uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Xuebing Qing

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Policy-Driven Access Control for Mobile Environments

TITRE DE LA THÈSE / TITLE OF THESIS

Prof. C. Adams

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Prof. L. Peyton

Prof. A. Matrawy

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Policy-Driven Access Control for Mobile Environments

Xuebing Qing

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the M.Sc. degree in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Computer Science
University of Ottawa
Ottawa, Ontario K1N 6N5
Canada

Your file Votre référence
ISBN: 978-0-494-49266-6
Our file Notre référence
ISBN: 978-0-494-49266-6

ABSTRACT

SOA (Service Oriented Architecture) enables interoperability between heterogeneous systems or domains. By complying with proper SOA specifications that are usually XML based, a service provider can provide service to any group of service consumers. A service consumer or requester can ask for a service from any service providers that satisfy its service criteria without knowing any implementation specifications that the service providers adopt. However, a transaction or a set of message exchanges between mutually interactive parties cannot be realized prior to the establishment of a trusted relationship that is embodied in granting or denying access to certain resources under the governance of any involved party.

This thesis proposes XACML-based, policy-driven, access-control architecture and a key exchange/authentication framework/protocol at layer 7. The thesis examines a number of authorization and secure-channel establishment issues found not only in a normal service requester-provider environment, but also in an environment that involves mobile requesters. The proposal is derived from a careful study of authorization request/decision rendering procedures in a number of realistic business and leisure scenarios. The scenarios are characterized mainly by (a) the distribution of access control; (b) the indeterminacy of the access control model that is implemented; (c) the distributed storage of policies/organization guidelines/regional laws; (d) the lack of common representations of basic access-control elements such as subject (principle) IDs/resource IDs; and (e) the need to establish secure communication channels over unsecured public networks between security domains.

In order to solve these problems, some novel concepts are raised, such as (a) a subject ID mapping service; (b) meta policy server (MPS); (c) reverse authorization; and (d) private reputation server. Furthermore, a peer-to-peer security handshake protocol infrastructure (KEAML/KEAML-KE) at layer 7 and its implementation are presented.

KEY WORDS

Access control, Authorization in mobile environments, Subject ID mapping, Meta Policy Server, Reverse Authorization, XACML, Key Exchange and Authentication, KEAML/KEAML-KE, Standard security handshake protocol, XML

ACKNOWLEDGEMENTS

First, my sincere thanks go to my supervisor, Dr. Carlisle Adams, who offered not only academic guidance, but also the encouragement and trust that helped me to continue my research and finish my thesis.

My wife, Yun Jiang, who has been a strong and constant supporter of my pursuing research in ECE (electronic and computer engineering), deserves my heartfelt gratitude.

Obtaining a master's degree has been not only my dream, but also that of my father and mother. Without their continual support and encouragement, this thesis would have not been possible.

I would like to thank my friend, Lei Li, a master's student at the University of Waterloo, for the excellent discussions and endless encouragement.

Yun Jiang, the thesis is dedicated to you. I owe you so much that I cannot ever make it up to you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 INTRODUCTION

This chapter first examines the scenarios in which the targeted problems occur. Next, the problems are extracted and analyzed. Finally, a list of requirements and assumptions are summarized for this thesis.

1.1 Scenarios

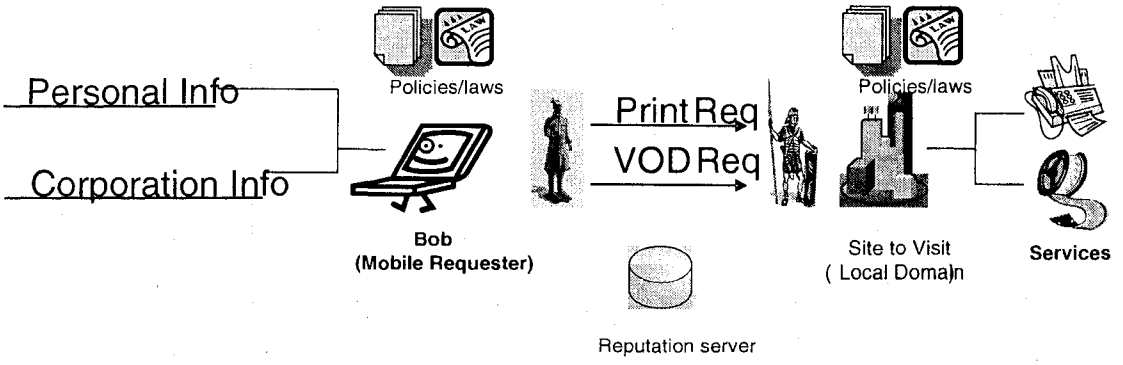

Figure 1-1 Scenario in Question

The situation in Figure 1-1 involves both business and leisure scenarios that can be summarized as: Bob, with a mobile device (i.e., mobile authorization requester), who works for New Age Business Inc., travels to Paris and wants to use the services available at the site, or the *local domain*. The mobile device may contain or have access to Bob's personal and confidential info, and/or the confidential information of his employer (i.e., company confidential data). In the typical scenarios that follow, we observe the site can be a place Bob has never visited before (e.g., a hotel or conference center) or the site of the entity in which he has some level of trust, such as a client's company location. The services may vary from simple document printing, to VOD (Video On Demand), to a complex pervasive computing application. Depending on which services Bob requests, three categories of scenario occur:

- **Scenario 1 - Personal activities**

Wherever he is, Bob wants to talk to his wife in audio, and print his bank statements. These activities involve Bob's personal confidential information being delivered to or through the services provided by the local domain that Bob has no (or limited) trust in.

- **Scenario 2 - Personal activities that are subject to regional regulations and laws**

Wherever he is, Bob wants to watch movies through an Internet video-on-demand service. This activity requires the service provider getting some personal information from Bob (personal information disclosure) and making sure the transaction does not flout the regional regulations and laws.

