation attack.
3	Adversary impersonates using valid SAML Ticket of another user (Replay Attack)	The user signs SAML Ticket which provides source authentication. In addition, our Secure Session Protocol compares distinguished name of digital signature certificate with distinguished name of key exchange certificate of SAML Ticket owner, and it uses random number as session ID which protects our system from Replay Attack.

8.3.7. Secure Sessions Protocol

We qualitatively analyzed our secure session protocol. It is designed for secure communication between two components of our security system. Our threat model for this generic security protocol and their countermeasures are described in Table 8.7.

Table 8.7. Evaluation and Validation of The Secure Session Protocol

No.	Threat	Solution/Mitigation
1	Adversary tries to access session symmetric key (Information Discloser)	Secure Application Server encrypts session symmetric key using public key corresponding to the key exchange certificate of user. It is computational infeasible for adversary to decrypt encrypted message.
2	Adversary tries to hijack session (Session Hijacking)	In Session establishment process both Secure Application Server and user verifies certificates and associated identities from IDMS. After that they use public key corresponding to key exchange protocol to encrypt messages. These steps prevent our system from session hijacking.
3	Adversary tries to extract information from message (Message Confidentiality)	The messages between Application Server and user are encrypted with session key, which is only shared between Application Server and user.
4	Adversary can alter messages (Tampering/Message Integrity)	All messages exchanged between Secure Application Server and users are digitally signed, so any alteration in messages at transport level can be deducted in verification phase.

8.3.8. SAML Authorization Protocol

The SAML Authorization protocol is designed according to the SAML standard with additional security features. Different potential attacks on SAML Authorization protocol and their countermeasures are described in Table 8.8.

Table 8.8. Evaluation and Validation of The SAML Authorization Protocol

No.	Threat	Solution/Mitigation
1	Adversary tries to access the contents of SAML Authorization messages (Information Discloser)	SAML Authorization object encrypts its consents using the public key corresponding to the key exchange certificate of recipient (if it is *SAMLAuthorizationRequest* then recipient will be XACML Policy Server. if it is *SAMLAuthorizationResponse* then it will be PEP server). It is computational infeasible for adversary to decrypt encrypted message to extract contents, so the only authorized recipient can view the contents.
2	Adversary modifies the contents of SAML Authorization Protocol (Tampering)	SAML Authorization object digitally signs *SAMLAuthorizationRequest* and *SAMLAuthorizationResponse* messages using the private key corresponding to the digital signature certificate of message generator. Any modification in message can be detected during the signature verification process.
3	Adversary sends fake message in order to gain illegal access of resources	The messages between PEP server and XACML Server are digitally signed which provides a notion of source authentication. Any fake message generated by adversary can be identified during the digital signature verification phase.
4	Impersonation	All SAML Authorization messages exchanged between PEP Server and XACML Policy Server are encrypted and digitally signed. So adversary cannot use these messages for reply attacks and impersonation because only authorized users with required security credentials can process these messages.

8.4. Validation of Methodology and Results

We designed and implemented our generic security objects using our described methodology. We qualitatively analyzed behavior of each object against potential threats in above sections. The aggregated results of our evaluation of generic security objects are shown in Table 8.9. We found that our generic security objects resist against software tampering and reverse engineering attacks because binaries of each object is digitally signed and encrypted.

We also found that our generic security objects resist to tampering, information disclosure, unauthorized access and non-repudiation attacks because attributes of each generic security object are stored securely and they are manipulated in secure execution environment. Furthermore, states, usage and communication between objects are protected and they are only accessible to authenticated and authorized users. So our generic security objects are protected against information disclosure, illegal access, spoofing and brute force attacks.

Table 8.9. Summarized Results of Evaluation and Validation of our Generic Security Objects

Generic Security Objects	Threats and Attacks											
	Binaries Tampering	Binaries Analysis	Tampering	Information Disclosure	Spoofing (Masquerade)	Elevation of Privilege	Brute force	Non-Repudiation	Password Compromise	Replay attacks	Signing of pre-defined data	Session Hijacking
SymmetricKey Object	×	×	-	×	×	×	×	-	×	-	-	-
AsymmetricKey Object	×	×	-	×	×	×	×	-	×	-	-	-
DigitalSignature Object	×	×	×	×	×	×	-	×	-	-	-	-
PKCS7 and SMIME Object	×	×	×	×	×	×	-	×	-	-	-	-
Strong Authentication Protocol	×	×	×	×	×	×	-	×	-	×	×	-
Single-Sign-On Protocol	×	×	×	×	×	×	-	×	×	×	-	×
Secure Session Protocol	×	×	×	×	×	×	-	×	-	-	×	×
SAML Authorization Protocol	×	×	×	×	×	×	-	×	-	×	-	-

We validated that states of each of our generic security objects are protected, their usage is protected, their actions are protected and their communications with other objects are also protected. Therefore, security system designed using such objects is completely secure, because our generic security objects provide complete set of security services which are required for implementing secure applications for cloud computing environments.

8.5. Performance Evaluation of Generic Security Objects

We quantitatively evaluated the performance of some of our generic security objects and compared their execution time with the objects of most popular Crypto Services Providers. We designed and implemented test-cases using our Generic Security Provider, Java Crypto Provider (JCP) and MS Crypto Services Provider (MS-CSP) using .NET (C#). After that we calculated execution time in order to compare performance of various security aspects. In this section we described summarized results of our experiments as shown in Figure 8.5. Complete information about the test-bed, hardware specifications and evaluation results are described in Appendix D. In these experiments, we considered both basic and advanced cryptographic functions. Basic functions are: symmetric key cryptography, asymmetric key cryptography and hash functions. The advanced and complex security functions are: certificate generation, encapsulation standards like PKCS7SignedData cryptographic standard and SMIME standard.

For symmetric key cryptography, we created an instance of our *SymmetricKey* object which encrypts and decrypts a predefined data using DES algorithm (with PKCS7 padding). We also implemented same functions using standard Java Crypto Provider and MS Crypto Services Provider. After evaluating, we found that our generic security object takes 16 milliseconds (ms) to encrypt and decrypt data using symmetric key cryptography. We also calculate that the Java Crypto Provider takes 377 ms and .NET Provider takes 3 ms for same functions. The analysis shows that our generic security object for symmetric key is efficient than Java Crypto Provider but the .NET Provider provides better results for symmetric key cryptography.

For asymmetric key cryptography, we created asymmetric key pair (RSA-1024) using our *AsymmetricKey* object, standard Java Crypto Provider and .NET Provider. We used public key to encrypt predefined text and private key to decrypt cipher text. After calculating execution time, we found that our generic security object takes 33 ms, Java Crypto Provider takes 328 ms, and .NET Provider takes 290 ms. The experiment results of asymmetric key cryptography show that our designed solution is efficient than the competitors.

For performance evaluation of hash functions, we created an instance of our *Hash* object in order to compute hash value using SHA-1 algorithm. We also implemented test cases using Java Crypto Provider and .NET Provider for hash computation. We compared execution time and found that every provider takes less than 1 ms.

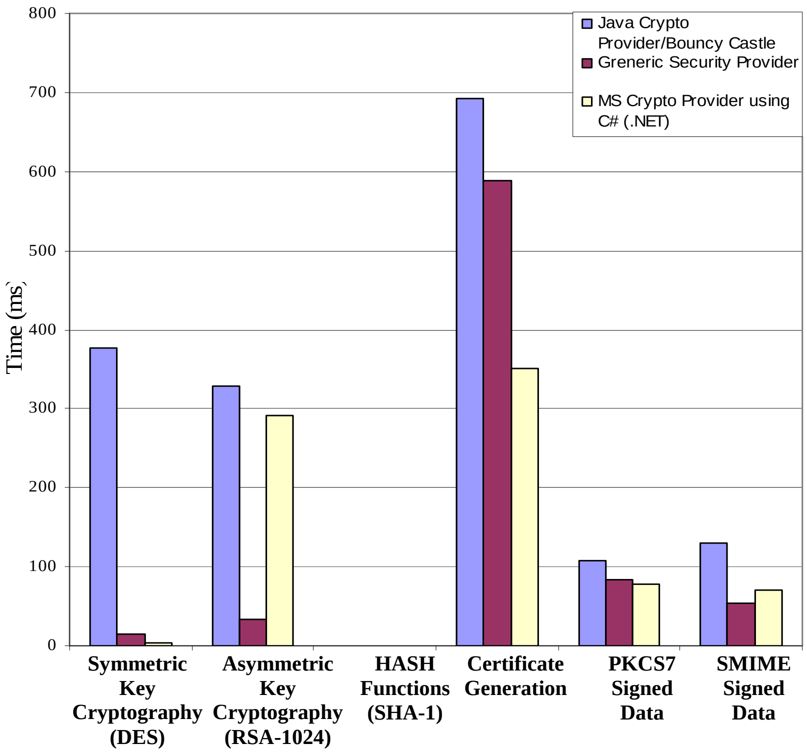


Figure 8.5. Performance comparison of security functions

We also implemented test-cases for complex security functions using above mentioned providers to compare their execution time. We considered X509Certificate, PKCS7 and SMIME objects. After evaluating, we found that our generic security object `Certificate` takes 589 ms to generate X.509 Certificate while Java Crypto Provider takes 692 ms, and .NET Provider takes 281 ms. For PKCS7, our generic security object *`PKCS7`* takes 84 ms to generate PKCS7SignedData while Java Crypto Provider and .NET Provider takes 108 ms and 78 ms respectively. For SMIME, our generic security object *`SMIME`* takes 53 ms, Java Crypto Provider takes 130 and .NET Provider takes 63 ms in order to generate digital signature of E-mail letters using SMIME standard.

After analyzing and comparing the results of our experiments we established that our generic security objects are efficient and provides better results for Java applications. We also analyzed and compared security functions of MS Crypto Services Provider and found that some of our security functions are efficient and some are slightly slower than our Generic Security Provider.

In this comparison, we also compared lines of code and found that using our generic security objects, a developer can implement security functions using in fewer lines of code (as shown in Appendix D) than Java Crypto Provider and MS Crypto Services Provider. Therefore, it justifies our claim that our methodology decreases the complexity of secure software development; they are easy to understandable and support rapid development methodology.

Part II: Security Architecture and Applications

9. CryptoNET Architecture

In this chapter we describe the objectives of our framework for protection of IT resources, messages, operations and software modules. The framework is structured as a layered model. We elaborate security features, functions and components of each layer. In addition, in this chapter we describe the roles and responsibilities of each component.

9.1. Overview of CryptoNET Architecture

CryptoNET is designed based on simple principle that all applications' resources are protected using encryption. If all the resources, operations, attributes, messages and software modules are maintained and manipulated in a cryptographic environment, then all workstations and servers are protected against mobile code, malicious software, intruders and incorrect operations. The design of CryptoNET is based on generic security objects, components, well established secure technologies, and security standards. The components of CryptoNET are categorized into the following four groups:

- The first group comprises security engine components which are basic components of our security system. These components are: Security Provider, Security Protocols, Secure SDK and Secure Execution Environment. These components are complete in terms of their functionality, they transparently handle security credentials, and they are easy to integrate with other components and applications. The details of each component were described in Chapter 4, Chapter 5 and Chapter 6 respectively.
- The second group comprises various application components. These components implement security functions using a single Security Provider. Furthermore, application components use security protocols for various network services. All application components are protected and executed in our secure execution environment.
- The third group of components comprises application servers and domain level security servers. The components of this group provide security services to the components of the second group. Domain level security servers are part of our cloud environment and provide security services and credentials to all components that exist in a domain.
- The fourth group of components comprises global security infrastructure servers which are deployed in our cloud computing environment. These servers support standard protocols in order to develop trust between domains. The roles and functionality of these servers are described in Section 9.2.4.

The components of the last three groups are structured into four layers. The components of the first group are enabling components. Therefore, they exist at each layer for providing the same set of security services and network security protocols to all other components.

CryptoNET is complaint with FIPS 201 (PIV) smart cards for cryptographic functions, local and remote authentication, authorization, Single-Sign-On and secure communication. In addition, we designed additional security applets for the CryptoNET system. The detailed description of security applets was given in Chapter 4.

9.2. Layered Model of the CryptoNET Architecture

The model of CryptoNET architecture is structured into four layers. Various components located at individual layers, together with their interconnecting security protocols, comprise security architecture of our CryptoNET system. Layering of CryptoNET components is based on the following principles:

- Components at an upper layer provide services to the multiple components at the lower layer;
- Component at upper layer link the same types of components at lower layers in different domains;
- Components at an upper layer complement functionality of components at lower layers.

Therefore, based on these principles, it appears that our CryptoNET architecture is hierarchical. This means that it possesses properties of interconnectivity of multiple components, scaling, expansion, and interoperability.

The bottom layer of our CryptoNET architecture is Integrated Secure Workstation which provides various security features and services to users. The next layer is Secure Application Servers. It comprises Secure E-Mail Server, Secure Web Server, Secure Library Server, and Secure Software Distribution Server. Above that is Credential Servers layer. It contains various components in order to distribute security credentials, provide authentication and authorization services to various components exist in a domain. These are: Issuing PKI Server, XACML Policy Server, Strong Authentication Server (SA Server) and Identity Management Server (IDMS). In our system, these three layers are deployed inside an organization (Security Domain) and administrated by a Security Manager. The fourth layer is global security Infrastructure layer. It contains PKI Top Server, PKI Policy Server and SMI Server. The components of this layer used for developing relationships and trust between domains. These servers are administered by Network Security Administrators and normally deployed outside of an organization or a domain.

9.2.1. Layer 1: Integrated Secure Workstation

Integrated Secure Workstation (ISW) is located at the bottom layer of the CryptoNET architecture, as shown in Figure 9.1. The workstation comprises various security components that provide security services to end-users. These components are: Secure Station Manager, Secure E-Mail Client, Secure Documents Manager, and Secure Web Browser extended with proxy. The workstation protects local IT resources, messages and operations for multiple applications. It implements security functions using Security Provider, which provides standard and extended security functions and features. Furthermore, Security Provider transparently handles security credentials and integrates security hardware tokens, like smart cards, if available. The purpose of smart cards is to perform two-factor authentication. The cards support cryptographic functions, operating systems' authentication, application login, strong authentication, Single-Sign-On, and secure sessions. In addition, ISW supports network aspects of its applications by connecting to Secure E-Mail Server, Secure Web Server, Secure Library Server, Secure Software Distribution Server and to servers located in the global security infrastructure.

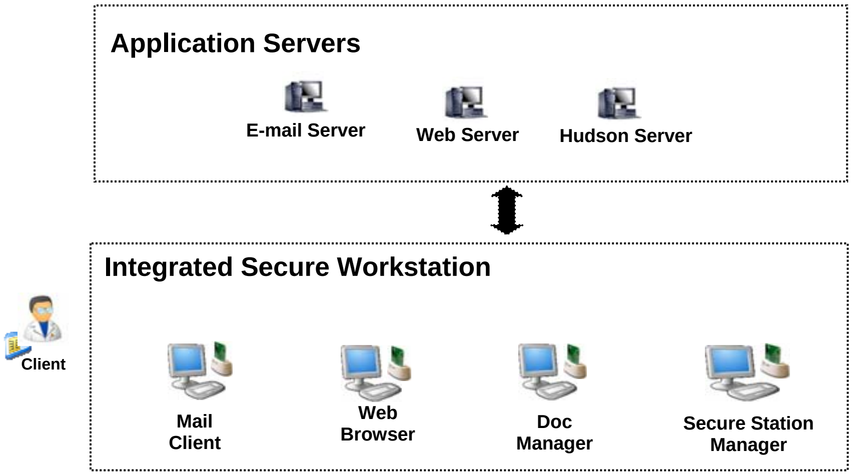


Figure 9.1. Components of Integrated Secure Workstation

The following are the main components and functions of the workstation:

Secure Station Manager

Secure Station Manager (SSM) is an important component of this layer. Functionally, it is equivalent to the Windows Explorer. This component performs various security management and operational functions. The end-user (Security Manager) uses this component to register new users in a domain. The Security Manager also issues smart cards to the newly registered users which are personalized with required applets and credentials (see Chapter 4).

One of the most important functions of Secure Station Manager is transparent handling of certificates. This component transparently checks the availability of current users' certificates. If they are not available, SSM generates three self-signed certificates. These certificates are digital signature certificate, key-exchange certificate, and non-repudiation certificate. Secure Station Manager also detects smart card. If smart card is installed, then Secure Station Manager generates key-pairs in a smart card and also stores certificates in it. If Local CA Server (PKI Issuing Server) is configured, then Secure Station Manager requests and receives three certificates from Local CA Server and overrides self-signed certificates. In addition, Secure Station Manager facilitates end-users to perform various certificate management functions.

Using Secure Station Manager, users can perform standard file management functions. It generates local-resource-symmetric-key to encrypt local files and IT resources, using standard cryptographic format – PKCS#7. It stores local-resource-symmetric-key in a smart card, if it is installed. Otherwise, it stores it in key-file, protected by public key corresponding to key-exchange certificate of the current user. Furthermore, it also creates encrypted AuditLog file and decrypts it upon current user's request for inspections of its entries. The end-user can also use this module to configure various servers for other applications.

Secure E-Mail Client

Secure E-Mail Client performs standard e-mail functions: sending and receiving secure e-mail using Secure/Multipurpose Internet Mail Extensions (S/MIME) standard. It uses standard mail transport protocols: Simple Mail Transfer Protocol (SMTP) and Post Office Protocol-3 (POP3). Secure E-Mail Client stores contacts into the address book which are encrypted. For encryption of the address book entries, Secure E-Mail Client uses local-resource-symmetric-key. It encrypts address book entries before storing them into address book and decrypts them before displaying on the display panel. Furthermore, this application is connected with Secure E-Mail Server to upload and download protected address book for recovery and portability.

Secure E-Mail Client uses only Signed and/or Enveloped e-mail letters. It also sends and receives only E-mail to/from authorized users. In order to provide authorization, each Secure E-Mail Server applies authorization polices, specifying other authorized "Sending To" and "Receiving From" Secure E-Mail Servers. The complete functionality of Secure E-Mail System is explained in Chapter 11.

Secure Web Browser

Secure Web Browser is a component located in front of a standard Web Browser in order to provide various security functions. This component performs standard exchange information with Secure Web Server using HTTP protocol. It creates secure session with the Secure Web Server to fetch Web contents. Furthermore, Secure Web Browser encrypts cookies and browsing history in order to provide user-privacy. Key features of this component are Single-Sign-On, secure commutation, XACML authorization policies, and management of cryptographic keys. The detail operations of the Secure Web Browser are explained in Chapter 12.

