CryptoNET: Software Protection and Secure Execution Environment

Abdul Ghafoor†, Sead Muftic†

†The Royal Institute of Technology, DSV, Borgarfjordsgatan 15, SE-164 40, Kista, Sweden

Summary
The software modules are key component of information technology. Most of software owners and users are concerned about the protection of software modules against reverse engineering, illegal tempering, program-based attacks, BORE (Break Once Run Everywhere) attack and unauthorized use of software. Some efforts have been made to protect software modules using cryptographic techniques like digitally signed Java Applet which is verified by Java Virtual Machine (JVM) before execution.
However today, software modules are not protected using strong encryption techniques and extended cryptographic functions, because existing execution environments do not support to process and execute protected software modules. Normally, such environment should act as a middleware platform between software modules and operating system. This paper describes protection of software modules which is based on strong encryption techniques, for example public key encryption and digital signature. These protected software modules are encapsulated in our designed XML file which describes a general syntax of protected software modules. In addition, our designed system also securely distributes software modules to authorized user. Secure software distribution system is based on well established standards and protocols like FIPS-196 based extended strong authentication protocol and SAML based authorization security policies. We also designed secure execution environment which is capable to execute signed and encrypted software modules, 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 asynchronous sessions.

Key words:
Strong authentication, format of protected software modules, secure software distribution, reverse engineering, cryptographic functions

1. Introduction

Software is the foundation of information technologies that businesses, enterprises, educational institutions, and critical security infrastructure owners and operators rely upon for their most imperative and critical operations. For this reason, everybody is concerned with protection and security of software and its protection against exploitation by attackers, who maliciously disrupt it, illegally modify or use it. Software industry, which was a US$ 303.8 billion industry in 2008 with an annual increase of 6.5% [1], is also concerned with misuse and illegal distribution of their products. They are actively participating in software protection activities to solve software distribution, legal use and modifications, but still they are bearing billions of dollars loss annually [2].

Various security technologies and applications have been developed and deployed so far for software assurance and protection. Some examples of such technologies and applications may be: Intrusion Detection Systems (IDSs), Intrusion Prevention Systems (IPSs), anti-virus programs, firewalls, etc. However, these security solutions do not provide satisfactory level of software assurance and protection. Furthermore, these solutions do not completely eliminate software vulnerabilities which are exploited by attackers.

Security researchers and standardization organizations are working in three directions to achieve a high level of software protection against reverse engineering, illegal tempering, program-based attacks, BORE (Break Once Run Everywhere) attack [3] and unauthorized use of software. The first group is working on defining and implementing digital rights management and software licensing laws to protect intellectual property rights. Examples are Software and Information Industry Association (SIIA) [4] or World Intellectual Property Organization (WIPO) [5]. The second group is working on software security, recommending that software should be secure by design in order to provide certain level of security. Examples are Software Security Assurance (SSA) [6], Cigital [7], etc. The last group is working on obfuscation and cryptographic protection techniques which are based on some technical solutions: Examples are Trusted Platform Group [8], Microsoft's Software Protection Platform [9], LaGrande Technology [10], etc.

In this paper, our approach is based on the last category of software protection solutions. Currently, the most popular method is verification of software against viruses. An example of such software is signed Java Applet [11] which implements limited security functions. However today, software modules are not protected using strong encryption and extended cryptographic

Manuscript received February 5, 2010
Manuscript revised February 20, 2010

functions, because those modules require security enhanced execution environment. Normally, such environment should act as a middleware platform between software modules and operating system. Examples may be Java Virtual Machine [12], Common Language Runtime [13] or some execution environments implemented as an extension of operating system.

During this research we analyzed existing solutions and commercial products (outlined in Section 2) and found that none of the current solutions are capable to protect software modules using strong encryption techniques. Furthermore, current middleware platforms do not support execution of encrypted, signed and encapsulated software modules, enveloped in a standard cryptographic format.

Recognizing this gap, our paper describes the design and implementation of a comprehensive software protection system which provides software confidentiality, tempering resistance, and even protection against illegal coping and distribution. Furthermore, we designed an XML (Extensible Markup Language) file which describes a general format of protected software modules to support alternative cryptographic syntax and standards. In addition, we also 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 such as certification protocol, secure session handling, use of smart cards, and Security Assertion Markup Language (SAML) authentication protocol. We believe that our system is an effective solution against reverse engineering, software tempering, illegal copying and BORE attack based on authentication, access control, integrity, and confidentiality security services for software modules. In particular, we also solved a critical issue of secure execution of such protected software.