- **Scenario 3 - Business activities**

Wherever he is, Bob wants to print confidential documents from the company he works for, and then have a confidential videoconference with his colleagues. These activities involve corporate confidential data being delivered to or through the services provided by the local domain that Bob and his company have no (or limited) trust in.

A variant of Scenario 3 (Scenario 3a) happens between two business entities. From a security perspective, the only difference from Scenario 3 is that (a) the requester is either one of the two parties, and (b) a Security SLA (Service Level Agreement) has been reached between the two before the interaction.

The goal to be achieved in the above scenarios (and in similar scenarios) is that a proper authorization request must be formed by the mobile requester (Bob), and a corresponding authorization decision must be made in the local domain (the security system governing the service providers at the site). In particular, the domain protecting personal information, the domain protecting corporate interests, and the local domain protecting the site's resources / services must each be able to collect proper and sufficient information in order to render an appropriate decision. According to the XACML [37], the necessary information includes the subject attributes/resources/action identifier extracted from the access request, the context data from the environment, security and privacy policies, and for Scenario 2, the relevant regional or national laws and regulations.

However, further investigation into these scenarios unveils some major access-control problems that are not addressed by (or not in the scope of) XACML or any other XML-based security specifications. The next section discusses these problems.

1.2 Problems and Relevant Requirements

The following problems are discussed in the context of an XACML application. In particular, we assume that one of the most interoperable platforms – the XACML-based Attribute Based Access Control (ABAC) system – is used [64].

1.2.1 Problem 1 – Unrecognizable Subject ID and Indeterminacy of AC Model

Since the XACML-based ABAC system is used in the scenarios we observe, Bob has to first form an XACML authorization request by using a subject ID that is recognizable by the local domain. However, for most cases in a mobile environment (and this often applies to the interactions between fixed domains as well), a subject ID issued by one domain (in this case, Bob's home domain) may not be recognized by another (e.g., the local domain). Hence, an XACML request cannot be processed properly; that is to say, a mobile requester cannot be authorized by a local domain until s/he obtains a valid subject ID from the local domain, or a subject ID that is recognized by both domains.

The above subject ID issue has the same nature of so-called identity problem, which is not unique to the mobile environment, but is common in many scenarios where interactions between enterprise applications are involved. SSO (Single-sign-on)-based Identity Management seems to solve the problem [54] by connecting domains together beforehand and assigning credentials to principles that are recognized by all domains. Yet to achieve this, the flexibility introduced by the mobility is sacrificed; neither does it work very well for temporary subject IDs that could last as briefly as one day. For example, Bob does not want to apply for a SSO account just for a day's stay in the hotel; however, the hotel may not be able to afford to become SSO–enabled. Even if it is SSO enabled, it may not want to create an SSO account for Bob because he will be staying in the hotel for only one night.

Even if the subject ID issue is solved completely, an XACML access request still cannot be structured and sent. First, the mobile requester does not know the AC (access control) model employed by the local domain (so that the specific attributes required in the request are unknown). Second, non-standard terms or vocabulary may be used even among systems that adopt the same access model. For example, Bob may have a "manager" role in his role-based home domain, but he cannot simply put "manager" as the value of the role-ID attribute in the request to a role-based local domain, because the value "manager" may not be interpreted the same way as in the home domain, or the value may not even exist in the local domain.

1.2.2 Problem 2 – Locate and Apply Applicable Policies During Authorization

One of the basic security requirements for two interactive security domains is that the interaction should be conducted in such a way that neither party's security is comprised. This can usually be done by locating and applying *all applicable* security and privacy policies, which are combined to reach proper decisions [37]. In the scenarios we discuss, once a local domain receives an acceptable authorization request (i.e., the subject ID etc. is understandable and other request attributes also make sense to the AC model deployed in the local domain) from a mobile user, the first issue it faces is where to find *all* policies that are *applicable* for the mobile requester and the action/access s/he is asking for, especially if they are stored in a distributed fashion. In Scenario 3, the local doman must find (a) *all* the policies to avoid violating the regulations of the region where the mobile requester is from; (b) the security guidelines of the company that the requester works for (New Age Business Inc.); and (c) *all applicable* policies so the security requirement of the service provider residing in the local domain will not be compromised.

Although the issue does exist for all policy-driven security systems, it does not get enough attention from the researchers. XACML depicts the infrastructure that specifies the logical location to store policies (PAP) and suggest the way to locate applicable policies by using indexing based on the combination of subject ID/resource ID/Action ID. However, neither XACML nor other relevant security specifications describe how this infrastructure fits into a distributed system composed of several security domains. In order to locate all the policies associated with (or translated from) real world regulations, laws and company security guidelines, a configurable "super policy" redirector and store, or a Meta Policy Server (MPS, see Section 3.5.3) should be sitting in the security domain. Moreover, the information about the security

requirements of a service provider should be configurable and stored somewhere in the domain, which can be referenced when retrieving policies for the service provider.

1.2.3 Problem 3 – Enforce Regulations/Laws and Security SLAs

All three typical scenarios involve interaction between at least two security domains, which are likely constrained by different legislation and regulation systems. In Scenario 2, for example, the local domain requires the regional laws and regulations that are applicable to Bob to be accessible and executable as normal policies. In Scenarios 3 and 3a, corporate security guidelines/ standards and security SLA (Service Level Agreement) come into play.

Security Service Level Agreements enable security interoperability among domains (Scenario 3a), while federal/regional/company regulations/laws provide high-level security standards that override or guide operational/functional security (access control) policies (Scenario 3). Therefore, incorporating them into the authorization decision-making system is extremely important for the success of an access control system. Unfortunately, this problem is relatively new compared with the others; little research work has been done on it. To solve this problem, the following questions need to be addressed - how and where can a Security SLA (Service Level Agreement) be implemented between domains so that it is executable and usable? How and where can security configuration information (such as from which authority to find regional laws needed for an authorization decision) be implemented? The proposed architecture in Chapter 3 addresses this issue.