Secure Documents Manager

Secure Documents Manager is an extension of the OpenOffice application with security functions. OpenOffice provides standard functions to end-users, like manipulation of documents, spreadsheets, image editing and presentations. This component provides security features like protection of files in PKCS#7SignedAndEnvelopedData format, sharing of files in a group environment, management of cryptographic group-keys, enforcement of section level XACML authorization policies, SAML based Single-Sign-On and secure communication. In order to share files in a group environment, this component is connected to the Secure Library Server and also to security servers in our global security infrastructure. The detail operations of the Secure Documents Manager are described in Chapter 13.

9.2.2. Layer 2: Secure Application Servers

The second layer of the CryptoNET system is Secure Application Servers layer, shown in Figure 9.2. This layer comprises Secure E-Mail Server, Secure Web Server, Secure Library Server, and Secure Software Distribution Server. Each server transparently acquires the required certificates from the Local Certification Authority (LCA) Server. In the CryptoNET deployment environment, these servers are managed by a Security Manager. Some of application servers' functions are mentioned in the previous sections, so in the following sections we will briefly mention the features and roles of each server in our framework. The detail functions and their operations are explained in subsequent chapters.

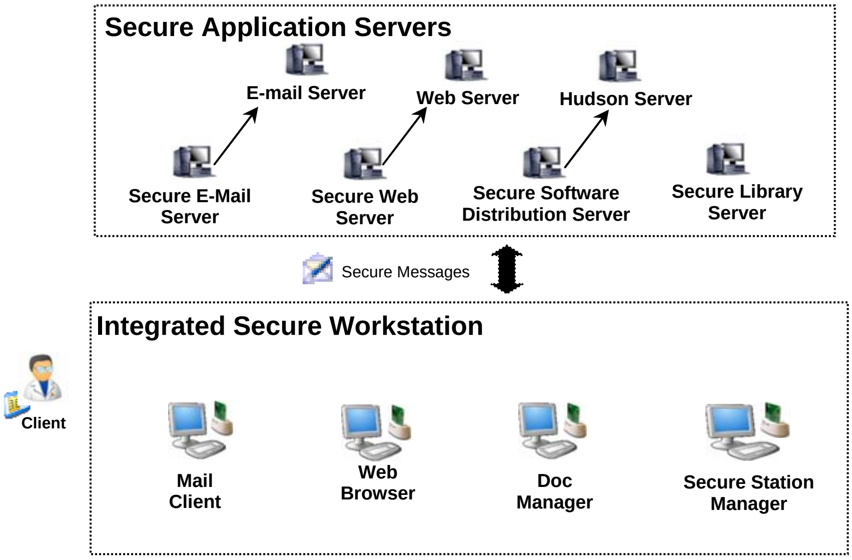


Figure 9.2. Secure Application Servers and Components of The Integrated Secure Workstation

Secure E-Mail Server

Secure E-Mail Server is a proxy server. It is located between Secure E-Mail Client and standard E-mail Server. The Server is responsible to forward secure E-mails to standard E-mail server and fetches secure E-mails on behalf of E-mail client. In addition, this component provides extended security functions to Secure E-Mail Clients like protection and management of address books, management of address-book-symmetric-keys, handling of attachments, confirmation messages, enforcement of authorization policies, Single-Sign-On, and secure communication. In addition, this Server also cooperates with Secure E-mail Infrastructure servers for registration and validation of domain addresses in order to protect user's inboxes from SPAMs.

Secure Web Server

Secure Web Server is located in front of a standard Web Server as its security proxy. It generates encrypted Web pages and deploys them at a standard Web server. It transparently integrates secure execution environment for Web server in order to process encrypted web pages. Secure Web Server is responsible for sending/receiving protected HTTP pages, handling of certificates, enforcement of XACML authorization policies, and secure communication with Secure Web Browser.

Secure Library Server

In the CryptoNET system we also introduced Secure Library Server. This Server provides security services for protection and distribution of secure documents in group environments. It enforces XACML authorization policies in order to provide access of documents and sections of documents to authorized users. It is also responsible to manage and distribute cryptographic group keys using the Group Secure Association Key Management Protocol (GSAKMP). Secure Library Server supports Single-Sign-On protocol, secure communication protocol, transparent handling of certificates, and management of groups.

Secure Software Distribution Server

Secure Software Distribution Server interacts with Hudson Server and Secure Web Server to generate and distribute secure software modules to authorized users. It generates executable binaries and then encapsulates them in our designed XML file. Secure Software Distribution Server uses authentication and authorization of Secure Web Server in order to distribute protected binaries to authorized users. In addition, this component also digitally signs secure execution environment in order to execute protected software modules.

9.2.3. Layer 3: Security Management Servers

This section describes various Credential Servers, shown in Figure 9.3. These servers are in principle Security Management Servers in our CryptoNET architecture. They are deployed in our cloud computing environment and are responsible for distribution and management of certificates, creation and management of XACML policies, management of identities, and strong authentication services. These servers are based on well-established security standards and are interoperable with servers in other domains. The components of the CryptoNET architecture, located at the lower layers, are connected with these servers in order to obtain security services. In our framework, an instance of these Credential Servers must be deployed in a domain, but multiple instances may also be used.

PKI Issuing Server

In our system, PKI Issuing Server is also known as Local Certification Authority (LCA) Server. LCA Server is a standard Certificate Authority Server which issues and distributes X.509 certificates to all components in a domain. This Server may be configured as a Single Certification Authority, in that case it generates self-signed certificates. This Server may also be linked to the PKI in order to exchange certificates and support establishing trust between various domains. In this case, the upper level trusted certification authority issues the certificate of the Issuing PKI Server.

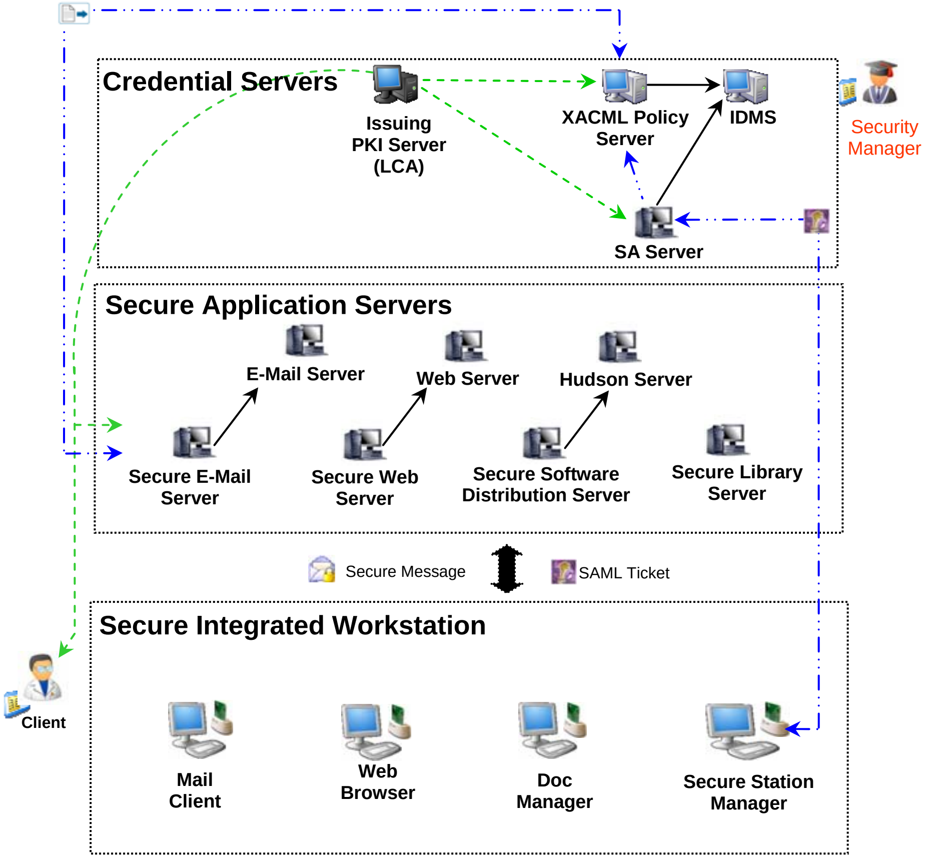


Figure 9.3. Credential Servers of the CryptoNET Architecture and Interactions between them

XACML Policy Server

XACML Policy Server is also known as Policy Decision Point (PDP). This Server supports management of groups, roles, XACML policies, and policy sets. The Server also creates and validates SAML tickets for the SSO protocol. In addition, this Server is also responsible to distribute XACML policies to various components for local decisions. The XACML Policy Server supports SAML Authorization protocol for evaluation of authorization policies. It also supports SAML Authentication protocol for Single-Sign-On.

Strong Authentication Server

Strong Authentication (SA) Server performs mutual strong authentication with clients using extended strong authentication protocol, which is FIPS-196 compliant protocol. The Server has also connection with XACML Policy Server to generate SAML Ticket for authenticated clients.

IDentity Management Server (IDMS)

Identity Management System is used to create, manage and delete identities in collaborative environments. This system uniquely identifies users in a global environment and provides identity verification services to other components.

9.2.4. Layer 4: Infrastructure Servers

This section describes various infrastructure level servers located in a cloud computing environment. Their main purpose is to synchronize security policies and functions across multiple domains. These servers are shown in Figure 9.4. The role of each server is explained in the following sections.

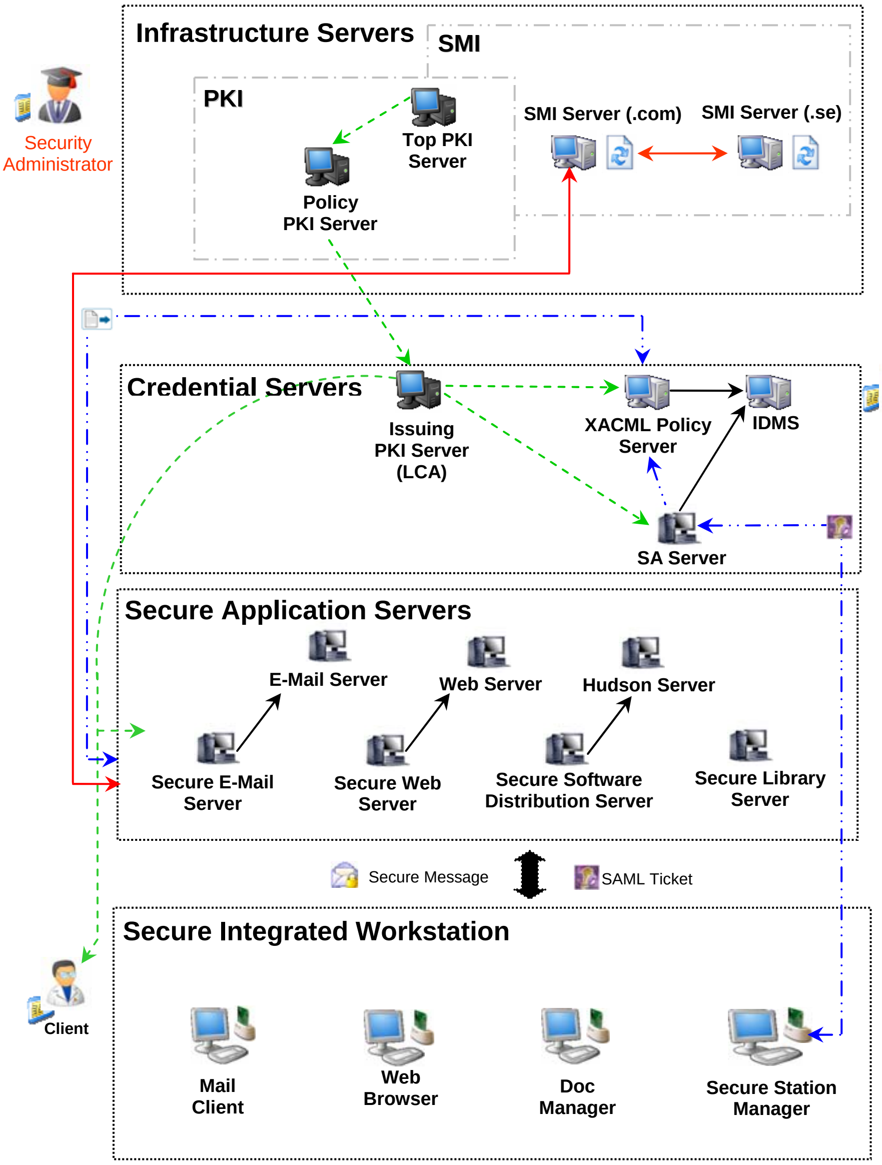


Figure 9.4. Layered Model of the CryptoNET System

PKI Top Server

PKI Top Server is the component of the global security infrastructure. PKI Top Server is a root CA server and it generates self-signed certificates. The Server also supports the concept of cross-certification in order to develop trust between two domains.

PKI Policy Server

Below PKI Top Server is PKI Policy Server. In our system PKI Top Server issues certificates to the PKI Policy Server and PKI Policy Server issues certificates to LCA Servers. The purpose of PKI Policy Server is to regulate, impose and enforce certificate policies to domain-level LCA Servers.

SMI Server

SMI Server is the component of our Secure E-mail System. The purpose of this Server is to register SEM servers and certify domain names. SMI Server also coordinates with other SMI servers to validate the domain names. We describe operations of the SMI Server in Chapter 11.

9.3. Summary

CryptoNET is a collection of enabling components, client components, application servers, and security servers. CryptoNET system is complete in terms of functions and security services that it provides. CryptoNET is structured in the form of four layers and each layer provides security services to other components using the same enabling components, so the same cryptographic engines are used by all applications. CryptoNET also supports global security protocols for interaction between various domains. In this architecture, credentials servers and global security infrastructure servers are part of our cloud computing environment. CryptoNET protects resources, operations, database attributes, messages and software modules without any intervention from end-users using well-established security standards and technologies.

10. Integrated Secure Workstation

In this Chapter we describe Integrated Secure Workstation which integrates various client components in order to protect IT resources. This workstation uses Security Provider for cryptographic functions and for transparent handling of security credentials and smart cards. It also uses security protocols for security of various distributed components. The Integrated Security Workstation is protected, so it is executed using our extended secure execution environment.

The Integrated Secure Workstation comprises several client applications: Secure Station Manager, Secure E-mail Client, Secure Web Browser, and Secure Documents Manager. Each component maintains and manipulates user's resources using single instance of enabling components and supports multiplatform operating systems.

10.1. Overview and Features of The Integrated Secure Workstation

Integrated Secure Workstation (ISW) comprises four secure client applications: Secure Station Manager, Secure E-Mail Client, Secure Documents Manager, and Secure Web Browser. It protects local IT resources, messages and operations across multiple applications using enhanced security functions. It implements security functions using Security Provider, which provides standard and extended security functions and features. Furthermore, Security Provider transparently connects to security hardware tokens, like smart cards, if available, which enables users to perform login, create signatures, and store security credentials in smart cards.

ISW also supports security protocols that are used to connect to components of our security infrastructure: LCA Server, IDMS, SA Server and XACML Policy Server. Furthermore, in each domain, ISW supports network aspects of its applications by connecting to Secure E-Mail Server, Secure Web Server, and Secure Library Server. Security features of those application servers are discussed in the subsequent chapters. To make it simple and understandable, we categorized security functions and features into two groups: common security functions and application–specific security functions. Common security functions are used across multiple applications, while application–specific security functions are used only by individual applications.

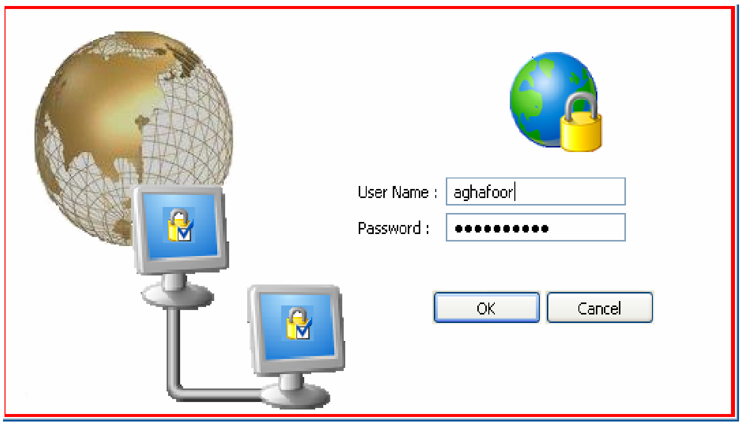
10.2. Common Security Functions

10.2.1. User Registration

In our system Security Manager (SM) with administrator's rights, registers users in the IDMS. In that process SM provides complete profile of users like username, address, telephone number, etc. In addition SM also specifies login-name, password, role and associated domain for each user. The password of the system is protected using local-resource-symmetric-key. In this system we are using two types of roles. One is *User* which can only perform client-side functions. The other one is *Security Manager*, who inherits capabilities of normal users, plus he/she can perform some additional tasks, like management of applications servers, registration of users, issuance of smart cards, etc. In our System, user's registration information is used to create a DistinguishedName for various certificates (see Section 10.2.3).

10.2.2. Local User Authentication

At start-up, the system activates generic login module, shown in Figure 10.1. This module loads Security Provider in order to provide security functions. During this process Secure Workstation creates connections with the IDMS Server and the LCA Server, if these servers are accessible. The module is generic in a sense that it uses user name and password, if smart card is not available; otherwise it requires PIN to authenticate to the card. Local user authentication based on smart cards is compliant to the FIPS 201 (PIV) standard [24], i.e. it supports PIN–only or PIN plus fingerprint authentication. After successful authentication, it displays generic graphical interface, as shown in Figure 10.2. In this phase, Integrated Secure Workstation also checks the presence of certificates. If certificates are not available, then it notifies user to request certificates from the LCA Server.