2. Overview and Analysis of Current Software Protection Solutions

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 mentioned in the first section. We structured those 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.

2.1. Protection of Software Modules

Software Protection Initiatives (SPI) [14] group 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 to protect the value-added software.

Reverse engineering of software is the first threat to generate and modify source code. Some decompilers, like Compuware Numega SoftICE [15], URSoftware W32Dasm [16], Datarescue Interactive Disassembler Pro (IDA) [17] and Oleh Yuschuk's OllyDbg [18] are quite effective for reverse engineering of Windows-based software [19]. One of the first solutions against reverse engineering and illegal modifications of software executable modules was explained by Kent in 1980 [20]. 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 [21][22][23]. In 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 [24], 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 [25] uses asymmetric key to protect software executables against piracy and illegal distribution. A hybrid software protection technique, described in [26][27], 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.

2.2. Secure Software Distribution

Most of the systems for secure software distribution rely on the Internet technologies which focus on integrity of software and authentication of clients. Vendor of open source and free software, available on the Internet, generates the hash value of executable modules and uploads it to Internet with static hash value [28]. 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 [29]. 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" [30] 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. Furthermore, the paper [30] also mentioned that S/MIME can be used to securely distribute the software.

2.3. Controlled Execution Environment

Currently, a well known software execution environment is Java Virtual Machine which verifies signed Java applets before execution. Trusted Computing Group [31] 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 [32] which uses cryptographic functions in a low cost memory chip. Microsoft is working on the concept of "Next-Generation Secure Computing Base (NGSCB)" [33] 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 keep cryptographic keys so that makes them impossible to retrieve. The aim of this environment is to execute software in a secure environment. On the other hand, Apple is working on incorporating a Trusted Platform Module (TPM) into their Apple Macintosh line of computers for the integrity and confidentiality of the software modules [34].

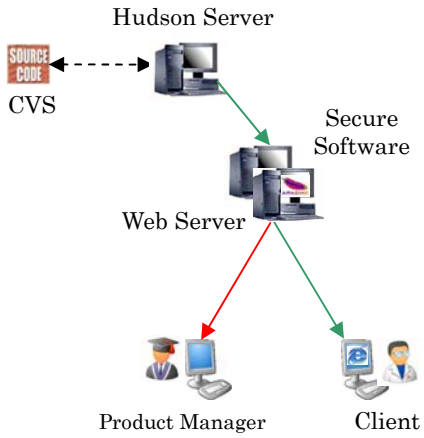

Fig. 1 Components of Software Protection and Distribution System

2.4 Analysis of Existing Solutions

Analyzing exiting solutions and techniques, we found that these solutions use obfuscation (non cryptographic technique), limited security features, and host dependent secure environments, while some solutions require special hardware and components which are economically expensive and usually not available on a large scale. Contrary to these solutions, our approach is to address authentication, authorization, confidentiality, and integrity services using extended security features and functions. Our solution also solves some of problems addressed by software security group, like software consistency. Other then support to standard security mechanisms and services, the distinctive features and properties of our solution are the following:

(i). protection of software modules using strong encryption techniques, for example public key encryption and digital signature;
(ii). XML file which describes a general syntax of protected software modules;
(iii). secure software distribution using strong authentication and SAML based authorization security technologies;
(iv). secure execution environment which is capable to execute signed and encrypted software modules, supports standard security services and network security protocols, such as transparent handling of certificates, use of smart cards, single-sign-on protocol, strong

authentication protocol, and secure asynchronous sessions.

3. Overview of System Components and Functions

In a professional software development environment source code is managed by some versioning system. In our environment, we use Concurrent Versioning System (CVS) for this purpose. As shown in Fig. 1, the CVS is linked with Hudson Server which generates binaries from source code when Product Manager launches the build process. At the end of the build process, Hudson Server publishes the newly generated binaries at the Secure Software Distribution (SSD) Server which are available to clients through web server.

The interested client securely downloads protected software products (in this paper we refer to software modules as software products) after following authentication and authorization procedures and protocols explained in next section. Furthermore, client also downloads secure execution environment which is capable to execute signed and encrypted software modules.

In this system, SSD Server and Web Server are incorporated into our global security infrastructure (not shown in Fig. 1) and we assume that the following security infrastructure servers exist in domain and are fully functional:

(i). Certification Authority (CA) Server which issues and distributes X.509 certificates to all components;

(ii). IDentity Management System (IDMS) Server which manages identities of different resources and clients;

(iii). SAML Policy Server, also known as Policy Decision Point (PDP), responsible for creation of SAML tickets, authentication policies and policy sets, and making decisions based on the SAMLAuthenticationRequests; and

(iv). Strong Authentication (SA) Server which performs strong authentication with clients and passes SAML ticket to clients. Furthermore, it also acts as the bridge between SAML Policy Server and Policy Enforcement Point (a component of SSD Server) for handling SAML Authentication and Authorization requests and responses.