1.2.4 Problem 4 – Protect the Requester's Confidential Data

Prior to sending personal/business confidential data to a service provider (Scenarios 1 and 3), the mobile requester has to check the corresponding privacy policies and/or corporate security guidelines, to make sure the service provider can be trusted, and the confidentiality and integrity of data will not be compromised. Similar to Problem 3, this problem is relatively new and only be found important in a mobile enviornment, since in a security domain where the service provider and the requester know each other, a trust relationship is already established before any interactions between them occur.

Simply matching security preferences/parameters between the mobile requester and the service provider may not be sufficient or appropriate, since it may result in an inability to find appropriate services, which is caused by incompatibility between security parameters and preferences. For example, an appropriate service may choose not to expose its security features as static parameters – it will then never be picked up even if it actually meets the requester's security requirements. Moreover, some security requirements are inexpressible with static security parameters. An example of such a security requirement would be "Only print my document when the printing service has not been used by anyone in the past 10 minutes."

1.2.5 Problem 5 – Privacy Threats in Distributed System

Privacy problem is not a new issue at all in the security area. However, two privacy threats are found when observing the scenarios mentioned earlier in a mobile enviornment. On one hand, because a mobile requester can easily end up using a number of services, it is hard for the requester to track all the disclosed information are not abused by service providers. Even worse, a service provider may not be forced to implement privacy protection in the system. On the other hand, a service provider, which is willing to incorporate privacy protection in the system, may not have the ability to do so. More detail is given below.

Sometimes a mobile requester has no choice but to send over his/her confidential personal information such as age, gender etc. to unknown service providers in local domains in order to get requested services. However, the service providers that obtain the privacy information from the requesters may abuse the confidentiality and disclose the privacy information to third parties without notifying the user. They do not mention this procedure in the privacy policy prior to the service conducted, and force the user to agree to an unreasonable privacy policy in order to be served. In addition, the service providers announce their compliance to certain regional/national privacy laws and assure the privacy right of the requesters. It is impossible for individual requesters to find out how their privacy information is used and which is the offending service provider if any disclosure of personal information occurred, because the requester may have used a variety of services from hundreds of service providers which collected his/her privacy information. In general, the enforcement of the five principles of privacy [19] is impossible for an individual requester. On the other hand, the service provider may not be able to provide, either financially or technically, a reasonable IT system/administrative system as compliant with the five principles of privacy by law [19].

1.2.6 Problem 6 – Applicability of Reputation Data

The lack of trust between first-time-interacting parties can be overcome by using the other party's reputation data that were collected during other or previous activities. For example, a mechanism can be introduced to all security domains in the scenarios we are discussing so that Bob can query the reputation of a server to find out how trustworthy the provider is before sending confidential data out for the service. On the other hand, the service provider can retrieve Bob's reputation data from the server and use it to render the authorization decision.

Some reputation data are only applicable to specific services where they are collected (e.g., book returning history), while some are generic yet not suitable or precise enough for all situations (e.g., financial credit history cannot be used to measure a person's credit on book loaning activities). The applicability of reputation data, however, is ignored during their collection in many applications and hence, the reputation data are stored and used improperly.

Even though this problem is quite noticeable, it is never mentioned and addressed in previous literature. In this thesis, we present a solution with the proposed architecture in Chapter 3.

1.2.7 Problem 7 – Set up Secured Communication Between Parties

In the proposed infrastructure that we will discuss in Chapter 3, in order to process the access request from the mobile user, the "super policy" redirector or a super server of the local domain may need to acquire the mobile requester's subject attributes from the requester's home domain, through a secure channel over the Internet. The secure channel to convey the subject attributes cannot be established, however, until the two parties agree upon a key with which to protect their communications.

It is not uncommon that an enterprise application-level business entity residing in one security domain needs to be engaged in protected information exchanges across unsecured public networks (such as the Internet) with a number of other entities from separate security domains in an ad-hoc fashion. Well-known HTTPS protocol will not solve the problem, since its client/server mode cannot be easily used in peer-to-peer scenarios as we observe here. More importantly, HTTPS, which is based on SSL/TLS, requires data to be descrypted at each intermediate network node, therefore, it does not provide end to end security. Because each node has to conduct

decryption/encryption, the transit performance drops down with the increase of the number of the hops (network nodes).

Neither having a trusted central authority to manage this activity, nor a proprietary security handshake protocol designed for particular applications, works. The former is often impractical for distributed security domains across the Internet, while the latter is targeted to particular applications or specific environments. Hence, proprietary protocols lack interoperability and cannot be massively deployed. Experience has shown that proprietary protocols are usually very weak and vulnerable to a variety of attacks. For example, a security information exchange protocol specific to machine-readable travel documents (MRTDs) from the International Civil Aviation Organization (ICAO), which involves setting up secure communication channels, is suffering from security problems [28].

Key establishment must be cryptographically linked to authentication to ensure that the resulting key is shared with the valid party instead of an impostor [17].

Surprisingly, there is no standard key establishment and authentiation protocol found at layer 7. It would be useful, therefore, to have a standardized protocol for mutual authentication and key establishment that is secure and yet sufficiently flexible to be deployed in a number of environments. This paper proposes such a protocol for the application layer.

It is desirable to have a common infrastructure/protocol on the enterprise-application level to standardize mutual key exchange and authentication that is resistant to a variety of attacks, scalable, and sufficiently flexible to be deployed in a number of different application environments. Unfortunately, such an infrastructure/protocol is not offered by any of the major standards efforts in XML security, including W3C (World Wide Web Consortium) and OASIS. In Chapter 4, we define an XML-based key exchange and authentication framework along with a protocol, with concepts such as two-phase negotiation, standardized key-exchange templates that resist attack, and the public components for Diffie-Hellman exchange, which are borrowed from ISAKMP/IKE (the layer 3 security framework and protocol for VPN). The proposed protocol also leverages the W3C XML Encryption and XML Signature specifications to allow field-level encryption and signing of KEAML protocol messages, where required.