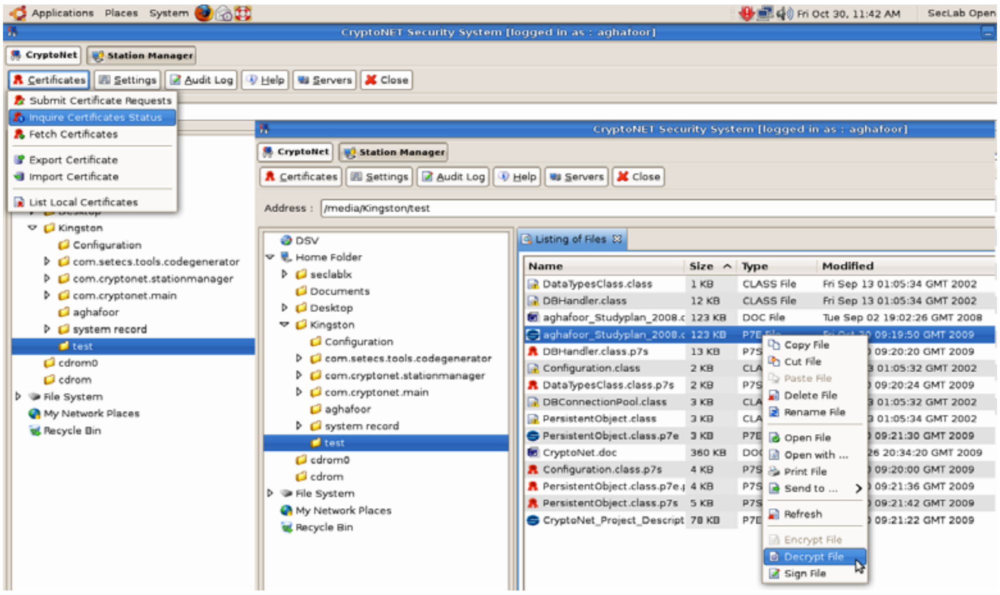
User Name : aghafoor

Password : ●●●●●●●●●●●

Figure 10.1. Generic Log-in Module for Local User Authentication

10.2.3. Handling of Certificates

One of the important functions of our Integrated Secure Workstation is transparent handling of certificates. ISW fetches current user's registration data from the IDMS Server and creates user's Distinguished Name, which is used for generation of the three self–signed certificates. These are digital signature, key exchange, and non-repudiation certificates. If connection to the LCA Server is established, then ISW requests and receives certificates from the CA Server. It stores certificates in a smart card, if it is connected; otherwise, it stores them in a local certificate database. In addition, various certificate management functions are available with Secure Station Manager, as explained in Section 10.3.1.



Name	Size	Type	Modified
DataTypesClass.class	1 KB	CLASS File	Fri Sep 13 01:05:34 GMT 2002
DBHandler.class	12 KB	CLASS File	Fri Sep 13 01:05:34 GMT 2002
aghafoor_Studyplan_2008.c	123 KB	DOC File	Tue Sep 02 19:02:26 GMT 2008
aghafoor_Studyplan_2008.c	123 KB	P7E File	09:19:50 GMT 2009
DBHandler.class.p7s	13 KB	P7S	09:20:20 GMT 2009
Configuration.class	2 KB	CLA	01:05:32 GMT 2002
DataTypesClass.class.p7s	2 KB	P7S	09:20:24 GMT 2009
DBConnectionPool.class	3 KB	CLA	01:05:32 GMT 2002
PersistentObject.class	3 KB	CLA	01:05:34 GMT 2002
PersistentObject.class.p7e	3 KB	P7E	09:21:30 GMT 2009
CryptoNet.doc	360 KB	DOC	26 20:34:20 GMT 2009
Configuration.class.p7s	4 KB	P7S	09:20:00 GMT 2009
PersistentObject.class.p7e.p	4 KB	P7S	09:21:36 GMT 2009
PersistentObject.class.p7s	5 KB	P7S	09:21:42 GMT 2009
CryptoNet_Project_Descript	78 KB	P7E	09:21:22 GMT 2009

Figure 10.2. Certificate Management Functions of Secure Station and View of Protected Files and Actions in Data Panel (Listing of Files). Running on Linux environment.

10.2.4. Single-Sign-On and Secure Communication

ISW supports Single-Sign-On protocol in order to gain access to various application servers using SAML Ticket. ISW transparently performs Single-Sign-On protocol, as described in Section 5.2.3, when it is required or invoked by client applications. After Single-Sign-On is successfully completed, the Secure Station establishes secure communication with a corresponding application server in order to exchange secure messages.

10.2.5. User Authentication with Application Servers

Some of application clients included in ISW may create connections with their corresponding application servers. In that case authenticated client checks the presence of SAML Ticket for a Single-Sign-On protocol. If SAML Ticket does not exist, then it performs Strong Authentication Protocol with the SA Server, as described in Section 5.2.2, in order to acquire SAML Ticket.

10.3. Application–Specific Security Functions

10.3.1. Secure Station Manager

Station Manager is an application equivalent to Windows Explorer, but extended with security. Using Station Manager users can perform standard file management functions like copy, cut, paste, rename file and folder, open file, etc. Station Manager generates a local-resource-symmetric-key to protect local files and other IT resources, using standard cryptographic format – PKCS#7, as shown in the front data panel of Figure 10.2. Station Manager stores local-resource-symmetric-key in the Security

Applet, if smart card is connected; otherwise it stores it in a key file, protected using public key from the key-exchange-certificate. Local file protection works as follows:

When a user right-clicks on a file to encrypt it, our system encrypts it using local-resource-symmetric-key. Our system also uses the same local-resource-symmetric-key to decrypt the encrypted file(s). Similarly, when end-user performs "*Sign*" function our system digitally signs a file using private key corresponding to the digital signature certificate. Digital signature generation is performed by a smart card, if it is installed; otherwise, software-based encryption is used. To encapsulate protected files and resources, we use *PKCS7SignedData* for signed resources, *PKCS7EnvelopedData* for encrypted resources, and *PKCS7SignedAndEnvelopedData* for signed and encrypted resources. Our system stores protected files with standard extensions: *p7s* for signed files, *p7e* for encrypted files, and *p7m* for signed and enveloped data.

If a user right-clicks on a protected file our system automatically detects encapsulation technique based on extensions of protected files. For example, if a user clicks a file which has *p7s* extension, then Station Manager automatically verifies digital signature using public key extracted from digital signature certificate. If extension of a file is *p7m*, then the system automatically verifies signature before decrypting the selected file using local-resource-symmetric-key.

In addition, Station Manager interacts with the Local CA Server in order to generate, fetch, verify and list certificates, as shown in the background panel of Figure 10.2. All certification functions are based on various certificate management standards. Station Manager also provides features for setting various servers and configurations. Furthermore, it also creates encrypted AuditLog and decrypts it upon user's request for inspection of its entries.

10.3.2. Secure E-Mail Client

Secure E-Mail Client, a component of the Integrated Secure Workstation, performs standard E-mail functions: sending and receiving secure E-mails using S/MIME standard. It uses standard mail transport protocols: SMTP and POP3. Secure E-Mail Client stores contacts into the address book. For their protection Secure E-Mail Client uses address-book-symmetric-key. Secure E-Mail Client encrypts address book entries before storing them into the address book and decrypts them before displaying them on the display panel. Furthermore, the Client is connected to the Secure E-Mail Server in order to upload and download protected address books and address-book-symmetric-keys for recovery and portability purposes.

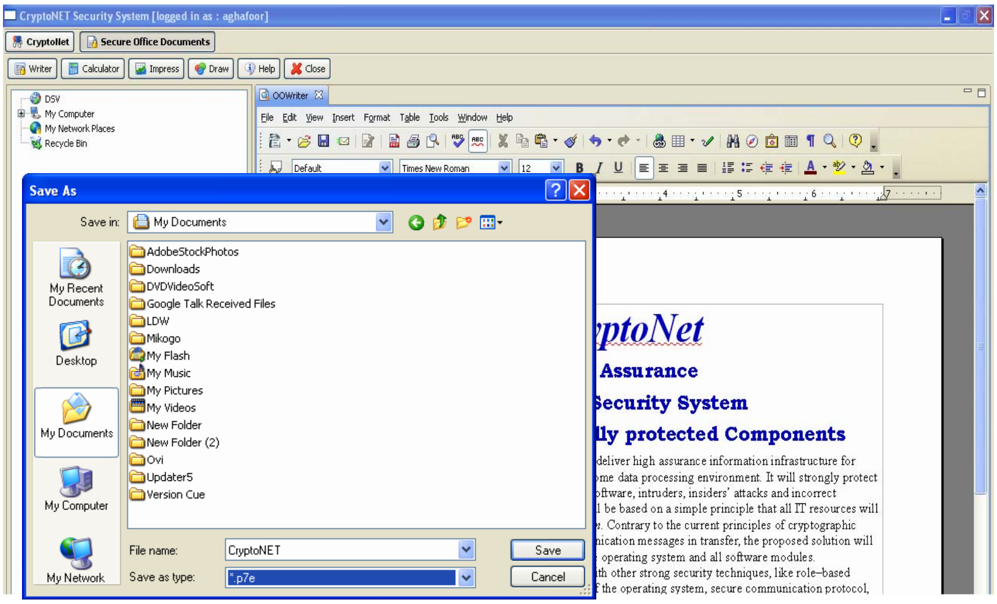
Secure E-Mail Client uses only Signed and/or Enveloped E-mail letters. This approach reduces threats of viruses, spam and malicious code. In addition, in order to be authorized to receive those mails, each Secure E-Mail Server applies authorization polices, specifying authorized "Sending To" and "Receiving From" Secure E-Mail Servers. The complete functionality of our Secure E-Mail Server is explained in Chapter 11.

10.3.3. Secure Web Browser

Secure Web Browser is a component of Integrated Secure Workstation. In order to provide extended security functions and features, it is logically located between Secure Web Server and standard Web browser. This component redirects Web browsers' HTTP requests to Secure Web Server, gets responses from Secure Web Server, and sends them to Web browser. It also creates secure session with Secure Web Server in order to protect web contents at the transport level. This component accepts only data in `PKCS7SignedAndEnvelopedData` format, which also protects the Workstation from viruses and malicious code. Furthermore, it uses local-resource-symmetric-key to protect history and cookies information in order to provide privacy and protection of Workstation from Web tracking. Detail functions of this component are described in Chapter 12.

10.3.4. Secure Documents Manager

Secure Documents Manager offers standard documents handling functions to end-users, like manipulation of word documents and spreadsheets, image editing and presentations. This functionality is based on the OpenOffice, which is extended with security features, as shown in Figure 10.3.



Save in: My Documents

File name: CryptoNET

Save as type: *.p7e

yptoNet

Assurance

Security System

lly protected Components

deliver high assurance information infrastructure for
ome data processing environment. It will strongly protect
oftware, intruders, insiders' attacks and incorrect
l be based on a simple principle that all IT resources will
. Contrary to the current principles of cryptographic
nication messages in transfer, the proposed solution will
operating system and all software modules.
ith other strong security techniques, like role–based
f the operating system, secure communication protocol,

Figure 10.3. Secure Documents System based on OpenOffice with Security Extensions. It saves documents in encrypted format with *.p7e extension.

Secure Documents Manager stores transparently protected files using `PKCS7SignedAndEnvelopedData` format. This module also uploads and downloads documents to and from the Secure Library Server, which is actually a repository of documents in a cloud computing environment. Furthermore, it provides options to securely distribute documents within the group of users.

Secure Documents Manager uses security protocols for distribution of documents explained in the Chapter 5. The format of distributed documents is also

PKCS7SignedAndEnvelopedData. In addition, this application also manages documents in group environments and structures documents into sections, where each section is accessible only to authorized users. Enforcement of authorization polices and key management is performed by the Secure Library Server using standard protocols. The details of Secure Documents System are described in Chapter 13.

10.4. Evaluation of the Integrated Secure Workstation

We qualitatively evaluated our Integrated Secure Workstation using threat model. We also analyzed the behavior of our system against several potential threats. The result of our evaluation is shown in Table 10.1. In User Registration process, our system uses SymmetricKey object in order to protect password before storing it into IDMS. This function is protected against Information Disclosure and Spoofing attacks. The Login function of our system is protected against Information Disclosure, Spoofing, Elevation of privileges and Repudiation attacks. Our login function either uses username and password or smart card for local authentication. In both cases it protects against the above attacks.

As shown in Table 10.1, Local Files Management functions protect files and IT resources against Tampering, Information Disclosure, Spoofing, Elevation of privileges and Repudiation, because we used PKCS7 object to encapsulate local files and IT resources in standard cryptographic format; *PKCS7SignedAndEnvelopedData*.

Our system handles transparently security credentials which are protected based on well-established security standards. Furthermore, our Integrated Secure Workstation generates certificate-requests for a current user after verifying his/her identity from the IDMS server. In addition, it protects security credentials using our generic security objects which are already in Chapter 8 proved that they are protected against Tampering, Information Disclosure, Spoofing, Elevation of privileges and Repudiation attacks. Similarly, our Workstation transparently handles smart card and smart card based cryptographic functions after authenticating users to a smart card using PIN.

We described in earlier sections that our Integrated Secure Workstation comprises several client components which interact with various corresponding Secure Application Servers. For communication we use our secure session protocol which protects messages from Tampering, Information Disclosure, Spoofing and Repudiation attacks, since each message is encapsulated in the *PKCS7SignedAndEnvelopedData* cryptographic format. AuditLog management is a key feature of our Integrated Secure Workstation. In our system entries of our AuditLog modules are protected using SymmetricKey object, so the system protects Log files form various attacks, as shown in Table 10.1. After evaluating and validating our Integrated Secure Workstation, we established that if a generic security object is tested and protected against various attacks, then the whole system developed using such objects is also secure, tested and verifiable.

Table 10.1. Threat Model for the Evaluation of Integrated Secure Workstation.

No.	**Functions and Features**	**Security Threats**				
		Tampering	***Information Disclosure***	***Spoofing***	***Elevation of privilege***	***Repudiation***
1	User Registration	-	×	×	-	-
2	Login Function	-	×	×	×	×
3	Local Files Management	×	×	×	×	×
4	Security Credentials	×	×	×	×	×
5	Smart Cards Handling	-	×	-	×	-
6	Communication	×	×	×	×	×
7	Audit Log Management	-	×	×	×	-

10.5. Summary

Our Integrated Secure Workstation (ISW) provides comprehensive set of security services for user Workstation environments and selected applications. The main principle was to cryptographically protect local IT resources, properties and messages. It transparently handles security functions and services. The design of ISW is based on the concept of generic security objects, which can be used by any application included in the Secure Workstation.

11. Secure E-mail System

In this Chapter we describe Secure E-mail System which uses security objects from the Security Provider in order to send and receive secure E-mail letters, to protect address books, to manage address-book-symmetric-key, to handle confirmation messages, and to protect inboxes from spam. Furthermore, we introduced Secure Mail Infrastructure Server which provides trust between domains using Federation Protocol in order to register E-Mail Servers, to exchange domain names and for validation of domain names.

11.1. Overview and Featues of The Secure E-mail System

Secure E-mail System supports standard E-mail functions and their extended security features. It comprises three core components. These are Secure E-Mail (SEM) Client, Secure E-mail (SEM) Server, and Secure Mail Infrastructure (SMI) servers. All these components use our security objects as engines which provide cryptographic and network level security services. In addition, these Secure E-Mail System components are also connected with our cloud security infrastructure that provide certification, authentication, authorization, and secure communication services.

The following are the key features of our system:

- Supports strong authentication protocol complaint with FIPS 196 standard and SAML-based Single-Sign-On;
- Transparent handling of security credentials and smart cards;
- Protection of mailboxes and E-mail letters (confidentiality, integrity, server’s and receiver’s authenticity);
- Secure and efficient handling of E-mail attachments;
- XACML-based authorization policies applied to sending and receiving E-mail letters in order to protect inbox from spam.
- Protection of address book entries and management of address-book-symmetric-keys;
- Confirmation of delivery of E-mail letters;
- Management of Secure Mail infrastructure performing registration of SEM Servers and exchange of domain names; and
- Validation of domain names to avoid spreading spam.

11.2. The Concept of Secure E-mail System

The architecture of our Secure E-mail System is shown in Figure 11.1. It is derived from our CryptoNET framework. Each component of the Secure E-mail System is based on our security objects and designed according to our research approach, described in Chapter 1. We adopted proxy–based architecture in order to provide extended security features and functions to E-mail clients. The core components of Secure E-mail System are: Secure E-Mail Client, Secure E-Mail Server, and Secure Mail Infrastructure Servers. Some of supporting components, shown in Figure 11.1, are already described in Chapter 9.

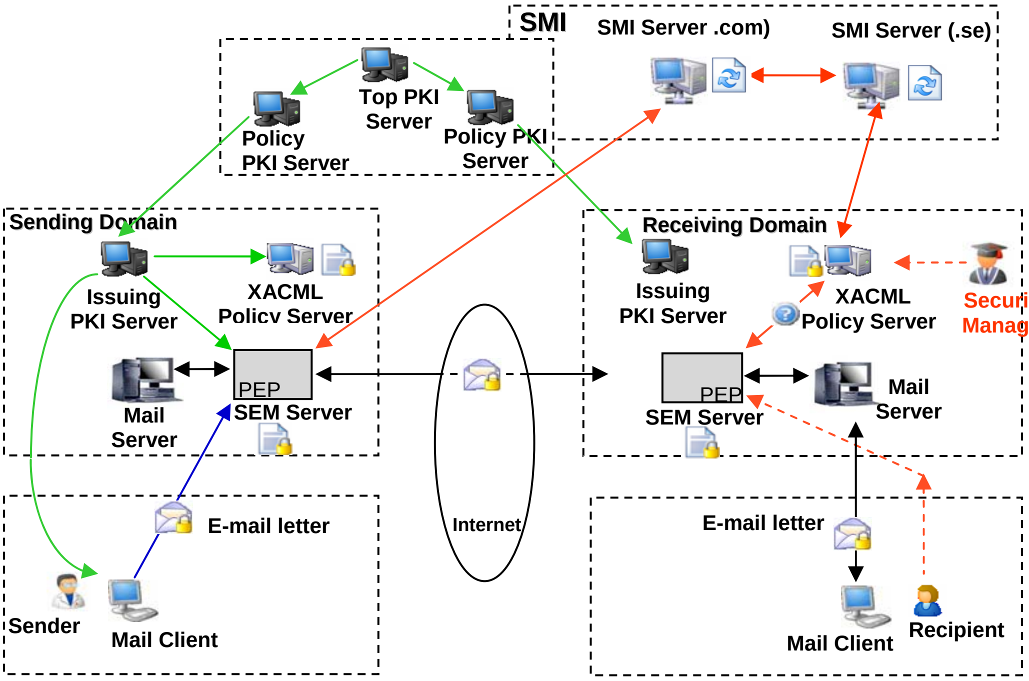


Figure 11.1. Architecture of Secure Mail Infrastructure and Interactions between Components

The first component of the system is Secure E-Mail Client, located at the bottom of the system, as shown in Figure 11.1. It cryptographically protects E-mail resources. It sends and receives digitally signed and/or encrypted E-mail letters encapsulated in S/MIME standard. In addition, it cryptographically encrypts address book entries; manages address-book-symmetric-keys; supports Single-Sign-On and secure sessions protocols with the SEM Server.