SSD Server protects software modules using strong protection techniques. At the initial start up it requests and receives three certificates (digital signature, key exchange and non- repudiation certificates) from the local CA server. If smart card is installed, it generates keys in a smart card and stores received certificates in it. Otherwise, it stores them in a local certificate database. Furthermore, the SSD Server contains PEP component which interacts with security infrastructure servers for single-sign-on authentication and authorization.

In this system we also specified an XML file which describes general syntax of protected software modules which supports many different cryptographic syntax and standards. Our SSD Server uses PKCS7SignedAndEnveloped cryptographic syntax. For that purpose, it digitally signs software modules using private key of digital signature certificate and envelopes them using the public key of client's key encipherment certificate. After successful cryptographic enveloping in Public-Key Cryptography Standards (PKCS7) format, it generates XML file shown in Fig. 2

```
<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>
```

Fig. 2 Different elements of XML file which contains information about protected software modules and security standards

representing this specific scenario. It contains information about the cryptographic standard used to protect and envelop software module. The detail specification and description of each element of the XML file are explained in Appendix A.

In addition to protection of software modules, it signs and encrypts fingerprint (see Section 4) and identity of a client and embeds in protected software. At the client side, this secured information is used to protect software against illegal copying and distribution. In addition, the SSD Server is also responsible to digitally sign the Secure Execution Environment which is extended with security features and functions those are explained in next section. The SSD Server also keeps encrypted log its all of actions. It is also capable to receive log history from clients for analysis and detection of misuse and anomaly.

4. Operations of the System

Registration of clients in IDMS is performed according to the procedure described in [35].This is performed by using our registration web pages. The registration information is sent to Web Server using SSL protocol. Upon successful registration, our Web Sever displays information message "*Registration request received. Please logon using user name and password after some time*". The security manager views the stored data and authorized the client to download protected software from web server. At this phase, the software vender may ask for payment and licensing policies which are not disused in this research paper. Security manager creates policy in SAML Policy Server specifying that the registered client can download software modules. If the security manager rejects the request, the web browser does not allow the user to login into web server and sends back a deny message. Using the login web page, a user downloads signed ActiveX module from the web server which interacts with smart card, installed on client machine, to perform strong authentication, cryptographic functions and storage of credentials after opening card using PIN or PIN+ fingerprints. Upon successful login, web browser activates ActiveX to fetch certificates from CA Server and performs Strong Authentication with SA Server in order to fetch SAML identity assertion as described in [36].

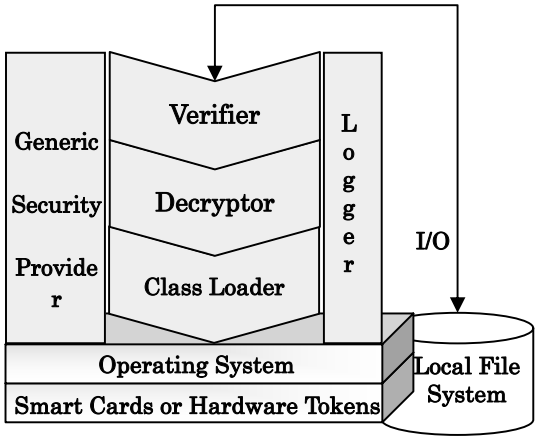

Fig. 3 Components of Secure Execution Environment

In order to download Secure Execution Environment (SEE), the client sends SAML ticket to Web Server by using HTTP POST request. The Web Server consults the SAML Policy Server to verify the SAML ticket and evaluate the authorization polices. If SAML Policy Server permits to download SEE then the Web Server redirect request to SSD Server. The client downloads and installs the SEE. The setup verifies the signature in order to ensure that SEE is downloaded from authenticated server and is not tempered during the downloading phase. In this solution, we assume that the client machine has basic execution environment like JVM which is capable to verify the signed SEE.