1.3 Requirements and Assumptions

- **Independency from Access Model Implementations**

In the proposed infrastructure and relevant system methods, a security domain (e.g., the local domain in the aforementioned scenarios) can use any methodology to implement any access model such as BLP, Chinese Wall, MAC, DAC [58] etc.

- **Independency of Policy Storage/Update/Revocation**

The discussion of how policies are stored, updated and revoked is outside the scope of this thesis. However, the infrastructure and relevant system methods proposed in this thesis should not impose any limitations on policy storage/update/revocation. All policies should be able to be converted to XACML, or be written in XACML.

- **Applicable Policies**

The authorization for decision-making should depend not only on the policies that embody the requirements of the security domain itself, but also on the policies of the region that are translated from national laws, regulations, legislation, or provincial laws, whenever they are applicable (e.g., Scenario 2). Likewise, the decision processing should also apply the policies of the corporation or organization that the authorization requester works for, if s/he is doing business for the company (e.g., Scenarios 3 and 3a). Applying these policies or not would be defined by the policies about policies, or meta policies. For example, if the meta policies say that the requester could watch a specified movie but only if his/her region's law allows him/her to do so, then the nationality of the requester should be retrieved to get the corresponding region's laws or policies for the decision rendering. Yet if the meta polices say that s/he could watch a specified movie no matter whether the requester meets the requirements of the regional laws where s/he is from, then the regional law will not be checked.

- **Hierarchical Policies and Policy Combination Algorithms**

Policies in a security domain or among the security domain's distributed policies storage spots, *may* be organized and stored in a hierarchical structure, i.e., the processing of incoming

authorization requests is governed by a set of policies that are organized hierarchically. In this case, a corresponding policy that defines the combination algorithms must be provided. In general, any sub-hierarchy of policies would be associated with a combination algorithm policy. A combination algorithm policy can be referred by many sub-hierarchies of policies. It is possible that there is only one combination algorithm policy for the whole hierarchy of policies of a security domain.

A hierarchical structure to store policies described above is recommended for the proposed infrastructure, since it simplifies the combination of policies with meta policies as well as policies that are translated from regional laws or security SLAs.

- **Software Agent on Mobile Device**

Finally, we assume that a software agent exists on a mobile device. A software agent will enable the mobile requester to interact with other security domains in order to complete an authorization procedure.

1.4 Audience

Individuals, security professionals or security researchers may be interested in this thesis. The infrastructure and relevant concepts as well as the key exchange protocols presented in the following chapters may be particularly attractive to those who have a special interest in the XACML-based policy-driven access control model.

1.5 Organization

This thesis proposes a generic architecture that examines the problems described in Section 1.2, with emphasis on the design and implementation of security protocol KEAML/KEAML-KE, which is part of the architecture.

First, the thesis examines terminology and related works in Chapter 2. Second, the proposed architecture is described in Chapter 3, where all important concepts and components of the architecture are introduced. In Chapter 4, the syntax and semantics of KEAML/KEAML-KE

are presented, followed by the implementation considerations of the protocol, which are covered in Chapter 5. Finally, Chapter 6 concludes the study.

Chapter 2 BACKGROUND AND RELATED WORK

Chapter 2 gives an overview of the terminology used in the thesis and then briefly introduces works related to security access control and key establishment protocols. Chapter 3 will discuss the proposal and Chapter 4 will describe the design of the communication protocol,.

2.1 Terminology

2.1.1 Local Domain

Local domain refers to the security management system deployed in the domain from which the mobile requester asks for services. *Local domain* has, at the very least, its own authentication and authorization subsystems.

2.1.2 Home Domain

Home domain is the counterpart of local domain. It refers to the security management system used in the security domain that the mobile requester is from or is governed by. Similarly, *home domain* has, at the very least, its own authentication and authorization subsystems.

2.1.3 Mobile Requester

Mobile requester refers to a service consumer who travels to a *local domain* and is authenticated by the local domain before it requests service or resource access that is governed by the local domain. The *mobile requester* does not know anything about the *local domain*. S/he does not know the type of access control model the *local domain* uses or how the model is implemented.

2.1.4 Subject ID

Subject ID is the identifier or identification of the subject (or principle) used by a *mobile requester* to make an authorization request to a *local domain*. A subject ID can be generated from a successful authentication of the mobile requester, or it can be a membership ID, employment ID, or even global ID that is signed and stored beforehand in the requester's device. Of course, no matter what the subject ID is, it has to be recognizable by the local domain. Please refer to Section 3.3 for an example.

2.1.5 Subject Attribute Authority (Subject AA)

Subject AA (Attribute Authority) is an authorized agent or service provider that distributes subject attributes upon request. The subject and the subject attributes must be specified in a subject attribute retrieval request. On one hand, the requester will not send a subject attribute retrieval request to subject AA if s/he does not trust subject AA; on the other hand, subject AA will not respond to a request from the requester which the subject AA cannot authenticate or authorize.

A *subject AA* usually resides outside a security domain.

2.1.6 Context/Environment Attribute Authority (Context AA)

Context or Environment AA is a dedicated agent or service provider that collects and distributes context/environment attributes on request. A context/environment AA usually sits inside a security domain and offers services to an internal requester, e.g., PDP (Policy Decision Point).

2.1.7 Resource Attribute Authority (AA)

Resource AA is an agent or service provider that distributes resource attributes specified by the requester. A *resource AA* sits inside a security domain and most of the time provides service to an internal requester, e.g., PDP.