The second component of the system is Secure E-Mail Server. This server acts as a proxy between Secure E-Mail Client and standard E-Mail Server, as shown in Figure 11.1. Secure E-Mail Server is logically a separate component, but physically deployed on the same machine where standard E-Mail Server is running. In our system Security Manager manages this server. The Server transparently fetches two certificates (digital signature + non-repudiation, key-exchange) from the LCA Server. Secure E-Mail Server is responsible for sending and receiving E-mail letters to/from standard E-Mail Server using SMTP and POP3 protocols. This Server supports Single-Sign-On, secure communication with SEM Client, efficient handling of secure E-mail letters, management of clients’ address books, management of address-book-symmetric-keys, enforcement of E-mail sending and receiving authorization policies, registration with SMI Server and exchange of client certificates with other SEM Servers.

The third component of Secure E-mail System is Secure Mail Infrastructure (SMI) Server. SMI Server is logically located at the top of the infrastructure in order to support Internet-wide cooperation and trust between various E-mail domains. SMI Server acquires two certificates from the CA server. The key functions of SMI server are: registers SEM Servers, certifies and validates domain names of SEM Servers, and exchanges domain names between infrastructure level servers. We designed Secure Federation Protocol (see Section 11.3.7) which creates federation of SEM Servers and SMI Servers and performs federation management functions.

11.3. Operations of Secure E-mail System

The following are extended security functions of our Secure E-mail system.

11.3.1. Strong Authentication and Secure Communication

At the startup SEM Client checks the possession of certificates, SAML tickets, and other required security credentials. If certificates do not exist, then it notifies user to acquire certificates using Secure Station Manager, as described in Section 10.2.3. If certificates exist, then it checks SAML Ticket. If it does not exist, then it performs Strong Authentication with the SA Server to acquire SAML Ticket, as described in Section 5.2.2. After this, Secure E-Mail Client performs Single-Sign-On with SEM Server and establishes secure session in order to exchange secure messages.

11.3.2. Secure E-mail Letters and Attachments

Protection of E-mail letters is based on the S/MIME standard. E-mail sender fetches recipients’ certificate from the LCA Server in order to encapsulate E-mail letters in S/MIME message. For efficient and secure handling of attachments, SEM Client encapsulates attachments into signed and/or enveloped PKCS7 objects and uploads them to the SEM Server. SEM Server sends back an URL corresponding to each attachment. SEM client embeds URL(s) into the body of an E-mail letter before sending an E-mail letter.

At the recipient’s side, SEM Client fetches new E-mail letters from SEM Server. SEM Server fetches E-mail letters from a standard E-mail server and verifies domain name (see Section 10.3.6) and applies receiving authorization policies (see Section 10.3.5). After that it sends authorized E-mail letters to SEM Client and sends confirmation message to sender (see Section 11.3.4). SEM Client receives E-mail letters and also requests SEM Server to provide certificate of the sender for verifying signature of each E-mail letter. The SEM Server may interact with sender’s SEM server for exchanging certificates. In this case, it requests SMI server to resolve domain name to get the IP and Port number of the sender’s SEM Server. After acquiring sender’s certificate, SEM Client verifies signature and opens Secure E-mail letter. SEM Client stores protected E-mail letter in the inbox folder.

For downloading attachments SEM Client parses the body of the message and extracts URL(s). SEM Client downloads the file from the received URL and decrypts it before storing it in a local file system; as SEM Client already has sender’s certificate. Deletion of stored files on the SEM server depends on the accessibility of particular E-mail. If an attachment has already been downloaded, then it can be deleted from the SEM server. Otherwise, it is deleted when user deletes E-mail from inbox.

11.3.3. Secure Address Book

Protection of address book is one of the key functions of our SEM Client. For this purpose, Client generates an address-book-symmetric-key (also known as symmetric-key in this chapter) for encryption of address book entries. If a user wants to add a new contact, SEM Client uses the same symmetric-key to encrypt newly generated entry and saves it in the address book. Furthermore, Client has also an option to upload encrypted address book to the SEM Server to protect it against accidental loss. For management of symmetric-keys, SEM Client can upload symmetric-key to the SEM server. In this function, the symmetric-key is protected using public key extracted from key-exchange certificate of the SEM Client. Furthermore, recovery of the address book from accidental loss is performed by downloading of encrypted address book together with the relevant symmetric-key from the SEM server.

11.3.4. Confirmation Messages

Tracking user’s E-mails is also a feature of our SEM system. It is achieved by designing three types of confirmation messages: confirmation of delivery, confirmation of receipt, and confirmation of acceptance. Confirmation of delivery message is sent by the SEM server when it receives E-mail from the standard E-mail server. Confirmation of receipt message is sent by the SEM server when recipient downloads E-mail. Confirmation of acceptance message is sent by recipient’s SEM client when it opens the message.

11.3.5. E-mail Authorization Policies

In our system spam E-mails are prevented by applying XACML authorization polices. These policies are managed by the XACML Policy Server and enforced by the PEP server. XACML policy file contains entries about the authorized users and domains. Security Administrator of the SEM Server manages domain information, while individual users specify the list of authorized recipients.

PEP Server, proxy of the SEM server, enforces authorization policies for outgoing E-mails by consulting XACML Policy Server. PEP Server uses *SAMLPolicyAuthorizationRequest* and *SAMLPolicyAuthorization Response* messages to communicate with the XACML Policy Server in order to handle authorization services when sending E-mails. Similarly, authorization policies for incoming E-mails are handled by verifying “From” E-mails address and domain names from the XACML Policy Server.

In the described authorization solution, it is still possible for an attacker to send spam to a remote SEM server using “From” field with a valid E-mail address. To prevent system from such attacks, E-mail address of each user must be certified by the Security Administrator when he/she creates user account. Security Administrator encapsulates the E-mail address in a verifiable format: *PKCS#7SignedData*. The Sender of an E-mail inserts certified E-mail address in the header of E-mail, which is then verified by the receiving SEM Server, before enforcing authorization policies. In addition, recipients' SEM Server can validate the domain name of the sending SEM Server by interacting with the SMI Server using Secure Federation protocols.

11.3.6. Validation of Domain Names

SMI Server registers Secure E-Mail Server. It stores IP address of the SEM Server, its port number, and domain name into the SMI Domain Names List. The details of this registration process are described in Section 11.3.7. SMI Server certifies domain name and packages it into *PKCS7SignedData* format. When recipient Secure E-Mail Server receives E-mail, it consults SMI server to validate the domain name. Validation of the domain name is based on two functions: (1) existence of domain names in SMI domain names list, and (2) verification of a signature generated by SMI server during certification process. If both functions return positive results, then SMI server returns IP address and port number of the sender's SEM server. Otherwise it discards E-mail and sends a failure notification to the sender that the E-mail is not coming from valid domain.

11.3.7. Federation of SEM and SMI Servers

Certification of domain addresses in the SEM System is achieved by introducing a Secure Mail Infrastructure (SMI) Server. SMI Server is responsible to register SEM Server, certify domain names, and then establish a federation of SEM and SMI Servers.

Each SEM Server in our system must be registered with a SMI Server and SMI Servers must communicate with each other in order to exchange domain names and to perform verification of domain names. To register SEM Server with an SMI Server, we designed Unilateral Registration Protocol (URP). For creating SMI level federation and exchange of domain names, we designed Mutual Registration Protocol (MRP).

URP performs functions related to registration of SEM Servers with an SMI server and to certification of domain addresses, while MRP is used to exchange registration information between SMI Servers. Furthermore, we also structured the following messages for creation and management of the federation procedure at the infrastructure level. The format of each message is shown in Appendix C.

Registration Request: The purpose of this message is to apply for registration with a SMI server. This message contains originator's name, receiver's name, session ID, domain name of SEM server (URL), IP, port and General Security Function (GSF). URL, IP and port of SEM server are required by the SMI Server for registration, while GSF is needed in order to verify the integrity of message contents. GSF field is calculated using asymmetric cryptography, as shown in equation 8.1:

```
GSF =  E ( H( SessionID | URL | IP | Port ) , PR ) ………  (8.1)
     Where E = Asymmetric encryption
     H = Hash function
     PR = Private Key of GSF creator
```

Verification of the GSF field at the receiver's side is performed by using the following functions:

```
HS = D ( H ( SessionID | URL | IP | Port ) , PP)    ……… (8.2)
      H` = H ( SessionID | URL | IP | Port )
      Verification of GSF:  if        H` = H
      Where
      HS = Received hash from sender
      H` =Hash calculated by receiver
      PP = Public Key of sender
```

In the URP, GSF field is encrypted using public key from the certificate of the Security Administrator, while in the MRP GSF field is encrypted using public key of the certificate of the Administrator of the SMI Server.

Registration Reply: The purpose of this message is to receive acknowledgement from SMI server regarding the registration of the domain name. Registration Reply message contains originator's, receiver's, session ID, GSF, and certified URL. GSF field is calculated using cryptographic functions specified in Equation 8.1, but in this case session ID is only the input to this function, while the certified URL is in the format of *PKCS7SignedData*.

Registration Reply-Request: The Registration Reply Request message is a special type of a message which contains Reply and Request. The reply part is acknowledgement of the Registration Request, while Registration Request is the request for registration of the SMI server with the corresponding SMI server. The format of the reply message is similar to the Registration Reply, but without certified URL field. Registration Request message, which is nested in the reply message, contains information required by SMI server for registration of the requested SMI server in order to establish trusted federation.

Deregistration: Deregistration message is used when SMI Administrator decides to leave the federation. This message contains originator's name, receiver's name, session Id, URL, VURL and GSI. GSI is the output of the GSF function, which takes session Id and URL as input.

AddReferences: This message contains originator's name, receiver's name, session Id, and the list of references. These references are separated by ';'. The GSI field is the output of the GSF function. In this message GSF accepts session ID and list of references as input.

DelReference: This message is sent by an initiator SMI server to other SMI servers in the federation after deregistration of the particular SEM server. This step is needed in order to synchronize the domain name list and makes it consistent. This message contains originator, receiver, session Id and values of reference which is candidate for removal from domain name list.

Unilateral Registration Protocol

As mentioned above, the Unilateral Registration Protocol registers SEM server with an SMI server and SMI certifies the domain name of the SEM server. In this protocol SEM server initiates the protocol and in the first phase performs strong authentication, as shown in Figure 11.2 a. and as described in Section 5.2.2. After successful authentication, SEM server sends Registration Request to the SMI server which verifies the GSF field and compares session ID. Furthermore, the correctness of the session ID gives a notion that the message is received from an authentic source.

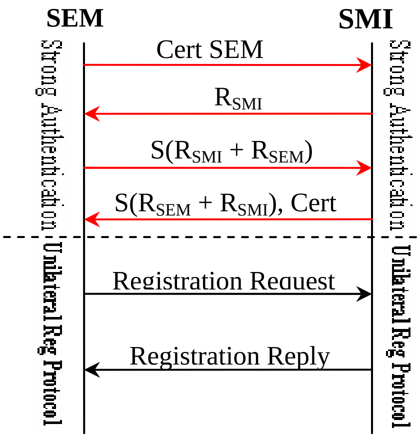


Figure 11.2 a. Unilateral Registration Protocol and communication between SEM and SMI Server

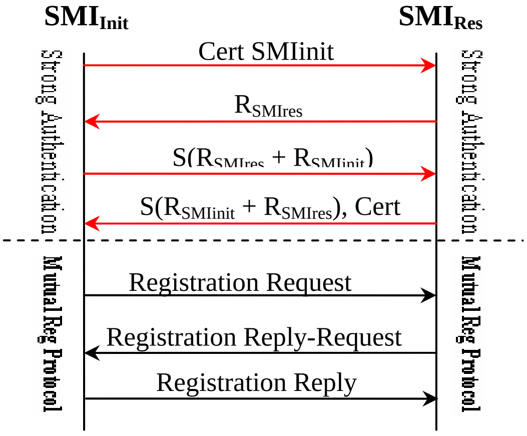


Figure 11.2 b. Mutual Registration Protocol and communication between SMI Initiator and SMI Responder

Upon successful verification of the GSF field, SMI stores URL, IP and port number in temporary storage accessible only by the Security Infrastructure Administrator. The Administrator views stored information and certifies domain name. SMI Server then stores the URL, IP and port number in the domain name database, cryptographically protected to ensure integrity of stored domain names.

SMI server sends Registration Reply message back to the SEM server which contains validated domain name. SEM Server processes Registration Reply and verifies the GSF field. Furthermore, SEM Server keeps certified URL in a local storage. SMI Server also sends a message to the SMI servers in the federation to notify them about newly registered SEM server.

Mutual Registration Protocol

Mutual Registration Protocol registers SMI Server with the another SMI Server and creates federation between infrastructure level servers. In this protocol any SMI Server can be the initiator of the protocol. It initiates strong authentication process, as shown in Figure 11.2 b. and as described in Section 5.2.2. The Initiator SMI Server sends Registration Request to the Responder SMI Server which verifies the GSF field and compares the session ID.

Upon successful verification of the GSF field, SMI stores URL, IP and port number in temporary storage, which is accessible only by the Security Infrastructure Administrator.

The Administrator views the stored information and declares the requested SMI Server is the trusted one. The Responder SMI Server then stores the URL, IP and port number in a domain names list and sends Registration Reply-Request message back to the Initiator SMI Server. The Initiator SMI Server processes Registration Reply and verifies the GSF field. After that, it processes Registration Request and stores URL, IP and port number in a temporary storage. The Administrator views stored information, marks the requested SMI Server as the trusted one, and sends Registration Reply message as acknowledgement.

Trust between two domains inherently develops trust between SEM Servers of both domains. SMI Server stores information cryptographically protected to ensure the integrity of URLs.

Management of The Federation

Management of the federation includes exchange of newly registered domain names with SMI servers in a federation and deregistration of domain names. When an SMI Server certifies a new domain, it sends a message to other SMI servers that exist in a federation to add new server into the domain name list as a reference. Similarly, if SMI Server Administrator finds that the particular SEM Server is compromised, he can deregister it by sending deregister reference message to other SMI servers. Corresponding SMI Servers will upgrade domain names list accordingly. Furthermore, manipulation messages are digitally signed in order to provide message source authentication and integrity.

11.4. Evaluation of the Secure E-mail System

We qualitatively evaluated our Secure E-mail System by applying Threat Model on its individual functions as shown in Table 11.1. The authentication in our system is performed using Strong Authentication object which protects authentication mechanism against Tampering, Spoofing and Repudiation attacks because it uses certificates and digital signature of random numbers.

Secure E-mail system uses Secure Sessions protocol for secure communication so each message between Secure E-mail Client and Secure E-mail Server is protected against tampering, information disclosure, and spoofing attacks. Since, messages are encapsulated in *PKCS7SignedAndEnvelopedData* cryptographic format so this protocol also resists man-in-the-middle and impersonation attacks. E-mail letters in our system are encapsulated in S/MIME format using S/MIME generic security object. So, E-mail contents are protected against Tampering, Information Disclosure, Spoofing, Elevation of Privileges, and Repudiation attacks. Similarly, our E-mail attachments are encapsulated in *PKCS7SignedAndEnvelopedData* cryptographic format which also protects our attachments against above mentioned attacks.

Our Secure E-mail Client protects address book entries using *SymmetricKey* object which loads address-book-symmetric-key, either from smart card or from key-file. This function protects our address book entries from Information Disclosure and Spoofing attacks. The confirmation of E-mail delivery is a key feature of our system. These messages are digitally signed, so they are protected against Tampering, Spoofing and

Repudiation, as shown in Table 11.1. Secure E-mail System interacts between domains using our Secure E-mail Infrastructure servers. Domain name of each Secure E-mail Server is certified by SMI server and each Sender Secure E-Mail Server adds it in the header of E-mail before sending to the recipient Secure E-Mail Server. This feature protects our domain names against Tampering, Spoofing and Repudiation threats.

After evaluating and validating our Secure E-mail System, we established that if generic security objects are completely protected, tested and verifiable then the system developed using those objects is also secure, tested, and verifiable.

Table 11.1. Threat Model for the Evaluation of Secure E-mail System.

No.	**Functions and Features**	**Security Threats**				
		Tampering	***Information Disclosure***	***Spoofing***	***Elevation of privilege***	***Repudiation***
1	Strong Authentication	×	-	×	-	×
2	Secure Communication	×	×	×	×	×
3	Protection of mail boxes and Email Contents	×	×	×	×	×
4	E-Mail handling	×	-	×	×	-
5	E-mail attachments	×	×	×	×	×
6	Protection of Address book	-	×	×	-	-
7	Address book Symmetric Key Management	×	×	×	×	×
8	E-mail Confirmation Status	×	-	×	-	×
9	Validation of Domain names	×	-	×	-	×

11.5. Summary

Our Secure E-mail System is a network application, designed and implemented based on our security objects, components and principles. Enabling components of the security system provide transparent handling of security credentials and interaction with FIPS 201 (PIV) smart cards. The system may be verified to provide solutions of all identified problems. In this system, users are authenticated using strong authentication and Single-

Sign-On protocols, while communication is secured using secure session protocol. E-mail letters are protected (in storage and in transit) using S/MIME standard, so the system is interoperable with existing E-mail systems. The XACML authorization policies, certified E-mail addresses, and domain names greatly eliminate spam, which eventually decrease the threats of spreading malicious contents. Attachments are handled very efficiently and securely, so our system does not unnecessarily increase network traffic. Address books entries are encrypted, what eliminates theft of addresses. In addition, management of address-book-symmetric-keys provides the feature to recover address books from accidentals losses. Three types of confirmation messages provide full tracking of E-mails by the sender. In a global environment, where multiple domains are deployed, federation protocol establishes trust between SEM and SMI servers in order to form a federated infrastructure. As a proof of the concept, we implemented and tested this system which provides protection of E-mails' resources, so it can be used for exchange of business documents and important information between authorized users.