The SEE fetches SAML ticket from smart card for single-sign-on authentication and authorization purposes in order to fetch protected software modules from SSD Server. Upon successful, authentication and getting permission, the SSD Server establishes secure session with SEE according to the protocol described in [36]. The SEE generates fingerprint which contains processor id, serial number of hard disk and distinguished name. The SEE signs fingerprint and securely sends to SSD Server which embeds it in protected software modules as explained in section 3. SEE downloads signed, encrypted and enveloped software modules from SSD Server and stores in the local file system. The SEE comprises five components as shown in Fig. 3. These components are: Generic Security Provider, Verifier, Decryptor, Class Loader and Logger. The detailed security functions and features of Generic Security Provider are explained in [37] which are used by different components for cryptographic functions. The Verifier component

fetches protected software modules from local file system. It verifies the signature of software modules by processing header of XML file. If verification process fails then the SEE logs the event and stops its execution. Upon successful verification, the software modules are passed to Decryptor. Initially, the Decryptor decrypts the fingerprint and identity file and compares with the local fingerprint and identity. After successfully comparison, the Decryptor decrypts the value of contents element of the XML file according to the standards mentioned in the header of XML file. The output of this process is plain files and Decryptor takes any of the following action upon the values of SM-Type element:

- If SM-Type is Native then it handover to Class Loader for loading modules in memory. Examples are java classes, executable jars
- If SM-Type is Configuration then it only use required information on-fly. Examples are fingerprints and identity file, server configuration file.
- If SM-Type is External then it saves in temporary directory and will be loaded by specific class loader at its execution time. At the closing time, SEE deletes these temporary files. Examples are *so*, dlls, exe files.

The Logger component of SEE, maintains the log of each action performed by other components. The SEE periodically submits the log to SSD Server for anomaly and misuse detection of software modules.

4. Conclusion

Currently, most of software is being protected using digital signature techniques which are only used for verification. In addition, some key based protection solutions also designed but some of them are not effective against reverse engineering, illegal tempering, program-based attacks, BORE (Break Once Run Everywhere) attack. We adopted holistic approach for protection of software modules, secure software distribution and secure execution environment which is based on well established security standards, tested cryptographic techniques, strong encryption functions and extended security features. These are: general syntax of protected software modules in XML file, FIPS-196 based extended strong authentication, SAML based authorization security policies, execution of encrypted and digitally signed software modules, transparent handling of certificates, use of FIPS-201 compliant smart cards, single-sign-on protocol and secure asynchronous sessions.

This designed solution provides protection of software modules for individual users but still there is problem in distribution of software protection key in grouped environment which will be addressed in our future research.

Appendix A

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

(i). ***SPS:*** Starting element of XML file.
(ii). ***Version:*** Current version of software protection file format.
(iii). ***Content-Type:*** This element indicates the type of contents in contents filed which helps secure execution environment to process it according. Some examples are SignedAndEnvelped, Enveloped, Signed etc.
(iv). ***Encapsulation-Standard:*** This filed contains information about encapsulation standard like PKCS7.
(v). ***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.
(vi). ***SM-Name:*** Name of protected software modules/file.
(vii). ***Content-Description:*** This filed provides the descriptions of encapsulated modules and is an optional filed.
(viii). ***Contents:*** This element contains the actual contents of software modules which are protected using cryptographic and encapsulation standards defined in element Content-Type and encapsulated in standard mentioned in Encapsulation-Standard element.

References

[1] Datamonitor, "*Datamonitor's Software: Global Industry Guide*", report code- DO-4959, http://www.infoedge.com/product_type.asp?product=DO-4959, April, 2009.

[2] Business Software Alliance, "*Fifth annual BSA and IDC global software, piracy study 2007*", www.bsa.org.

[3] Arxan Technologies, "*Protecting .NET Software Applications*", Arxan Best Practices, White Paper, http://www.softwaremag.com/pdfs/whitepapers/protecting-NET-software-applications-wp.pdf?CFID=14965377&CFTOKEN=16715129

[4] Software and Information Industry Association (SIIA), http://www.siia.net/index.php?option=com_content&view=article&id=159&Itemid=6, visited in Oct, 2009

[5] World Intellectual Property Organization (WIPO), "*World Intellectual Property Indicators*", http://www.wipo.int/portal/index.html.en, 2009

[6] "*Software Security Assurance*", State-of-the-Art Report (SOAR) Joint endeavor by Information Assurance Technology Analysis Center (IATAC) and Data and Analysis Center for Software (DACS), July 31, 2007

[7] Gary McGraw , "*Building Secure Software*", http://www.cigital.com/papers/, Oct 15, 2008

[8] Trusted Computing Group, http://www.trustedcomputinggroup.org/resources

[9] Cori Hartje, Feature Story, "*Microsoft's Software Protection Platform: Protecting Software and Customers from Counterfeiters*", Microsoft Genuine Software Initiative, http://www.microsoft.com/presspass/features/2006/oct06/10-04SoftwareProtection.mspx

[10] Intel Corporation, LaGrande "*Technology Architectural Overview*", 252491-001, ftp://download.intel.com/technology/security/downloads/LT_Arch_Overview.pdf, September 2003

[11] Daniel Griscom, Article, "*Code Signing for Java Applets*", http://www.suitable.com/docs/signing.html last updated on May 5, 2009.