2.1.8 Private (Local) Reputation AA

This is a new component introduced in the proposed architecture (Chapter 3).*Private Reputation AA* is a dedicated agent or service provider that collects and distributes reputation attributes upon request. A private reputation AA sits only in a security domain; it collects reputation data within the domain and provides such service to an internal requester, e.g., PDP.

2.1.9 Public (Third party) Reputation AA

This is a new concept introduced in the proposed architecture, versus *private reputation AA*. *Public Reputation AA* is an authorized agent or service provider that collects and distributes reputation attributes upon request from either a service provider or consumer. A *public reputation AA* can be deployed anywhere and can be accessed publicly, collecting reputation data from multiple sources.

2.1.10 Service Discovery

Service Discovery is a service to find the most appropriate service from given discovery or search criteria. *Service discovery* can be WSDL or public web service compliant, but its design can also be based on in-house standards

A mobile user has to perform a service discovery or similar operation to find a list of potential service providers. To simplify the system, we assume that any mobile requester is automatically entitled to conduct a service discovery.

2.2 XACML

XACML (eXtensible Access Control Markup Language) [37] provides a framework in which interoperability among different access-control systems becomes possible. As a language, XACML is extensible and expressive enough to meet all requirements from different access control models such as the Role-Based Access-Control Model (RBAC), the Chinese Wall

Access-Control Model, Bell-LaPadula (BLP), etc. In addition, XACML is designed to support the distributed access-control system.

2.2.1 Combinable Rule-based Policies

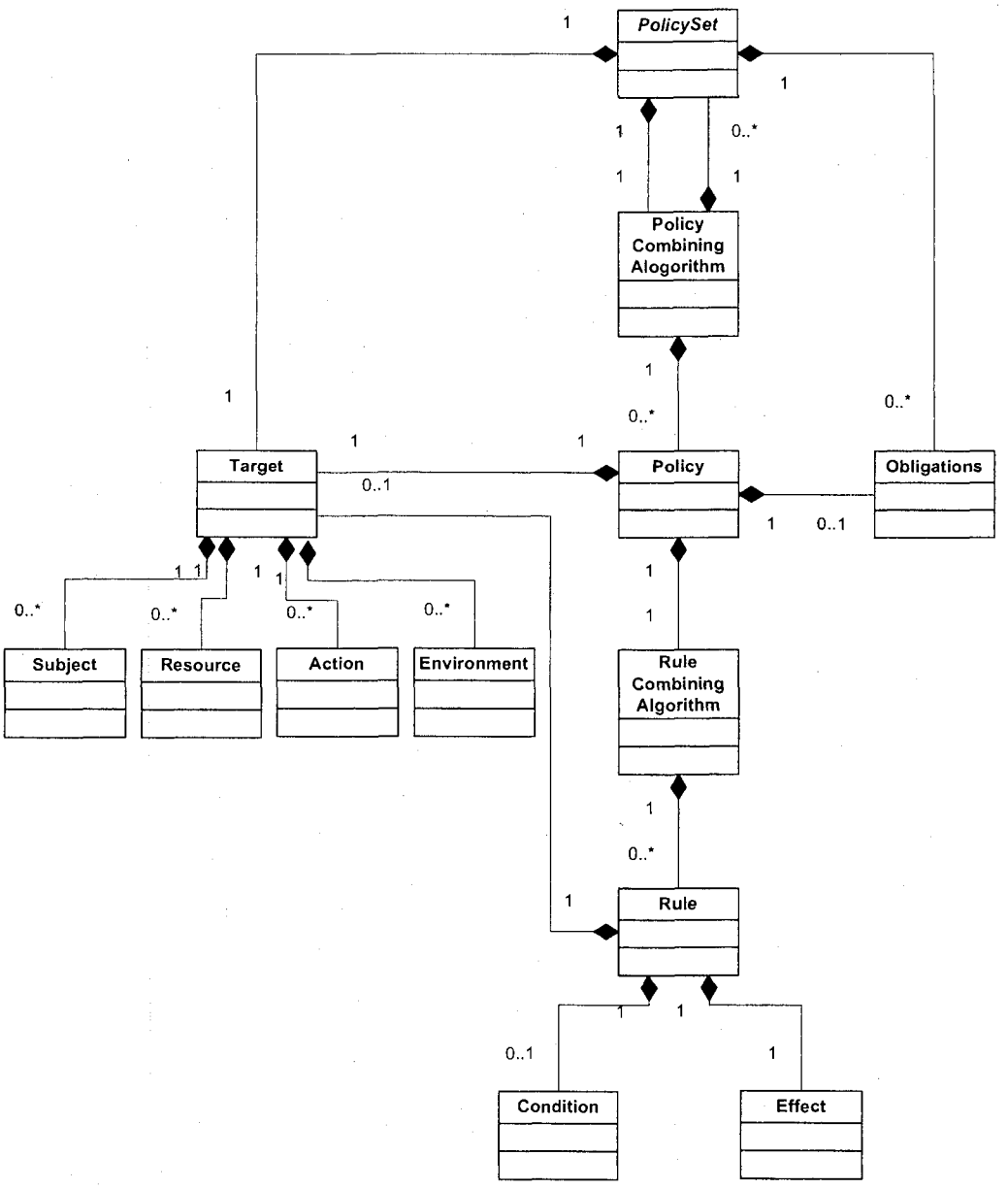

Figure 2-1 XACML Class Diagram [37]

An access-control policy is always about a security domain, which (subject) is entitled to do what (actions) to which objects (resources and services) under what conditions, particulary in given external situations (environments) such as time, temperature, place.

XACML follows the module-based structure to define policies as indicated in Figure 1-1. One *policy* is composed of *0* to *n rule*s, which is glued together through a *rule-combining algorithm*. In turn, policies may be grouped into a *policyset* through a *policy-combining algorithm*. This module-based design increases the reusability of policies in a security domain, which can be either distributed or self-complete (see Section 2.2.2). However, while a *rule* is the minimum composite unit, it is not the minimum retrievable unit. A security domain renders authorization decisions based on policies and/or *policysets*. A *policy* is the atomic operand during authorization decision calculation; hence, it is also the atomic element that is searchable and identifiable.