12. Secure Web System

In this chapter we describe the design of our Secure Web System whose purpose is to protect workstations from various Web attacks. The design of the system is based on our basic principles and the concept of generic security objects. For this system we designed proxy architecture and we introduced two new components for security extensions of existing Web systems. These components are: Secure Web Browser and Secure Web Server. The system uses our enabling components that provide the same cryptographic and network security services to both components of the system. The key features of our Secure Web System are: protection of Web pages stored at a Web server, execution of Web contents using Secure Execution Environment, XACML based authorization policies, FIPS 201 (PIV) smart card-based authentication protocols, strong authentication, Single-Sign-On, and secure distribution of Web contents.

12.1. Overview and Features of The Secure Web System

Our Secure Web System is also based on the principle that all resources, messages, operations and contents are cryptographically protected using strong encryption techniques. Secure Web System uses enabling components for protection of Web contents, both in storage and in transit. The system supports standard Web protocols, extended security features, and network security protocols. We designed proxy architecture, compatible with existing Web infrastructures without much modification.

The following are security features of our Secure Web system:

- Protection of Web pages and Web resources stored at Web Servers based on the PKCS#7 encapsulation standard;
- Secure Execution Environment supporting processing of encrypted Web pages and resources;
- XACML-based authorization policies for access to resources and execution of Web applications at Web servers; and
- FIPS-196 based mutual strong authentication and single-sign-on protocols using FIPS-201 (Personal Identity Verification) compliant smart cards.

In addition, our system supports standard network security protocols, such as certification protocol, SAML protocol, and secure sessions protocol, as explained in Chapter 5.

12.2. Design of The Secure Web System

The design of Secure Web System is based on the concept of proxy architecture. We introduced two new components, shown in Figure 12.1: Secure Web Server (SWS) and

Secure Web Browser (SWB). SWS is located in front of a standard Web server and performs security functions: strong authentication of clients, processing of protected messages, creation of secure channels, enforcement of authorization policies, etc. The detailed operations of these functions are explained in sections below.

Secure Web Browser is located between a standard browser and a network. This component is responsible for redirection of HTTP requests and responses to/and from the SWS. Communications (HTTP requests and responses) between SWB and SWS are protected using secure sessions protocol described in Section 5.2.4.

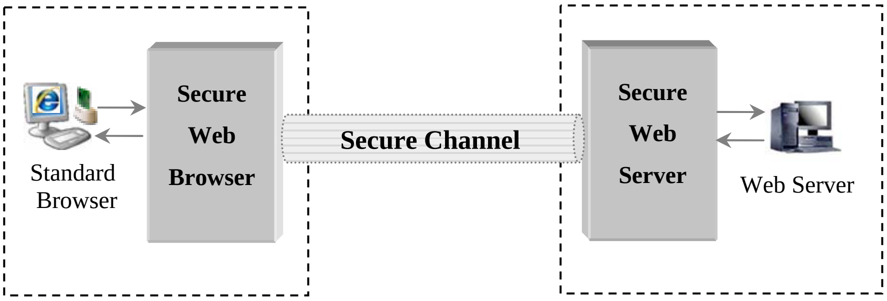


Figure 12.1. Secure Web Browser and Secure Web Server

12.3. Operations of The System

Both SWS and SWB acquire certificates from the LCA Server, as described in Chapter 9. Security Administrator of a Secure Web Server uses these certificates to protect Web pages and resources. In addition, Security Provider is integrated with this system in order to provide software-based or smart card-based cryptographic services.

12.3.1. Web Contents Protection

Security Administrator (SA) of a Secure Web Server is responsible for encryption of Web pages using strong encryption techniques. SA generates symmetric-key for encryption of Web pages and uses private key corresponding to the digital-signature certificate in order to sign encrypted Web pages. Our system encapsulates protected Web pages in the standard cryptographic format, PKCS#7, and structures them in our designed XML file, which describes general syntax of protected software modules, as explained in Section 7.2. After protecting Web pages, Security Administrator stores them at a standard Web server in the Protected Web Files Repository. SA also stores symmetric-key in a shared space. Symmetric-key is protected using public key corresponding to the SSL certificate of a Web server. In addition, SA defines the required roles and authorization policies at the XACML Policy Server in order to specify authorization for access to Web resources for authorized users.

We categorized protected Web pages in four groups based on the type of Web page and it’s processing at a Web server:

(1) Static Web Pages: These Web pages contain static code executed in a browser environment at the client side. Examples of these Web pages are: HTML pages, Java Scripts, CSS, etc.

(2) Active Web Pages: These Web pages contain HTML code and active code. These pages are processed by a Web server. Furthermore during deployment, Web server parses these Web pages and generates Web modules. Example of these Web pages is Java Server Pages (JSP).

(3) Web Modules: These Web pages contain active code which embeds HTML tags. These Web Modules are loaded using standard Execution Environment in order to generate HTTP responses. Example is Java Servlets.

(4) Core Web Modules: These modules are normally stored in a compiled form, loaded by the standard Java Class Loader. Examples of these core Web Modules are Java classes or executable Jar files.

Execution environment of standard Web servers, specifically designed for Web servers, cannot process and execute protected web pages. Therefore, we also designed an execution environment extended with security features and functions in order to process protected Web pages. Our secure Execution Environment is implemented in the form of Eclipse plug-in using Java technology and it works with Apache Tomcat Web server.

12.3.2. Secure Execution Environment for Web Contents

The components of our Web secure Execution Environment are shown in Figure 12.2.

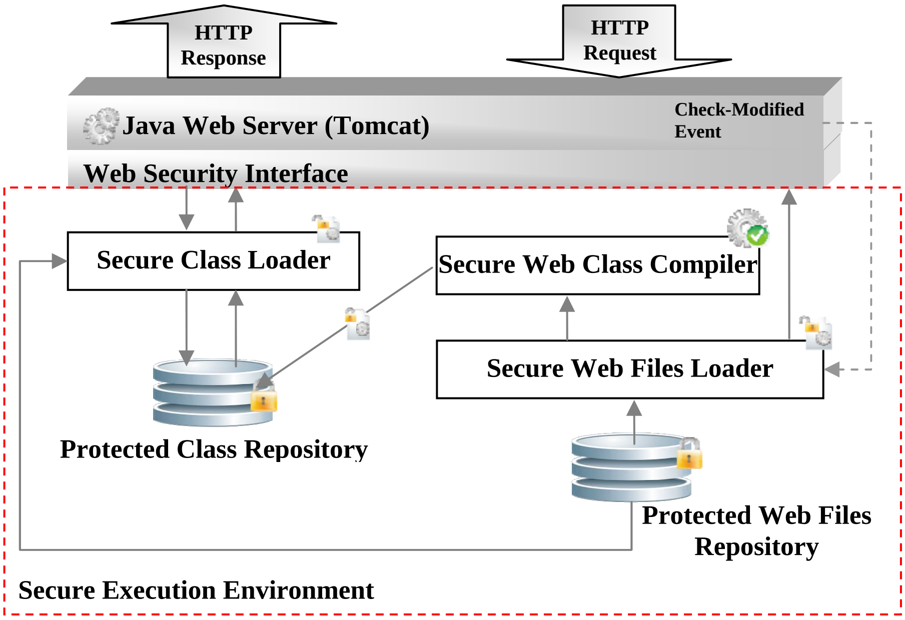


Figure 12.2. Components of secure Execution Environment and Interactions between them for processing of protected and encapsulated Web Pages.

When Security Administrator deploys protected web pages at a standard Web server, Web server triggers *Check-Modified Event* to load modified or newly loaded encrypted active Web pages. In response to this event, Secure Web Files Loader loads protected Active Web Pages structured in the XML format. This component verifies signatures of loaded Active Web Pages. Upon successful verification, it opens enveloped Active Web Page, as requested by the client, and handovers it to the Secure Web Class Compiler. This component transforms the received contents into Java source file, which is then compiled by the standard compiler to generate byte-code. This component also protects newly generated byte-code using its own credentials and stores it in the Protected Class Repository.

When an authorized user requests a Web page, Web server activates Web Security Interface (WSI), a component of the Secure Execution Environment (SEE) (see Section 12.3.2). WSI extracts physical path of the requested Web Page and checks the extension of the file in order to find the type of the Web Page. Based on the type of the requested Web Page, our system performs one of the following actions:

- If the requested file is a static Web page, then Web Security Interface sends the request directly to the Secure Web Files Loader. The Loader fetches static file from the Protected Web Files Repository and verifies its digital signature. Upon successful verification, it fetches encrypted symmetric-key and extracts it in order to decrypt the loaded Web page. *Secure Web Files Loader* passes clear Web page to the Web server through the Web Security Interface. Standard Web server processes the fetched Web file and generates HTTP response. After that, it sends the response back to the Web client.
- If the requested file is an active Web page, then *Web Security Interface* invokes *Secure Class Loader* in order to load protected classes. It reads header information and verifies digital signature of the loaded classes. Upon successful verification, it fetches encrypted symmetric-key and extracts it in order to decrypt the encrypted loaded classes. In addition, it loads, verifies, and decrypts all dependant classes and core business classes, if they are required. After performing all those security functions, *Secure Class Loader* passes loaded files to the standard Web server in order to generate HTTP response.
- If the requested file is a Web module, Web Security Interface invokes Secure Class Loader in order to load protected classes from the Secure Web Files Repository and processes Active Web Page.

12.3.3. Authorization and Distribution of Secure Web Contents

When a user wants to use Web site extended with our security features, he/she has to be registered as the user in the IDMS through a Web interface. Security Administrator assigns a role *User* to newly registered users, based on the predefined policies. These policies are flexible and can be changed according to the requirements of the Web site administrator. After successful registration, Web server sends signed ActiveX control back to the Web browser. This ActiveX control is extended with modules of our Security

Provider. ActiveX extends the functionality of the standard Internet Explorer (IE) in order to communicate with the FIPS 201 (PIV) smart card for various cryptographic functions and strong authentication. User uses ActiveX to fetch certificates from the LCA Server and stores them into a smart card, if it is installed. Otherwise, Web browser stores certificates in a certificate database.

In our system, a user provides PIN to open smart card and ActiveX component then automatically performs smart card-based strong authentication with the SA Server in order to acquire SAML Ticket. ActiveX stores SAML ticket in a smart card, as described in Section 5.2.2. SAML Ticket is then used for Single-Sign-On authentication and authorization services in the Secure Web System. For Single-Sign-On, Web browser digitally signs SAML Ticket using ActiveX control and sends it to the SWB which forwards it to the SWS for a Single-Sign-On, as described in Section 5.2.3. After this step, SWS establishes secure session with the SWB.

When user requests a Web page, he/she sends URL to the SWS through the SWB. PEP Server at the SWS filters the request and generates *SAMLAuthorizationRequest*, which contains SAML Ticket (subject), URL (object), and the requested action. PEP sends this request to the XACML Policy Server which evaluates it using XACML policy. The decision about request is then sent back to the PEP in the form of *SAMLAuthorizationResponse*. If response contains *Deny*, SWS generates an *Access denied* error message and sends it to the Web browser. Otherwise, SWS forwards the request to the standard Web server for further processing (see Section 12.3.2). Web server processes the request and sends HTTP Response to the Secure Web Server, which sends it back to a Web browser through the Secure Web Browser.

12.4. Evaluation of the Secure Web System

We qualitatively evaluated our Secure Web System using Threat Model. Comprehensive evaluation results of the Secure Web System are shown in Table 12.1. In this system, Web pages, stored at a Web server, are protected using PKCS7 object which encapsulates each Web page in *PKCS7SignedAndEnvelopedData*. This protection mechanism protects our Web pages against Tampering, Information Disclosure, Spoofing, Elevation of privilege and Repudiation threats. Similarly, in our system Web contents distribution process is also protected against above mentioned attacks, as shown in Table 12.1, because shared files are protected using shared-symmetric-key and encapsulated in *PKCS7SignedData*. In order to access shared files, we used XACML based authorization policies which are digitally signed, so our enforcement authorization policies process is protected against Tampering, Spoofing and Repudiation threats. Furthermore, as this function is specifically designed for authorization, this process also protects against unauthorized access to Web pages.

Authentication in this system is implemented using Strong Authentication object which protects authentication mechanism against Tampering, Spoofing and Repudiation attacks. Furthermore, we used Single-Sign-On with SAML Ticket in order to authenticate clients for multiple services. SAML Ticket is digitally signed, so our Single-Sign-On process is protected against Tampering, Spoofing, Elevation of privileges, and Repudiation attacks.

We evaluated this system and observed that it protects Web resources from several attacks. We also analyzed that the system is designed and implemented using our generic security objects what is a proof of our deductive approach that if generic security objects are secure, tested and verifiable, then the system which is developed using such objects is also secure, tested and verifiable.

Table 12.1. Threat Model for the Evaluation of Secure Web System.

No.	**Functions and Features**	**Security Threats**				
		Tampering	***Information Disclosure***	***Spoofing***	***Elevation of privilege***	***Repudiation***
1	Protection of Web Pages	×	×	×	×	×
2	Web Contents Distribution	×	×	×	×	×
3	Authorization Policy enforcement	×	**-**	×	×	×
4	Strong Authentication	×	**-**	×	**-**	×
5	Single-Sign-On	×	**-**	×	×	×

12.5. Summary

Our Secure Web System is based on extension of existing standard Web technologies, on various security standards, and security protocols. The system provides confidentiality and integrity of Web contents, either stored at a Web server or transmitted to authorized clients. We also designed secure execution environment, which extends the functionality of exiting Web servers in order to process protected Web pages. In our system the authorization is based on XACML policies, which are enforced by the PEP Server before processing user’s requests.

13. Secure Documents System

In this Chapter we describe our Secure Documents System which is an extension of the OpenOffice. The system provides protection of documents in local environments, internal documents security by structuring them into sections, secure distribution of documents in a group environment, section-level XACML-based authorization policies, key management, Single-Sign-On, and secure communications between components. The design of the system is based on our principles and it uses enabling components, Security Provider and Security Protocol, to provide extended security services.

13.1. Overview and Features of The Secure Documents System

Secure Documents System supports standard document processing functions extended with security features in order to protect the documents in local environments and enable their controlled sharing and distribution in a cloud computing environment. Two important components of the system are: Secure Documents Manager and Secure Library Server. Secure Documents Manager is an extension of the OpenOffice with security functions and some additional features.

Secure Documents Manager cryptographically protects local documents using the standard cryptographic format, *`PKCS7SignedAndEnvelopedData`*, against illegal access and malicious alternation. In addition to the protection of complete documents, we provide a feature to structure a document into several sections, which are stored in the XML File Format [97].

In our system a document can be shared between group members. The author of a document partitions a document into sections and assigns a sensitivity level to each section. The author also creates groups and assigns a role and an access level to the group members. Therefore, each group member is authorized to access only authorized section(s). Authorization policies are based on the XACML standard, while symmetric-key cryptography and XML signature standard is used to protect each section of a document. In addition, we use GSAKMP protocol for management and for distribution of section-symmetric-keys to the authorized group members.

For cryptographic functions and network level security services, we use our enabling components as cryptographic engines. These components provide FIPS 201 (PIV) smart card-based cryptographic services, if smart card is installed; otherwise, the system uses software-based cryptographic services. The enabling components provide the same set of cryptographic services and network security services to all the components of the system.

All the components of our Secure Documents System are integrated with various cloud security infrastructure servers which provide certification, authentication, and authorization services as described in Chapter 9. The following are the key features of our system:

- Extension of the OpenOffice with standard security functions to protect documents using *PKCS7SignedAndEnvelopedData* format;
- Protection and controlled distribution of documents in a group environment;
- Structuring of documents in sections and each section is accessible to authorized users by enforcing section-level XACML-based authorization policies;
- Symmetric-key encryption and digital signature of section based on the XML security standards;
- Key Management using GSAKMP protocol; and
- Transparent handling of the security credentials and integration of the FIPS 201 (PIV) smart cards using enabling components.

13.2. The Concept of Secure Documents System

The design of our Secure Documents System is based on our main CryptoNET framework and our research methodology, as described in Chapter 9 and Chapter 1 respectively. Core components of the Secure Documents System are: Secure Documents Manager, Secure Library Server, and Key Distribution Server. Some of the supporting components, like LCA Server, XACML Policy Server, SA Server, and IDMS are also part of the system those are already described in Chapter 9.

Secure Documents System uses enabling components of our security system to provide the same set of security services to all the components of the system. In addition, the design of the system is also based on our generic approach, so each component is implemented in the form of a generic security object, which cryptographically protects and distributes documents in a collaborative environment.

The first component of the system is Secure Documents Manager, which is part of our Integrated Secure Workstation. It is an extension of the OpenOffice with security functions. OpenOffice provides standard documents handling functions to the end-users, like manipulation of documents, spreadsheets, image editing and presentations. This component provides security features, like protection of documents in the *PKCS#7SignedAndEnvelopedData* format, structuring of documents in several sections and their storage in XML File Format, sharing of documents and sections in a group environment, SAML based Single-Sign-On, and secure communication. In order to share documents in a group environment, Secure Documents Manager is connected to the Secure Library Server and also to the security servers which are components of our cloud security infrastructure.

The second important component of the system is Secure Library Server. It is administrated by the Security Administrator. It transparently acquires certificates from the LCA Server, the same as our SEM Server, described in Chapter 10. Secure Library Server provides management of documents repository, enforcement of XACML authorization policies, management and distribution of section-symmetric-keys using GSAKMP, Single-Sign-On, and secure sessions protocols. The description of each operation is given in the following sections.

13.3. Operations of The System

Each component of the system is linked to our cloud security infrastructure. Secure Documents Manager checks the certificates when a user activates it. If the required certificates do not exist in a certificate database, then it suggests to a user to acquire certificates from the LCA Server using Secure Station Manager, described in Section 10.2.3. Similarly, when Security Administrator activates Secure Library Server, it also checks for certificates in the local certificate database. If they do not exist, then it automatically acquires certificates from the LCA Server.

All the components of the system use enabling security components as cryptographic engine of the Security Provider. These components provide smart card-based or software-based cryptographic functions to the components of the system.

13.3.1. Protection of Documents in Workstations

Secure Documents Manager is part of our Integrated Secure Workstation. It integrates security components with OpenOffice in order to provide to the end-users standard document processing environment with extended security features. When a user presses *Save* or *Save As* action, the Documents Manager activates the component of the Security Provider which encrypts the document using local-resource-symmetric-key and digitally signs it using private key corresponding to the digital signature certificate. It then encapsulates protected document in the standard cryptographic format: PKCS#7. Secure Documents Manager stores documents in the XML File Format with *.p7m* extension in a local storage.