[12] Sun Developers Network (SDN), Documentation, "*Ergonomics in the 5.0 Java[tm] Virtual Machine*", http://java.sun.com/docs/hotspot/gc5.0/ergo5.html visited in Oct, 2009

[13] Microsoft, Visual Studio Developer Center, .NET Framework Developer's Guide, "*Common Language Runtime Overview*", http://msdn.microsoft.com/en-us/library/ddk909ch.aspx visited in June, 2009

[14] 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.

[15] Compuware Numega SoftICE, "*The Advanced Windows Debugger*", ftp://94.141.61.146/pub/info/Information/Books/!!!Computer%20books!!!/stuff/Crack/SoftICE/SoftICE.pdf, viewed as HTML, read in Sep, 2009

[16] URSoftware W32Dasm,, http://www.softpedia.com/get/Programming/Debuggers-Decompilers-Dissasemblers/WDASM.shtml, last updated on March 11th, 2003.

[17] Datarescue Interactive Disassembler Pro (IDA), http://www.hex-rays.com/idapro/, website updated on January 1, 2009

[18] Oleh Yuschuk's "*OllyDbg Version 2.0*" , http://www.ollydbg.de/, updated on march 28, 2009

[19] 12Sabuj Pattanayek "*Toughening Software Protections*" http://www.defacto2.net/ documents.cfm?id=238

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

[21] Memon, Preventing]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.

[22] Stytz, M., and J. Whittaker, "*Software protection: Security's last stand?*", published in IEEE Security and Privacy, pp. 95–98, January, 2003.

[23] 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, Sep, 2008

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

[25] UltraProtect 1.05, risco software,Inc., http://wareseeker.com/publisher/risco-software-inc./31829/

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

[27] 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

[28] 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

[29] MSDN Microsoft, "*Introduction to Code Signing*", http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx, visited in Oct, 2009

[30] Zhang, X.N, "*Secure Code Distribution*", published by IEEE Computer Society, Volume: 30, Issue: 6, pp. 76-79, Jun 1997

[31] Trusted Computing Group, Incorporated, "*TCG Specification Architecture Overview*", Specification Revision 1.4 2nd August 2007

[32] 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, Sep, 2008

[33] Microsoft, "*Next-Generation Secure Computing Base (NGSCB)*", http://www.microsoft.com/resources/ngscb/default.mspx, visited in Sep, 2009

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

[35] A. G. Abbasi, S. Muftic, "*CryptoNET: Integrated Secure Workstation*", published in International Journal of Advanced Science and Technology, pp. 1-10, Vol. 12, November, 2009.

[36] A. Gahafoor, S. Muftic, G. Schmölzer, "*CryptoNET: Secure Federation Protocol And Authorization Policies for SMI*", published in the IEEE International Conference on Risks and Security of Internet and Systems, pp. Oct, 2009

[37] A. Ghafoor, S. Muftic, G. 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

Abdul Ghafoor received the M. S. (Network Technologies), from National University of Sciences and Technology (NUST), Islamabad, Pakistan in 2004. After working as a Lecturer in the Dept. of Computer Science (2002-2004), F. G. Postgraduate College, Islamabad and as a Lecturer and IT Manager in Kohat University of Science and Technology (2004-2006), he was appointed as a faculty member at National University of Sciences and Technology (NUST), Islamabad. In 2007, he started his PhD at the Royal Institute of Technology in the area of Network Security. His research interest includes secure technologies, security protocols, smart cards based technologies, and web security. He is a member of KTH-SecLab Sweden, Information Security and Distributed Computing Group SEECS Islamabad, IEEE Graduate Student, Stockholm Research Association, and IPID Sweden.

Sead Muftic has been working in the area of computer security for more than 30 years. He is professor of Computer Security at the Department of Computer and Systems Sciences (DSV), The Royal Institute of Technology, Stockholm, Sweden and also research professor at The Michigan Technical University USA). Dr. Muftic was the member of the Permanent Stakeholders Group (PSG), an expert advisory group to ENISA (European Networks and Information Security Agency), director of the EU COST–11 Security project, consultant to VISA, World Bank, Siemens and other international organizations. Dr. Muftic is the author of three international books and about 100 research and scientific papers published in journals or presented at international conferences.