Target is the core of the module-based structure, not only because it is referred by *rule*, *policy* and *policyset*, but also because a *target* is composed of *subject* (who or the principle), *resource* (objects to be accessed), *action* and *environment* (external condidtion), among which *subject*, *resource* and *action* are the most important components for an access-control system. Moreover, XACML recommends the target as the indexing element for policy storage.

2.2.2 Data Flow Model and Distributed Policies

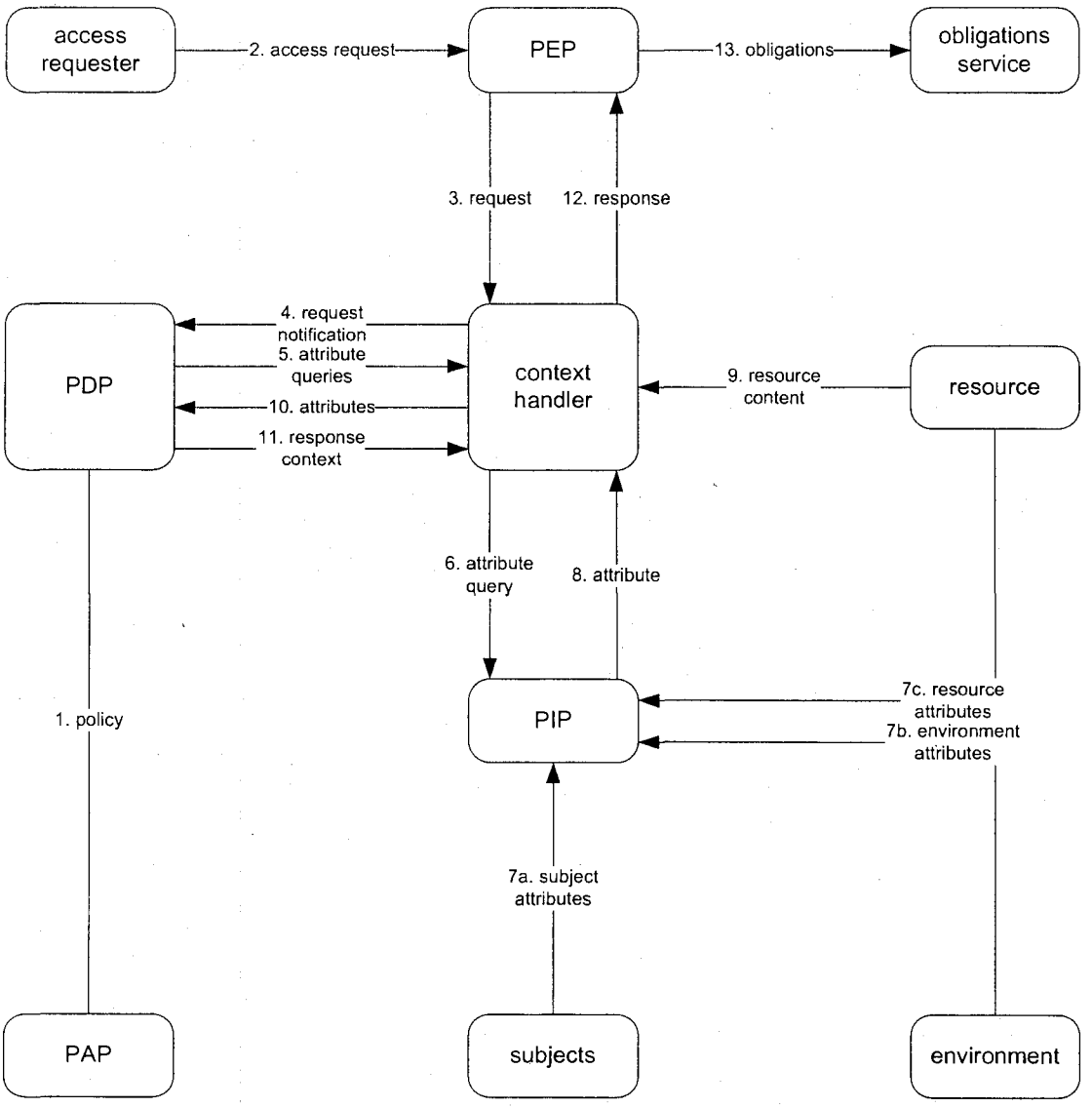

Figure 2-2 XACML Data Flow Model [37]

Figure 2-2 depicts the XACML data-flow model on which an XACML-based policy-driven system can be deployed. PAP (*Policy Administration Point*) stores and sends policies to PDP (*Policy Decision Point*) for authorization decision-making while PEP (*Policy Enforcement Point*) executes the authorization decision. PIP (*Policy Information Point*) collects the information required for the authorization decision, such as the subject information (*subject attributes*),

resource information (*resource content* and *attributes*), and environment information (*environment attributes*). Optionally, the requester is obligated to conduct specific actions as a condition to get the access requested (*obligation services*).

The model is by nature distributed, not only because XACML is in XML and therefore inherits the interoperability of XML, but because the model is so loosely defined that each of the PAP (Policy Administration Point), PDP (Policy Determination/Decision Point) and PEP (Policy Enforcement Point) has N-M mapping relationships with the others. That is to say, one point (e.g., PAP, PEP etc.) can be deployed in any location as long as it can cooperate with the rest through some communication infrastructure and protocols during an authorization decision rendering process.

Further, each policy or policyset is by definition identifiable and discoverable through its unique URI (i.e. *PolicyId*) or the locator across distributed security domains. No matter when a policy/policyset is authored and where deployed, it can be dynamically combined with predefined combination algorithms and enforced with other policies and policysets.