When a user executes *Open* document action, the system fetches the protected document and verifies its digital signature. Upon successful verification, it fetches local-resource-symmetric-key from a smart card, if it is installed, and then decrypts the document. After that, it passes the document to the OpenOffice in order to open it in the standard environment for viewing and/or editing purposes.

13.3.2. Distribution of Documents

Secure Documents Manager provides features to share documents in a group environment. In addition, it can structure documents into sections, which may contain one or more paragraphs. When a user starts *Share Document* process, a user creates a group and then defines XACML authorization policies. The system also supports key management functions in order to distribute section-symmetric-keys to authorized group members.

In addition to that, our system transparently fetches SAML ticket from a smart card and then performs Single-Sign-On with Secure Library Server, as described in Section 5.2.3. It also establishes secure session with the Secure Library Server in order to exchange secure information.

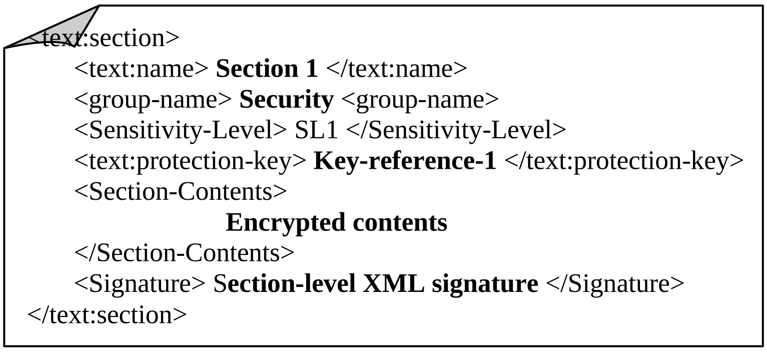
Groups and Group Level Management

The author of the document, also known as Group Controller (GC), is responsible to create a group, roles and sensitivity levels. GC creates various roles and assigns sensitivity levels to each role. After that, GC selects various registered users from the IDMS and then assigns a role to each user. In this system, each role has one-to-many relationship with the sensitivity level, so a user which have specific role may be authorized to access multiple sections of the document. In addition, the role is a logical link between sensitivity level and users, so a role may also be assigned to one or more users, because it provides flexible mechanism to assign the same level of access control to multiple users.

Upon creation, GC sends group information to the Secure Library Server. It forwards group-creation-request to the XACML Policy Server. XACML Policy Server dynamically creates a group in a shared space.

Creation of Authorization Policies for Multilevel Documents

An author of a document creates it using OpenOffice. A document may be further divided into various sections. Section starts and ends at paragraph boundaries and can contain any number of paragraphs. Each section is tagged with the predefined XML headers, as mentioned in [97]. Furthermore, we also defined additional XML tags to implement security extensions. An author of a document identifies sections and then adds additional tags in each section which identify group and sensitivity-level of the section, as shown in Figure 13.1.



```
<text:section>
      <text:name> Section 1 </text:name>
      <group-name> Security <group-name>
      <Sensitivity-Level> SL1 </Sensitivity-Level>
      <text:protection-key> Key-reference-1 </text:protection-key>
      <Section-Contents>
                    Encrypted contents
      </Section-Contents>
      <Signature> Section-level XML signature </Signature>
</text:section>
```

Figure 13.1. An Example of a Section in The XML Structure

The relationship between sections, sensitivity-levels, roles and users is shown in Figure 13.2.

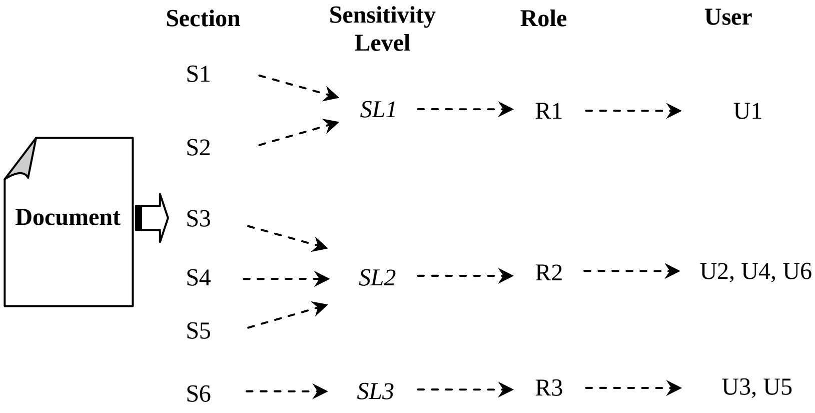


Figure 13.2. Relationship between Sections, Sensitivity Levels, Roles, and Users

Secure Documents Manager sends policy token to the Secure Library Server with all necessary information, like document's name along with section-names, sensitivity-levels and group identification. Secure Library Server forwards this policy token to the XACML Policy Server which dynamically creates XACML policies in a policy-file. After successful creation of authorization policies, XACML Policy Server notifies Secure Library Server which notifies further Secure Documents Manager that policy has been successfully created.

Protection of Shared Documents

Generic Key Distribution component is located at the Secure Library Server. It is compliant with the GSAKMP standard [15]. It is responsible for creation and distribution of section-symmetric-keys. In our system the process of creation of the section-symmetric-keys is started when Secure Library Server receives *XACML policy creation* notification from XACML Policy Server. Secure Library Server, in cooperation with the Generic Key Distribution component, creates section-symmetric-key for each section of the document. This key distribution technique has the following advantages:

- Section-symmetric-keys are not dependent on each other. So, any group member may insert a new section into a document without changing section-symmetric-keys of other sections.
- The author of a document may change the role of a group member or remove a user from a group, so in this case a rekeying action is required. In such situation, the system only needs to re-encrypt relevant section with the new section-symmetric-key.
- The author of a document may delete a specific section from a document. In this case our system deletes only designated section from a document and the corresponding section-symmetric-key.

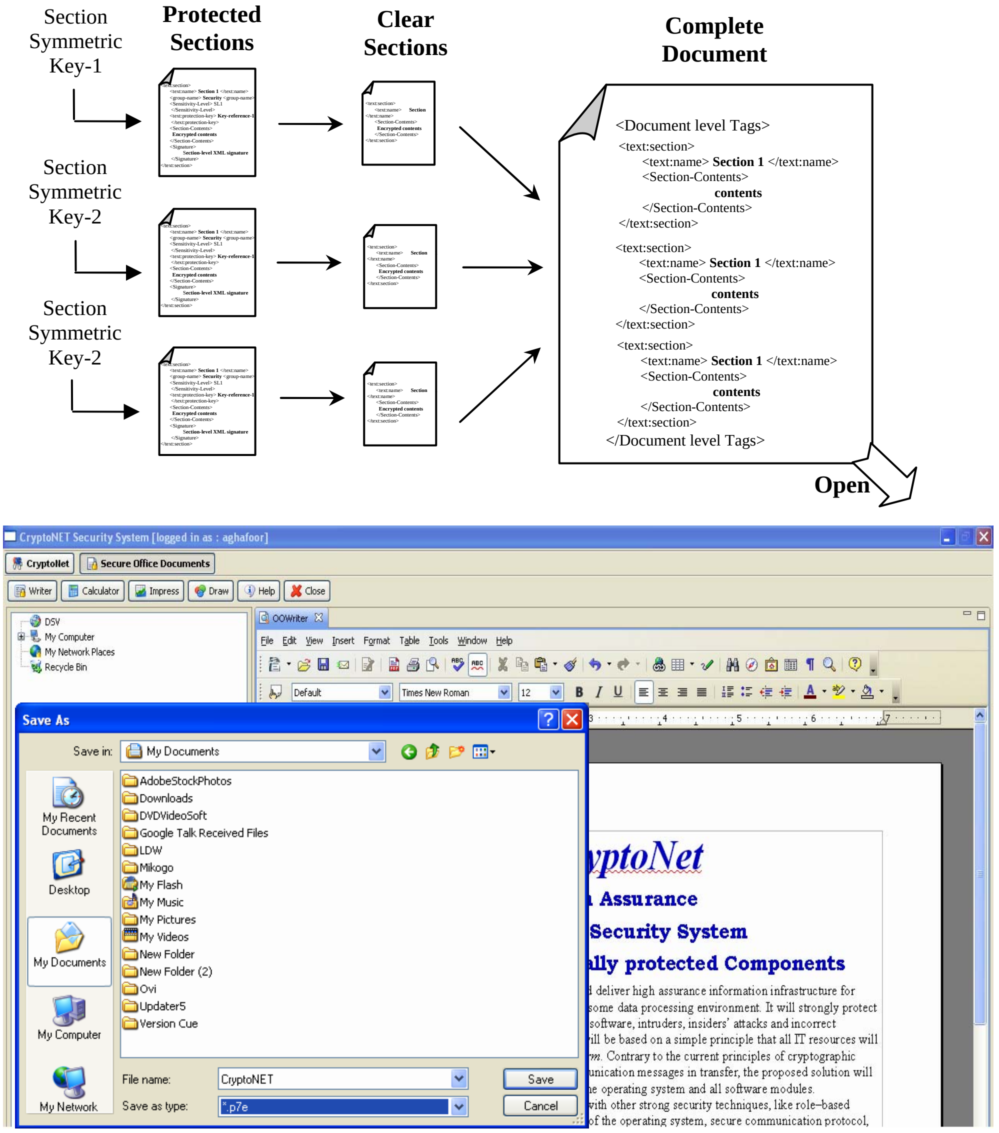
Secure Library Server sends section-symmetric-keys along with key-references to the Secure Documents Manager. Secure Documents Manager transparently encrypts a section with relevant section-symmetric-key and then it inserts the key-reference at the *text:protection-key* XML tag into the section, as shown in Figure 13.1.

In addition, it digitally signs XML Section by following the XML Signature standard. In this process, Secure Documents Manager digitally signs the complete section and inserts signature in the *Signature* XML tag. After protecting all sections, it creates the complete document and uploads it to the Secure Library Server.

13.3.3. Enforcement of XACML Authorization Policies

PEP Server at the Secure Library Server enforces authorization policies. When a group member is interested to access the authorized document from the Secure Library Server, it sends document name and SAML Ticket to the Secure Library Server. At the Secure Library Server, the PEP processes the request and generates a composite `SAMLAuthorizationRequest` object, which contains SAML Ticket (subject), name of sections (resources) and action (Read). PEP sends `SAMLAuthorizationRequest` to the XACML Policy Server which consults the

XACML policy file in order to evaluate the request against XACML policies. It generates *SAMLAuthorizationResponse* which contains either *Permit* or *Deny*. *SAMLAuthorizationResponse* is then sent back to the PEP Server which enforces the decisions. If *SAMLAuthorizationResponse* contains *Permit*, Secure Library Server fetches the document from shared repository and sends it back to the user along with key-references and section-symmetric-keys. Secure Documents Manager receives the document, key-references, and section-symmetric-keys. Secure Documents Manager verifies the signature of each section and decrypts each section using relevant session-symmetric-key. After that it removes additional XML tags and then combines all the sections into a complete document, as shown in Figure 13.3. Secure Documents Manager opens this document in an OpenOffice environment, which is a client component of our Integrated Secure Workstation.



File name: CryptoNET

Save as type: *.p7e

yptoNet
ı Assurance
Security System
ally protected Components
d deliver high assurance information infrastructure for
some data processing environment. It will strongly protect
software, intruders, insiders' attacks and incorrect
vill be based on a simple principle that all IT resources will
m. Contrary to the current principles of cryptographic
unication messages in transfer, the proposed solution will
he operating system and all software modules.
with other strong security techniques, like role–based
of the operating system, secure communication protocol,

Figure 13.3. Verification of Signature, decryption of Sections using section-symmetric-keys and opening of Documents in OpenOffice Environment

13.4. Evaluation of The Secure Documents System

In this section we evaluated and analyzed the behavior of our Secure Documents System in order to measure its resistance against several potential threats. As described in the above sections, our system encapsulates documents in the *PKCS7SignedAndEnvelopedData* cryptographic format using *PKCS7* objects, what protects documents against Tampering, Information Disclosure, Spoofing, Elevation of privileges and Repudiation threats. In our system, each document is structured into several sections. Each section is encrypted using shared group key and digitally signed using XML Signature standard. Furthermore, documents are shared between group members in a secure way, what protects these functions (Function 2 and Function 3 as shown in Table 13.1) from above mentioned attacks.

For enforcement of authorization policies, we use well-established SAML standard which protects our authorization process against Tampering, Spoofing and Repudiation attacks. Furthermore, as this function is used specifically for authorization purposes, it also protects against elevation of privileges in order to prevent unauthorized access to shared documents. In order to manage group keys between group members, we use GSAKMP protocol which protects key distribution mechanism against Tampering, Spoofing, Elevation of privileges and Repudiation attacks.

The authentication in this system is implemented using Strong Authentication object, which protects authentication mechanism against Tampering, Spoofing and Repudiation attacks. Furthermore, in this system we used Single-Sign-On protocol with SAML Ticket in order to authenticate clients for multiple services. SAML Ticket is digitally signed, so our Single-Sign-On process is protected against Tampering, Spoofing, Elevation of privileges, and Repudiation attacks.

As described in the earlier sections, communication between both components is protected using secure session protocol, which protects messages against Tampering, Information Disclosure, Spoofing and Repudiation, since each message is encapsulated in the *PKCS7SignedAndEnvelopedData* cryptographic format.

After evaluating Secure Documents System, we established that if each generic security object is secure, tested and verifiable then the complete system developed using such objects is completely protected and verifiable, what is a proof of our deductive verification approach.

Table 13.1. Threat Model for Evaluation of The Secure Documents System.

No.	**Functions and Features**	**Security Threats**				
		Tampering	***Information Disclosure***	***Spoofing***	***Elevation of privilege***	***Repudiation***
1	Protection of Local Documents	×	×	×	×	×
2	Distribution of Documents	×	×	×	×	×

3	Protection of Shared Documents	×	×	×	-	×
4	Authorization Policy enforcement	×	-	×	×	×
5	Group Key Management	×	×	×	×	×
6	Strong Authentication	×	-	×	-	×
7	Single-Sign-On	×	-	×	×	×
8	Communication	×	×	×	-	×

13.5. Summary

Our Secure Documents System is an extension of the OpenOffice with standard security functions in order to protect documents in the *PKCS7SignedAndEnveloped* format. The system uses our enabling components to provide the same set of cryptographic and network security services to all the components. The system protects and distributes documents in a group environment. In addition, our system provides feature to structure a document into sections, which can be shared by group members. Section level access control policies are enforced using XACML-based authorization policies. The system also manages section-symmetric-keys and provides all required security credentials only to authorized group members.

Part III: Overview of Significant Contributions and Future Research

14. Overview of Significant Contributions

The research presented in this thesis was focused on security in cloud computing environments and applications. Currently two the most popular two security approaches are Isolation and Software Security. In the first approach, various external software or tools are used to protect installed information systems. Such tools and products are Firewalls, Intrusion Detection Systems, Port Scanners, Packets filtering, Anti-viruses, etc. These protection tools protect resources after their installation. The second approach is Software Security. This approach includes methodologies for secure software design, development of secure libraries, rules for secure software development process, and formal and strict testing procedures. The goal of this approach is to create secure applications, even before their operational deployment. Current surveys, reports, news and experience clearly show that both approaches failed to provide an adequate level of security, where users would be guaranteed to deploy and use secure, reliable and trusted network applications.

Therefore, in the current situation, new approaches and new methodologies towards creating strongly protected and guaranteed secure network applications and cloud computing environments are required. In this thesis we proposed a new and innovative methodology for design of inherently security components, protocols, applications and large security systems. Our methodology is based on the concepts that if a system is internally secure and designed using secure objects, then it provides effective protection against viruses and external attacks. In order to achieve our objective, we designed a complete set of strongly and verifiably secure generic objects. ***Completely*** means that our generic security objects provide all five standard security services (integrity, confidentiality, authentication, authorization and availability), all major crypto algorithms, and all major security protocols. ***Strongly*** means that the generic security objects must provide provably correct functions and they are not vulnerable to threats and attacks. ***Generic*** means that each security object supports multiple alternatives and options. It provides protection of its data, functions, usage, combinations and objects instantiation. Furthermore, it dynamically integrates new algorithms and crypto objects without re-designing and re-development of software.

Our generic security objects are basic building blocks of our security system. We combined our generic security objects in order to design the complete and verifiably secure generic components of large-scale security environments. The result comprises several enabling components, such as Security Provider, Security Protocols, Generic Security Server, and Secure Execution Environment.

Our first enabling component is Security Provider. It is modeled and designed as collection of generic security objects in order to provide comprehensive set of cryptographic services, mechanisms, encapsulation techniques, and security protocols to all other components of our security system. The model of the Security Provider is structured in four layers and each layer comprises several generic security objects. Based on our model and design, we implemented an instance of the Provider comprising various objects: symmetric key cryptography, asymmetric key cryptography, hashing, encapsulation, certificates management, creation and verification of signatures, and various network security protocols.

We also designed Security Protocols for various distributed components, also a part of our enabling components. These protocols are designed on well-established security technologies and standards, which are interoperable with other components of the system in a cloud computing environment. They can be integrated with any application for secure communication, authorization, key distribution, Single-Sign-On, and strong authentication. These protocols are using our Security Provider in order to perform cryptographic functions and communications with smart cards. In addition, these protocols are generic, what makes them easy to use by developers for building secure distributed applications.

By using our Security Provider and Security Protocols, we further designed Generic Security Server, which provides complete set of standard security functions along with a number of extended security functions and features. It provides a template for developers in order to develop customized Secure Application Servers. We already implemented several initialization and management functions and several administrative actions. We also included APIs and libraries for cryptographic functions and security protocols, in order to provide the same set of security services in all instances of Secure Application Servers.

We packaged our enabling components in the form of the Security SDKs, protected using strong encryption techniques. Software protection solution comprises secure Software Distribution Server and Secure Web Server used to generate protected software modules and distribute them to authorized users. Our solution encapsulates these modules in the form of specially designed XML file which represents general syntax of protected software modules. We also extended current standard execution environments with special security features and functions. Our extended secure execution environment supports standard security services and network security protocols. Our solution protects software modules against reverse engineering, illegal tempering, program-based attacks, BORE (Break-Once-Run-Everywhere) attack and unauthorized use of software.

Furthermore, for our individual security objects and larger security systems, in order to prove their structural and functional correctness, we applied ***deductive scheme*** for verification and validation of security systems. We used the following principle: "*if individual objects are verified and proven to be secure, if their instantiation, combination and operations are secure, and if protocols between them are secure, then the complete system, created from such objects, is also verifiably secure*". Data and attributes of each object are protected and secure, and they can only be accessed by authenticated and authorized users in a secure way. This means that structural security properties of objects upon their installation can be verified. In addition, each object is maintained and manipulated in our secure environment, so each object is protected and secure in all its states, even after its closing state, because the original objects are encrypted and their data and state stored in a database or files are also protected.

We formally evaluated our approach and methodology of designing generic security objects using Threat Model. We analyzed our generic security objects individually and identified various potential threats for their data, attributes, actions and various states.

We also evaluated behavior of each object against potential threats and established that our software modules are not vulnerable to malware and viruses. Data of generic security objects are not vulnerable to illegal reading and theft, all messages exchanged in a networking environment are strongly protected, and all other resources of generic security objects are also strongly protected.

We have also solved some additional important aspects required for the full scope of security services for network applications and cloud environments: manipulation and management of cryptographic keys, and even secure and controlled collaboration of our encrypted applications in a cloud computing environments. During our research we have created the set of development tools and also a development methodology which can be used to create cryptographically protected applications. The same resources and tools are also used as a run–time supporting environment for execution of our secure applications. Such total cryptographic protection system for design, development and run–time of secure network applications we call CryptoNET system.

As a proof of our concept and methodology, we designed and implemented several applications for our CryptoNET environment. They are completely secure, protect their resources, designed based on standards, and implemented using generic security objects. We also applied deductive scheme in order to prove that each secure application is verifiably secure and tested, since they are designed using individually verifiable and secure objects and components. Following are our most popular secure applications:

- Integrated Secure Workstation is a client application which represents a collaborative environment and protects local IT resources, messages and operations across multiple applications. It comprises several components, i.e. four most commonly used PC applications: Secure Station Manager (equivalent to Windows Explorer), Secure E-Mail Client, Secure Documents Manager, and Secure Web Browser. These four components for their security extensions use functions and credentials of the enabling components Security Provider and Security Protocols. With this approach, we provide standard security services (authentication, confidentiality, integrity and access control) and also additional, extended security services, such as transparent handling of certificates, use of smart cards, strong authentication protocol, SAML based Single-Sign-On, secure sessions, and other security functions, to all PC applications with the same set of security modules and parameters.

- We also designed a Secure E-mail System using our proposed methodology. The system provides standard E-mail security services – signing and encryption of E-mail letters and, in addition, a number of extended and innovative security features. These new features are: transparent handling of certificates, strong authentication between Secure E-Mail Client and Secure E-Mail Server, archiving and recovery of encrypted address books, simple and secure handling of cryptographic keys, security sessions management, tracking of E-mail letters using confirmation messages, elimination of spam messages, prevention of fraudulent and infected attachments, and usage of smart cards. The system is based on proxy architecture, what makes it compatible with existing E-mail infrastructure. We also used XACML–based authorization policies at the sending and receiving Secure E-Mail Servers (SEM) to provide complete protection against spam. In our system, these policies are enforced by PEP, a component of the SEM server. In order to interconnect Secure E-mail systems in individual domains, we introduced new infrastructure level servers in order to develop trust between domains, exchange SEM registration information, and certify and verify domain names.

- We also solved some very critical security issues of Web systems using our enabling components. Features of our Web system are: protection of Web contents stored at Web Servers, execution of protected web pages, and their distribution to authorized users. We introduced additional components and added extended security features to standard Web system and protocols in order to provide confidentiality and integrity of Web contents. We also designed and implemented an extended secure execution environment for Java Web Server, which is capable to process and execute different types of encrypted and digitally signed Web pages encapsulated in *`PKCS7SignedAndEnvelopedData`* format. This system follows component-based architecture, what makes it compatible with the exiting Web infrastructure.

- We also deigned and implemented Secure Documents System in order to provide a proof of the concept and our methodology. This system comprises a set of security functions, features and components used as security extensions of the OpenOffice. The extended security features of this application are: protection of documents in local environments, distribution of secure documents to group members, group key management, enforcement of XACML policies for access control, smart card-based cryptographic functions, and transparent handling of security credentials. The design of the system is based on generic security objects and plug-in architecture, what makes it easy to extend and integrate with existing document systems. In addition, Secure Document System is linked to the cloud security infrastructure in order to provide security services in global environments by using certificates and standard, well established, security technologies and protocols.

We evaluated our secure applications individually and found that each application protects its data, resources, messages, usage and security credentials against potential attacks. We proved that our system is ***guaranteed*** to be secure and it is ***verifiable*** to provide the complete set of security services. We also proved that each application always operates correctly, what justifies our claim that it ***can not be compromised*** without user neglect and/or consent. We also established that if objects are individually secure, tested and verifiable, then the security system designed using such objects is also secure, tested and verifiable. Furthermore, we presented through implementation that the deductive scheme is an effective methodology in order to verify complex security systems. To the best of our knowledge, at the moment of this dissertation, we do not know any equivalent methodology, design concept, available security components and secure applications as achieved in this research.

14.1. Future Research

In our research, we already designed and implemented several generic security objects and deployed Security Infrastructure Servers as a service in private cloud but still there are number of security challenges and issues has to be investigated for the deployment of security and secure applications as a service in global cloud computing environment. Furthermore, future research can be carried out in order to investigate the solution of following issues:

- How personal information can be protected in cloud computing environments using our methodology without compromising security services?
- How can mathematically measure and proof the effectiveness of deductive verification in cloud computing environments?
- What is methodology to automate security services using 4th Generation Development Tools?
- What is the security metric to measure security in hybrid environment (both client-server and cloud computing environments)?

References

[1]. http://www.openssl.org/docs/OpenSSL, http://www.openssl.org/docs/, [visited: January 2009].

[2]. RSA Security, Inc. “BSAFE: A Cryptographic Toolkit”, Library Reference Manual Version 4.0 http://www.rsa.com/products/bsafe/documentation /cryptoc_411_ reference.pdf.

[3]. SUN Corporation, “Java Cryptographic Extensions (JCE)”, www.sun.com, [visited: February 2009].

[4]. Microsoft Corporation, “Cryptographic Services Provider (CSP)”, www.microsoft.com, [visited: February 2009].

[5]. FIPS PUB 201-1, “Personal Identity Verification (PIV) of Federal Employees and Contractors”, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, March 2006

[6]. Adams, S. Farrell, T. Kause, T. Mononen, “RFC 4210, Internet X.509 Public Key Infrastructure Certificate Management Protocol”, September 2005.

[7]. J. Linn, “Generic Security Service Application Program Interface”, RFC-2743, RSA Laboratories, January 2000.

[8]. K. Brown, “Explore the security support provider interface using the SSPI workbench Utility”, MSDN Magazine, Aug. Available at http://msdn.microsoft.com/msdnmag/issues/0800/ Security/Security0800.asp, 2000.

[9]. Denis Piliptchouk, “Java vs. .NET Security, Part 2 Cryptography and Communication”http://www.Onjava.com/pub/a/onjava/2003/12/10/javavsdotnet .html?page=1, [visited: October 2008].

[10]. Microsoft Inc.,“Microsoft CryptoAPI and Cryptographic Service Providers”, http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/distrib /dscj_mcs _xxgl.mspx?m fr =true build date on11/19/2009.

[11]. Scott Oaks, “Java Security (Security Providers)”, Chapter 8, Java Security”, First edition, ISBN 1-56592-403-7E, publisher O'Reilly Media and published in May 1998.

[12]. Y. Zhang, A. Timkovich, and J. Peck, “IBM Security Providers: An Overview”, Oct. 2004.

[13]. Institute for Applied Information Processing and Communication (IAIK), http://jce.iaik.tugraz.at/sic /products, [visited: December 2008].

[14]. J. Lee , J. Kim, J. Na, and S. Sohn, “ESES/j-Crypto and its application” published in Proceedings of IEEE International Symposium on Industrial Electronics (ISIE 2001), Pusan South Korea, pp. 1373-1377, vol.2, 2001.

[15]. Article: “VIA's New JAVA Cryptography Service Provider (VIA JCP) Implementation”, downloaded from https://embeddedjava.dev. java.net/community_site/articles /viajcp_embedded javacommunity_article.pdf, visited on Jan. 13, 2009.

[16]. OC4J: Oracle Fusion Middleware “Feature Overview: Security” http://www.oracle.com/technology/deploy/security/as_security/appserversec_1 0gr 3_fov.htm, visited on Dec 23, 2008.

[17]. T. Kern, “Next Generation Usability of Cryptography Combining FlexiProvider and JCrypTool” http://ww w.cdc .informatik.tu-darmstadt.de/reports/reports /Tobias_Kern .Diplom.pdf, Technische Universität Darmstadt Fachb- ereich Informatik Fachgebiet Kryptographie und Computer Algebra, July 2008.

[18]. NSP, “NSP’s Layered Approach to Network Security”, http://www.networksecurityprovider.com/approach.html, [visited: December 2008].

[19]. Entrust Entelligence, “Entrust Entelligence® Security Provider”, http://ww w.businesssignatures.com/resources/download.cfm/21204/Security%20Provider_oct25_ 07_web .pdf/?start, [visited: December 2008].

[20]. Mr. Jeff Hughes, Dr. Martin R. Stytz, “Advancing Software Security – The Software Protection Initiative”, AT-SPI Technology Office, AFRL/SN, 2241 Avionics Circle, WPAFB, OH 45433-7320, http://www.preemptive.com/ documentation/SPI_software_Protection_Initative.pdf, December 2001.

[21]. Stephen Thomas Kent, “Protecting externally supplied software in small computers”, Laboratory for Computer Science, Massachusetts Institute of Technology, Sep, 1980.

[22]. G. Naumovich, and N. Memon, “Preventing piracy, reverse engineering, and tampering”, published in the IEEE Computer Society, Vol. 36, No. 7, pp. 64-71, 2003.

[23]. Stytz, M., and J. Whittaker, “Software protection: Security’s last stand?”, published in IEEE Security and Privacy, pp. 95–98, January, 2003.

[24]. Sivadasan, Praveen, Lal, P Sojan, Sivadasan, Naveen, “JDATATRANS for Array Obfuscation in Java Source Code to Defeat Reverse Engineering from Decompiled Codes”, http://cdsweb.cern.ch/record/1128190, September 2008.

[25]. Nicol, D.M., Okhravi, H., “Performance analysis of binary code protection”, published in proceeding of Simulation Conference, ISBN: 0-7803-9519-0, 2005.

[26]. UltraProtect 1.05, Risco Software, Inc., http://wareseeker.com/publisher/risco-software-inc./31829/, [visited: October 2009].

[27]. Grugq, and Scut, “Armouring the elf, Binary encryption on the unix platform”, www.phrack.org/phrack/58/p58-0x05, 2001.

[28]. P.C. van Oorschot, “Revisiting Software Protection”, published in the proceeding of 6th International Conference of Information Security, ISC 2003, Bristol, UK, pp.1–13, October 2003.

[29]. Y. Chen , R. Venkatesan , M. Cary , R. Pang , S. Sinha and M. H. Jakubowski, “Oblivious Hashing: A Stealthy Software Integrity Verification Primitive”, LNCS, Springer Berlin / Heidelberg, Volume 2578/2003, ISBN-0302-9743, pp 400-414, 2003.

[30]. MSDN Microsoft, “Introduction to Code Signing”, http://msdn.microsoft.com/ en-us/library/ms537361(VS.85).aspx, [visited: October 2009].

[31]. Zhang, X.N, “Secure Code Distribution”, published by IEEE Computer Society, Volume: 30, Issue: 6, pp. 76-79, June 1997.

[32]. Trusted Computing Group, Incorporated, “TCG Specification Architecture Overview”, Specification Revision 1.4, August 2007.

[33]. V. Costan, L. F. G. Sarmenta, M. Dijk, and S. Devadas, “The Trusted Execution Module: Commodity General-Purpose Trusted Computing”, published in The eighth Smart Card Research and Advanced Application IFIP Conference, London, UK, pp. 133-148, September 2008.

[34]. Microsoft, “Next-Generation Secure Computing Base (NGSCB)”, http://www.microsoft.com/resources/ngscb/default.mspx, [visited: September 2009].

[35]. Amit Singh, “Trusted Computing for Mac OS X”, http://osxbook.com/book /bonus/chapter10/tpm/, written in October 2006.

[36]. B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Request for Comments: 3748, Extensible Authentication Protocol (EAP), June 2004.

[37]. Urien P, Badra M, Dandjinou M., “EAP-TLS smartcards, from dream to reality” published in the Proceedings of the Fourth Workshop on Applications and Services in Wireless Networks, ASWN 2004, Boston, MA, 2004.

[38]. White Paper, Aladdin Knowledge Systems Ltd., “Authentication Tokens: The Key to Secure PCs and Data”, http://www.aladdin.com/, [visited: July 2010].

[39]. Lexar Media, Inc., Lexar® JumpDrive SAFE S3000, http://www.lexar.com/, [visited: July 2010].

[40]. Athena Smartcard Solutions Inc., ASECard for Windows Smart Card Starter Kit, http://www.athena-scs.com/privacy.asp, [visited: July 2010].

[41]. Collective Software, LLC, “AuthLite”, http://www.collectivesoftware.com/, [visited: July 2010].

[42]. Smart Card Alliance, http://www.smartcardalliance.org/, [visited: July 2010].

[43]. McAfee SECURE, “Data Sheet”, downloaded form http://www.mcafee.com /us/local_content/datasheets/ds_endpoint_encryption.pdf downloaded on September, 2009.

[44]. Norton 360, “All in one Security”, http://www.symantec.com/norton/360, [visited: January 2009].

[45]. Symantec, “White Paper: Enterprise Security, Critical System Protection and Endpoint Encryption for the PCI Data Security Standard”, downloaded form http://www.symantec.com/business/products/whitepapers.jsp?pcid=pcat_security&pvid=endpt_encryption_1#, [visited: September 2009].

[46]. eCryptfs, “eCryptfs – Enterprise Cryptographic Filesystem”, https://launchpad.net/ecryptfs, [visited: May 2009].

[47]. Ubuntu File Browser, http://www.ubuntu.com/, [visited: July 2009].

[48]. AxCrypt, “Introduction and Features”, Axantum Software AB, Sweden, http://www.axantum.com /AxCrypt/Features.html, [visited: October 2009].

[49]. Crypt Manager, “User Guide", http://www.ubuntugeek.com/crypt-manager-an-encrypted-folder-manager-for-ubuntu-linux.html, [visited: October 2009].

[50]. E. Filiol, J. Fizaine, “White paper, Open Office v3.x Security Design Weakness”, Laboratorie de virology et de cryptologie operationnelles, France, March, 2003.

[51]. Information Security Awareness, “Browser Threats” http://infosecawareness.in/isea/women/browser-threats Last modified: 2009-05-08.

[52]. Microsoft Corporation, “Using Microsoft Office 2003 security features”, http://www.microsoft.com/protect /products/yourself/office2003.mspx, [visited: May 2009].

[53]. Sami Rautiainen “OPENOFFICE SECURITY” published in the processing of the 13th Annual Virus Bulletin International Conference (VB2003), Toronto, Canada 25-26, [visited: September 2003].

[54]. Document Security Systems, Inc. http://www.documentsecurity.com/, [visited: July 2009].

[55]. Jakarta Slide, http://jakarta.apache.org/slide/architecture.html, [visited: July 2009].

[56]. Open Source Project, JLibrary, “Tutorial: Security Management”, http://jlibrary.sourceforge.net/1/tut6.html, downloaded on July, 2009.

[57]. M. Alhammouri, S. Muftic, “A Model for Creating Multi-level-security Documents and Access Control Policies”, published in the Proceeding of SSI´2006 - 8th Intl Symposium on System and Information Security, Sao Jose Dos Campos, Sao Paulo, Brazil, November, 2006.

[58]. M. Alhammouri, S. Muftic, “A Design of an Access Control Model for Multilevel-Security Documents” published in the Proceeding of The 10th International Conference of Advanced Communication Technology (ICACT 2008), pp 1476-1481, Feb 2008, Phoenix Park, Korea.

[59]. A. Kapadia, “A Case (Study) For Usability in Secure E-mail Communication” Security & Privacy, IEEE Volume 5, Issue 2, March-April 2007, pp. 80-84.

[60]. S. Kaushik, D. Wijesekera, and P. Ammann, “BPEL orchestration of secure webmail” published in the Proceeding of the 3rd ACM workshop on Secure web services Alexandria, Virginia, USA, pp. 85-94, 2006 ISBN:1-59593-546-0.

[61]. C.-J. Tsai, S.-S. Tseng, and H.-T. Cheng, “Intelligent E-mail Management System” published in the Proceeding of the IEEE Conference on Systems, Man, and Cybernetics, 1999 SMC '99, volume 5, pp. 824-829, ISBN: 0-7803-5731-0 Tokyo, Japan 1999.

[62]. https://esecure.evolve-online.com/solutions/esecure-mail.php,[visited: December 2008].

[63]. H.-M. Sun, B.-T. Hsieh, and H.-J. Hwang, “Secure E-mail protocols providing perfect forward secrecy”, published in the Proceeding of the IEEE Communications Letters, Volume: 9, Issue: 1 pp. 58-60 ISSN: 1089-7798, Jan. 2005.

[64]. G. Kardas and E. Celikel, “A Smart Card Mediated Mobile Platform for Secure E-Mail Communication”, published in the Proceeding of the Fourth International Conference on Information Technology, 2007. ITNG’ 07 Volume , Issue, pp. 925 – 928, April 2007.

[65]. T. Ichimura, A. Hara, Y. Kurosawa, T. Ichimura, A. Hara, and Y. Kurosawa, “A classification method for spam e-mail by Self-Organizing Map and automatically defined groups”, published in the Proceeding of IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC. pp. 2044-2049 ISBN: 978-1-4244-0991-4, Montreal, QC, Canada, October 2007.

[66]. C. Masone and S. Smith, “Towards usefully secure E-mail”, published in the IEEE Technology and Society Magazine, Volume: 26, pp. 25-34, ISSN: 0278-0097 INSPEC, Spring 2007.