2.2.3 Context Awareness

XACML counts in environment parameters and external dynamics during access requests for authorization decision calculating. These environment parameters and external dynamics can be any context-attributes affecting the authorization decision rendering such as environmental, temporature, time, which are independent of a particular subject, resource or action [37]. In the loosely defined XACML data model (Figure 2-2), PIP (Policy Information Point) collects the context information and sends it to PDP for the request decision rendering.

2.2.4 Extensibility and Flexibility

XACML adopts XML schema's abstract and sub-classing mechanisms to lay out the foundation for extensibility. XACML allows arbitrary attributes for all major elements such as *subject*, *resource*, *action* and *environment*. Using arbitrary attributes makes this policy model flexible enough to suit any specific requirements from specific applications. For example, by introducing *role* as *subject*'s attribute, XACML supports an RBAC-based policy system [38]; by introducing *security label* attributes for both *subject* and *resource*, it becomes security-label-based policy

model. More specific, customized control can be introduced by using more application-specific attributes, such as *credit level/credit history,* etc.

In addition, XACML allows flexible naming and identification systems from the structural form of names (e.g., X.500, LDAP) to the location-centric description of structural document (e.g., XPath). Finally, XACML allows additions to predefined functions, operators, resource types, and environment event types.

2.2.5 Problems

XACML does not provide solutions to the problems listed in Section 1.2. First, XACML lacks a common vocabulary system which standardizes the basic terms in a policy-driven system such as action IDs, role names in a RBAC. These terms cannot be generalized across all access-control models. Second, XACML does not mention the conflict issues between policies in different security domains. In fact, it is impossible to have common arbitrators to solve the conflict problems, given the indeterminate nature of the mobile environment. Yet some security- service-level agreements (SLA) can be reached between interactive security domains to solve the problems. Section 1.2.3 discusses the problems from another perspective. The conflict issues can be solved by implementing security SLA as high-level policy which governs the operational and technical policies in low-level policy, in an XACML-based hierarchical policy system.

2.3 Attribute-based Access Control

The Attribute-based Access Control (ABAC) model [64] is an implementation instance of XACML in SOA (Service Oriented Architecture) rather than a new access-control model. The usage of attributes is emphasized in ABAC-based security domains. As shown in Figure 2-3, subject attributes, resource attributes and environment attributes are distributed by *subject AA*, *resource AA* and *environment AA* respectively. ABAC demonstrates the advantages of attribute-based access control over the traditional access-control models such as IBAC (Identity-based Access Control), RBAC (Role-based Access Control), LBAC (Lattice-based Access Control). It makes more fine-grained and flexible access control possible, while reducing the complexity of the access-control system [64].

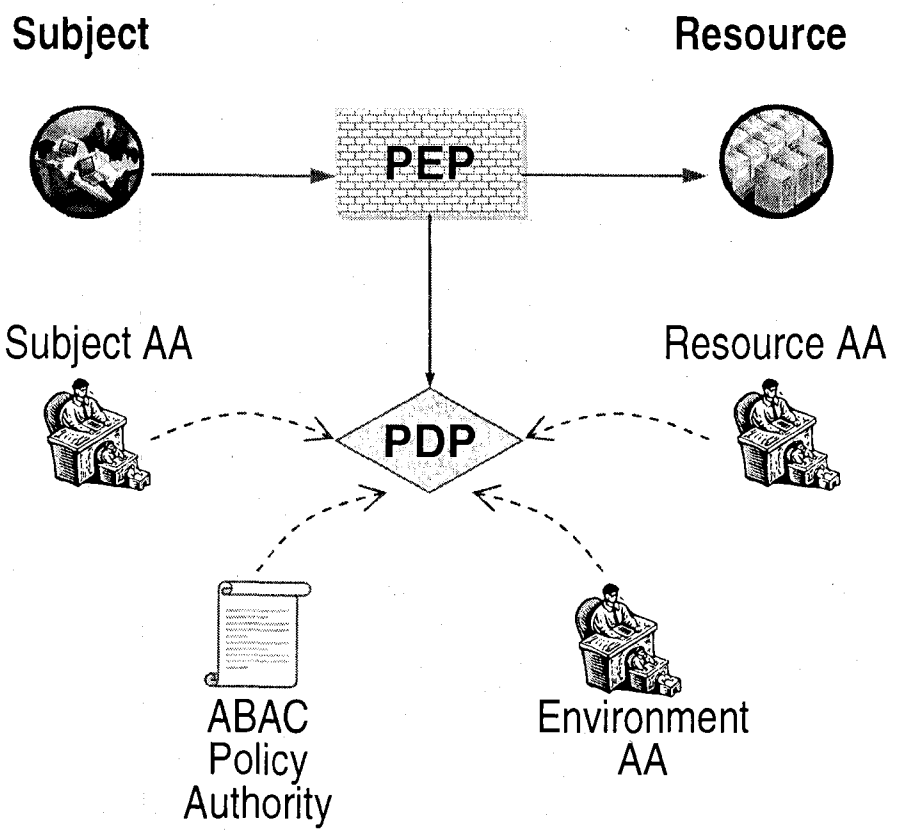

Figure 2-3 ABAC Authorization Architecture [64]

However, ABAC does not address the problems discussed in Section 1.2, either.

2.4 ISAKMP/IKE Version 1

ISAKMP (Internet Security Association and Key Management Protocol) [34] provides an extensible and flexible framework for peer-to-peer key management protocols, whereas the IKE (Internet Key Exchange) version 1 [25] is an extended instance of ISAKMP that defines a family of key exchange protocols. ISAKMP/IKE together enable secure communication at the IP level and allow IPSec-based VPN. ISAKMP/IKE is characterized by the following features:

- Standardized message *exchange* patterns;
- Separation of security parameters management from key exchange;

- *Diffie-Hellman key agreement* [49] and predefined *Oakley Groups* to provide *perfect forward secrecy (PFS)* and further standardization in IKE; and

- Two-phase mechanism to provide scalability.