[67]. “SPAM Problems?”, http://www.uni.edu/its/us/faqs/E-mail/SPAM/aboutSPAM .htm, [visited: December 2008].

[68]. Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu, “The Ghost In The Browser Analysis of Web-based Malware”, published in the Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (HotBots ’07), Cambridge, MA, pp. 4-13, April, 2007.

[69]. White paper, “Making Sense Of Man-In-The-Middle, Strategies For Mitigating A Menacing Threat”, RSA, The Security Division of EMC, MITB WP 1009, January, 2010.

[70]. Roger A. Grimes, “Malicious Mobile Code, Virus Protection for Windows”, ISBN: 1-56592-682-X, Chapter 11, August, 2001.

[71]. Vincenzo Ciaschini and Roberto Gorrieri, “Contrasting Malicious Applets by Modifying The Java Virtual MACHINE, Security and Protection in Information Processing Systems”, Published in the proceedings of the 18th World Computer Congress, Toulouse, France, pp. 47-64, August, 2004.

[72]. Jianwei Zhuge, Thorsten Holz, Chengyu Song, Jinpeng Guo, Xinhui Han, and Wei Zou, “Studying Malicious Websites and the Underground Economy on the Chinese Web”, Published by Springer in Managing Information Risk and the Economics of Security, pp. 225-244, December, 2007.

[73]. Symantec, White Paper, “Critical System Protection and Endpoint Encryption for the PCI Data Security Standard”, [Online], http://www.symantec.com /business/products/whitepapers.jsp?pcid=pcat_security&pvid=endpt_encryption_1#, [visited: February 2010].

[74]. Norton 360, “All in one Security”, [Online], http://www.symantec.com/norton /360, [visited: February 2010].

[75]. AVG Internet Security, [Online], http://www.avg.com/us-en/homepage, [visited: February 2010].

[76]. avast, [Online], http://www.avast.com/free-antivirus-download#tab2, [visited: February 2010].

[77]. Snort, “The defacto standard for intrusion detection/prevention”, http://www.snort.org/, [visited: February 2010].

[78]. nCircle, “Proactive Network Security”, [Online], http://www.ncircle.com/ index.php, [visited: February 2010].

[79]. “Network Security Switch, Intrusion Prevention System and Policy”, REFLEX. http://www.reflexsecurity.com/, [visited: February 2010].

[80]. NESSUS. [Online],http://www.tenablesecurity.com/nessus/, [visited: February 2010].

[81]. Thomas Burg, “SSL Certificates and PKI in the NonStop World - and Other Worlds”, Published in The Connection, pp. 17-20, June, 2004.

[82]. Robert McMillan, “Mass Web attack hits Wall Street Journal, Jerusalem Post”, http://www.computerworld.com/s/article/9177904/ Mass_Web_attack_hits_Wall_Street_Journal_Jerusalem_Post, (June, 2010), [visited: February 2010].

[83]. M. Blaze, “A cryptographic file system for UNIX”, published in the proceedings of 1st ACM Conference on Communications and Computing Security, 1993.

[84]. G. Cattaneo, G. Persiano, A. Del Sorbo, A. Cozzolino, E. Mauriello and R. Pisapia, “Design and implementation of a transparent cryptographic file system for UNIX”, Technical Report, University of Salerno, 1997.

[85]. J. Hughes and D. Corcoran, “A universal access, smart-card-based, secure file system”, Atlanta Linux Showcase, October 1999.

[86]. E. Zadok, I. Badulescu and A. Shender. “Cryptfs: A stackable vnode level encryption file system”, Technical Report CUCS-021-98, 1998.

[87]. RSA laboratories, “What is public-key cryptography”, http://www.rsa.com/ RSALABS/node.asp? id=2165, [visited: November 2009].

[88]. T. Dierks, E. Rescorla, RFC No. 5246, “The Transport Layer Security (TLS) Protocol”, SSL v3, August 2008.

[89]. E. Rescorla, A. Schiffman, RFC: 2660 “The Secure HyperText Transfer Protocol”, August 1999.

[90]. W. Tworek, G. Chiesa, F. Dahm, D. Hinkle, A. Mason, M. Milza, A. Smith, “Lotus Security Handbook”, First Edition, April 2004.

[91]. FIPS PUB 140-2, “Security Requirements for Cryptographic Modules”, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900, May 25, 2001.

[92]. M. Alhammouri, S. Muftic, “Management of Groups and Group Keys in Multi-level Security Environments”, 26th International Conference Computer Safety, Reliability, and Security (SAFECOMP 2007), Nuremberg, Germany, ISBN: 0302-9743, pp. 75-80, September 2007.

[93]. M. Alhammouri, S. Muftic, “A Model for Creating Multi-level-security Documents and Access Control Policies”, published in the proceeding of SSI´2006 - 8th Intl Symposium on System and Information Security, Sao Jose Dos Campos, Sao Paulo, Brazil, November, 2006.

[94]. White Paper: “Xerox DocuShare Security Features”, August 2009.

[95]. “Digital Signatures & Rights Management in the Acrobat Family of Products”, A guide for administrators and advanced users for Acrobat® Family of Products 9.x, Modification date: August 26, 2009.

[96]. Article, Signed Applets - How to sign an applet, http://forums.sun.com/thread.jspa?threadID=174214, [visited: March 2009].

[97]. OpenOffice, “OpenOffice.org “XML File Format, Technical Reference Manual”, Version 2, December 2002, Sun Microsystems, Inc.

[98]. Microsoft Corporation, “Winlogon and GINA”, http: //msdn.microsoft.com/ en-us/library/aa380543(VS .85).aspx, , [visited: October 2009].

[99]. CVS Concurrent Version System, http://www.cvshome.org/eng/, [visited: January 2009].

[100]. Hudson Server, http://hudson-ci.org/, [visited: March 2009].

[101]. XACML OASIS , S. Cantor, J. Kemp, R. Philpott, E. Maler, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0” OASIS Standard, March 15, 2005, http://docs.oasis-open.org/security/saml/ v2.0/.

[102]. http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/trun_genericsvr_create.html [visited: May 2010].

[103]. Peffers, Ken , Tuunanen, Tuure , Rothenberger, Marcus A. and Chatterjee, Samir, “A Design Science Research Methodology for Information Systems Research”, Published in the Journal of Management Information Systems, Vol. 24, No. 3, pp. 45 – 77, Winter 2008

Appendix A

Following are the most popular and common attacks on digital assets in distributed and integrated environments.

Name of Attack	Description
Tampering	An attacker may alter information either stored in local files, database or is sent over public network.
Eavesdropping/Information Disclosure	This type of attack occurs when attacker gains access in the data path and gains access to monitor and read the messages.
Repudiation	Sender tries to repudiate, or refute the validity of a statement or contract which is sent by him/her.
Elevation of Privileges	An attacker may access unauthorized to information and resources
Man-in-the-Middle Attack	This type of attack occurs when an attacks infiltrates the communication channel in order to monitor the communication and modify the messages for malicious purposes. This type of attack can be more effective when client and server are exchanging time constraint data. For example: stock exchange data, real time data
Replay Attack	A replay attack is defined as when an attacker or originator sends a valid data with intention to use it maliciously or fraudulently.
Identity Spoofing	Identity spoofing occurs when an attacker impersonates the users as the originator of the message in order to gain access on a network.
Directory Attack	In this type of attack, attackers some how get access in user's information stored in database or directory (LDAP) and steal critical information like password or keys.
Reverser Engineering Threat	In this type of attacks, an attacker can use some tools or techniques to either get knowledge about weakness of software or illegal modification.
Differential Analysis Threat	When new versions are released, a differential analysis of the new and old version would indicate where differences in the code exist. This differential analysis would provide an opportunity to attackers to locate the position where he/she can attack and integrate malicious code.

Viruses and Worms	Viruses and worms are very common and well known attacks. These are piece of code that decrease the performance of hardware and application even these malicious codes corrupts files on local file system. This situation becomes more critical when such malicious code damages the inner logic of software to make it abnormal or worthless.

Appendix B

This appendix explains different elements of an XML file, which is a format of protected encapsulated software module.

SPS	Starting element of XML file.
Version	Current version of software protection file format.
Content-Type	This element indicates the type of contents in contents field which helps secure execution environment to process it accordingly. Some examples are SignedAndEnvelped, Enveloped, Signed etc.
Encapsulation-Standard	This field contains information about encapsulation standard, like PKCS7.
SM-Type	This element contains information about the type of software modules. These types can be Native, Configuration or External. The secure execution environment handles these files according to the type of software module.
SM-Name	Name of protected software modules/file.
Content-Description	This field provides descriptions of encapsulated modules and is an optional field.
Contents	This element contains the actual contents of software modules protected using cryptographic and encapsulation standards defined in element Content-Type and encapsulated in the standard, mentioned in Encapsulation-Standard element.

Appendix C

C 1. Message Formats

In this protocol, each message class tag (MCL) and fields (tag/value) are part of the message body. The value of each field is in hexadecimal format and each field (tag/value) is separated by a space from other field. The following are the message formats of Federation and Validation Protocols. These are based on modified ANSI X9.26 specifications:

C 1.1. Unilateral Registration Protocol

- *Registration-Request*

Registration-Request = **CSM(MCL/TFV** ORG/orgVal **RCV**/rcvVal **SID**/sessionIdD **URL**/localsem.com **IP**/130.237.158.18 **PORT**/9411 **GSF**/(sessionID|URL|valIP|valPort) **SLA**/)

ORG	Identity of message sender
RCV	Identity of message recipient
SID	Session Id
URL	URL of SEM server’s domain
IP	IP address of SEM server
PORT	Port number of SEM server
GSI	The output of the GSF generated by initiator
SLA	Service Level agreement (optional parameter)

- *Registration-Reply*

Registration-Reply = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID **GSF**/sessionID **VURL**/localsem.com)

ORG	Identity of message sender
RCV	Identity of message recipient
SID	Session ID
GSR	The output of the GSF generated by responder
VURL	The certified domain name of SEM server in the format of PKCS#7SignedData

C 1.2. Mutual Registration Protocol

- *Registration-Request*

Registration-Request = **CSM(MCL/TFV ORG**/orgVal **RCV**/rcvVal **SID**/sessionIdD **URL**/localsem.com **IP**/130.237.158.18 **GSF**/(sessionID|URL|valIP))

ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
URL	URL of the SMI server
IP	IP Address of SMI server
GSI	The output of the GSF generated by initiator

- *Registration-Reply-Request*

Registration-Reply-Req = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID **GSF**/sessionID **MESSAGE**/Registration-Request)

ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
GSR	The output of the GSF generated by responder
MESSAGE	Registration request message but the GSF field is generated by responder

- *Registration-Reply*

Registration-Reply = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID **GSF**/sessionID)	
ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
GSI	The output of the GSF generated by initiator. The session Id is the input of GSF function.

C 1.3. Management of Federation

- *AddReferences*

AddReferences = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID Ref/RefList **GSF**/sessionID|RefList)	
ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
Ref	List of domain names separated by ‘;’
GSI	The output of the GSF generated by initiator. The session ID and list of references is the input of GSF function.

- *DelReference*

DelReference = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID Ref/valRef **GSF**/sessionID|valRef)	
ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
Ref	Domain name
GSI	The output of the GSF generated by initiator. The session ID and list of references is the input of GSF function.

- *Deregistration*

Deregistration = **CSM(MCL/SRF ORG**/orgVal **RCV**/rcvVal **SID**/sessionID Ref/valRef **GSF**/sessionID|valRef)	
ORG	Identity of the message sender
RCV	Identity of the message recipient
SID	Session ID
URL	Domain name of SEM server
VURL	Certified URL or Domain Name
GSI	The output of the GSF generated by initiator. The session ID and Domain Name are the input of this function.

Appendix D

We implemented basic and complex security functions using our Java Security Provider, Java Crypto Provider (using standard Java security packages and Bouncy Castle Provider), and MS - Crypto Services Provider (C#) in order to compare their performance. We used following test bed and cases for comparison. Furthermore, we also provide detailed results of our experiments in Section D 3.

D 1. Characteristics of Platform

Processor: Inter Core (TM)2 Duo T7300 @ 2.00 GHs

RAM: 2.00 GB

Operating System: Windows XP with SP-3

Integrated Development Environment: Eclipse 3.3

D 2. Sample Code

In this section, we only included code for Symmetric Key Cryptography as a sample in order to justify our claim that our system is better for rapid development of security services in applications.

D 2.1. Sample Code using Generic Security Provider

```
public long DESTest()
{
   long startTime = System.currentTimeMillis();
   // Instantiate an object of SymmetricKey
   SymmetricKey sk = new SymmetricKey();
   // Plain text
   String secretString = "Attack at dawn!";
   // Encrypt plain text using reference of instantiated symmetric key
      object
   byte encrypt[] = sk.encrypt(secretString.getBytes());
   // Decrypt cipher text using reference of instantiated symmetric
      key object
   byte dec[] = sk.decrypt(encrypt);
   long endTime = System.currentTimeMillis();
   return (endTime-startTime);
}
```

D 2.2. Sample Code using Java Crypto Provider (using standard Java security packages and Bouncy Castle Provider)

```
 public long  DESTest()
{
   long startTime = System.currentTimeMillis();
   // Define references of Cipher class for encryption and decryption
   Cipher ecipher=null;
   Cipher dcipher=null;
   SecretKey desKey=null;
   // plain Text
   String secretString = "Attack at dawn!";
    byte[] enc=null;
   try
   {
       // Generate symmetric key
```

```
        desKey = KeyGenerator.getInstance("DES").generateKey();
        // Instantiate cipher and decipher objects
        ecipher = Cipher.getInstance(desKey .getAlgorithm());
         dcipher = Cipher.getInstance(desKey .getAlgorithm());
         // Initialize cipher and decipher objects
         ecipher.init(Cipher.ENCRYPT_MODE, desKey );
         dcipher.init(Cipher.DECRYPT_MODE, desKey );

         // Encrypt plain text using reference of cipher object
         enc = ecipher.doFinal(secretString.getBytes());
         // Decrypt cipher text using reference of decipher object
         byte[] utf8 = dcipher.doFinal(enc);

   } catch (BadPaddingException e) {
   } catch (IllegalBlockSizeException e) {
   }  catch (NoSuchPaddingException e) {
       System.out.println("EXCEPTION: NoSuchPaddingException");
   } catch (NoSuchAlgorithmException e) {
       System.out.println("EXCEPTION: NoSuchAlgorithmException");
   } catch (InvalidKeyException e) {
       System.out.println("EXCEPTION: InvalidKeyException");
   }
   long endTime = System.currentTimeMillis();
   return (endTime-startTime);
}
```

D 2.3. Sample Code using MS - Crypto Services Provider (C#)

```
public static long DESTest()
{
   long l = System.Environment.TickCount;
   string PlainText=@"Attack on Dawn";
   // Create an instance of  DESCryptoServiceProvider which is a
   // DES key
   DESCryptoServiceProvider key = new DESCryptoServiceProvider();
   // Create a memory stream for Crypto Stream.
   MemoryStream ms = new MemoryStream();
   // Create a CryptoStream using the memory stream and the CSP DES
   // key.
   CryptoStream encStream = new CryptoStream(ms,
                        key.CreateEncryptor(), CryptoStreamMode.Write);

   // Create a StreamWriter to write a string to the stream.
   StreamWriter sw = new StreamWriter(encStream);
   // Write the plaintext to the stream.
   sw.WriteLine(PlainText);

   // Close the StreamWriter and CryptoStream.
   sw.Close();
   encStream.Close();

   // Get an array of bytes that represents
   // the memory stream.
   byte[] buffer = ms.ToArray();

   // Close the memory stream.
   ms.Close();

   // Create a memory stream to the passed buffer.
   ms = new MemoryStream(buffer);

   // Create a CryptoStream using the memory stream and the
   // CSP DES key.
   encStream = new CryptoStream(ms, key.CreateDecryptor(),
                                             CryptoStreamMode.Read);

   // Create a StreamReader for reading the stream.
```

```
       StreamReader sr = new StreamReader(encStream);

       // Read the stream as a string.
       string val = sr.ReadLine();
       // Close the streams.
       sr.Close();
       encStream.Close();
       ms.Close();
       long l2 = System.Environment.TickCount;
       return (l2-l);

    }
```

D 3. Results

The detailed results of our experiments are shown in the following tables.

Greneric Security Provider

No.	SymmetricKey Cryptography (DES)	AsymmetricKey Cryptography (RSA-1024)	HASH Functions (SHA-1)	Certificate Generation	PKCS7 SignedData	SMIME SignedData
1	16	32	0	578	78	47
2	15	47	0	516	94	47
3	16	32	0	515	110	46
4	15	32	0	468	63	62
5	16	31	0	562	94	47
6	15	47	0	625	110	31
7	16	16	0	875	62	63
8	16	31	0	547	109	63
9	16	32	0	547	93	94
10	16	32	0	656	31	31
Avg. (ms)	**15.70**	**33.20**	**0.00**	**588.90**	**84.40**	**53.10**

Java Crypto Services Provider/Bouncy Castle

No.	SymmetricKey Cryptography (DES)	AsymmetricKey Cryptography (RSA-1024)	HASH Functions (SHA-1)	Certificate Generation	PKCS7 SignedData	SMIME SignedData
1	375	594	0	765	94	125
2	391	281	0	500	109	125
3	406	359	0	563	110	125
4	375	235	0	672	125	109
5	375	188	0	500	109	156
6	375	422	0	1750	94	125
7	359	437	0	579	110	140
8	360	312	0	672	94	125
9	375	250	0	469	125	125
10	375	203	0	453	109	140
Avg. (ms)	**376.60**	**328.10**	**0.00**	**692.30**	**107.90**	**129.50**

MS Crypto Services Provider using C# (.NET)

No.	SymmetricKey Cryptography (DES)	AsymmetricKey Cryptography (RSA-1024)	HASH Functions (SHA-1)	Certificate Generation	PKCS7 SignedData	SMIME SignedData
1	0	359	0	407	78	78

2	0	422	0	328	78	78
3	0	235	0	343	79	62
4	0	250	0	453	78	78
5	0	203	0	563	78	62
6	16	203	0	297	78	78
7	0	203	0	343	79	78
8	0	219	0	250	78	62
9	16	484	0	235	78	62
10	0	328	0	281	78	63
Avg. (ms)	**3.20**	**290.60**	**0.00**	**350.00**	**78.20**	**70.10**