2.4.1 Basic Concepts

ISAKMP/IKE is based on SKEME [31] and Oakley [40], which are derived from Station-to-Station (STS) protocol [17]. The goal of the framework and protocol is to generate a set of secrets that are exclusively shared between two negotiating parties that have not shared a secret before the negotiation.

In order to set up a secure channel, it is not adequate to have only key(s) exchanged between two parties. The two parties have to agree upon a set of common parameters that define how the key(s) can be used in information exchange. This set of common parameters, and the shared keys, form a security association (SA; see Section 2.2), which is the contract the two parties must follow during secure information exchanges.

In fact, the two parties must negotiate the parameters for keys before proceeding to the key-generation stage. That is to say, the two parties must execute two steps to negotiate security associations. Each step is called a negotiation. A negotiation is depicted as follows:

Step 1. Parameter Negotiation
INITIATOR → (list of SA proposals) → RESPONDER
INITIATOR ← (SA proposal chosen) ← RESPONDER
Step 2. Key Exchange and Mutual Authentication

In the above, the initiator is the party that begins the protocol messaging; the responder is the party that replies to the first message from the initiator. Step 1 not only sets up parameters for keys, but also determines which key exchange protocol to use (see Section 2.4.2). For the SA proposal, please refer to Section 2.4.5.

Since key exchange protocols are error-prone, the ISAKMP specification offers some well-designed key exchange templates, or *exchanges* (see Section 2.4.3). Any key exchange protocols that fit into these templates are considered secure.

For frequent and efficient key refreshing using public key cryptography without getting a significant performance penalty, a two-phase approach is adopted in ISAKMP, even if it has a higher start-up cost for simple scenarios than simpler approaches [34]. *Phase I* and *Phase II* are both ISAKMP negotiations that generate *Security Associations* (SA) (see Section 2.4.2). Phase I, which applies expensive asymmetric key generation, runs infrequently and is used to establish *ISAKMP SA* or *Phase I SA*, which contains symmetric keys. Phase II is conducted under the protection of the ISAKMP SA, so it can run faster and more frequently to generate many *Protocol SAs*, or Phase II SAs, which are used to protect data traffic between the two parties. The two-phase mechanism addresses the conflict of sharing secrets by using public cryptography and efficiently refreshing the shared secret, and solves the scalability problem. Even protocols designed for Phase 1 can be used in Phase 2. It is recommended that more efficient protocols are employed in Phase 2 in order to take advantage of the two-phase approach.

2.4.2 Security Association

Security Association (*SA*) is the agreement between two or more parties on how the parties adopt available security mechanisms and services to communicate securely. A *Security Association* is composed of information exchange protocol that both parties agree upon, including the keys, the corresponding algorithms and the parameters.

The parameters in a security association include:

- Cryptographic algorithms;
- Authentication methods, such as pre-shared key, signature, encryption, etc.;
- Parameters for the algorithms such as key size and initialization vector;
- How and on what the algorithm and key should be applied;
- Key exchange protocol and relevant parameters; and

- Key lifetime and key refreshment policy.

Phase I SA or ISAKMP SA is identified by the combination of two cookies generated by both parties (see Section 2.4.8), while Phase II SA is further identified by a message ID generated by the *initiator* and a Security Parameter Index (*SPI*) generated by each party during Phase II negotiation. SPI is unique to Phase II negotiation and is unidirectional.

In ISAKMP/IKE, the procedure of establishment of SA is triggered by an SA request sent from the initiator. The SA request usually contains a list of SA proposals; the responder either chooses one of the proposals and sends it back the initiator, or sends a failure notification. Once the negotiation of security meta parameters is complete, keys are exchanged between the two parties. SA is established only when keys are exchanged. After the establishment of SA, all parties must adhere to the agreed-upon SA to enable secure communications.

2.4.3 ISAKMP Exchanges

An ISAKMP exchange defines the number of messages sent by each party, the order of messaging, and the types of information (or payload) each message carries, plus the actions a party must take upon the receipt or sending of a message. There are two basic exchanges in ISAKMP, (a) identity-protection and (b) aggressive (See ISAKMP specification [34] for details). All ISAKMP exchanges can be used in either Phase I or Phase II; they may provide different security properties in a specific phase, and it is good practice to design exchanges specific to Phase II so they can benefit from the efficiency of Phase II.

2.4.4 IKE Modes

The structure of the IKE *mode* is similar to that of the ISAKMP *exchange*, which is composed of a number of messages, the order and payload type of the messages, as well as actions upon receipt or sending of a message.

The term *mode* in IKE is the same as *exchange* in ISAKMP. The IKE main mode refers to (or extends from) the ISAKMP identity protection exchange and the IKE aggressive mode extends the ISAKMP aggressive exchange. Both modes are in IKE Phase I. In IKE Phase II, a quick mode has been defined. Each mode is used with different authentication methods; for example, pre-

shared key, public key signature, public key encryption, and revised public key encryption are authentication methods for IKE Phase I protocols.

2.4.5 SA Proposal

An *SA proposal* is a list of SAs or security meta parameters that are considered by the initiator to be acceptable to protect the communication between two parties.

An SA proposal is defined in a layer-based structure; each lower layer provides more fine-grained type information for the proposal. The top layer is the *security protocol.* For ISAKMP SA, the protocol is ISAKMP, for the Protocol SA or Phase II SA, the top layer value is the name of the security protocol in Phase II. The middle layer is a list of potential *transforms* for the security protocol defined in the top layer. In phase I, the transform can be IKE, the key exchange protocol used in ISAKMP negotiation. The bottom layer is a list of potential transform *attributes* for each of the transforms in the middle layer. In phase I, an example of transform attribute is SA (key) lifetime.

2.4.6 Security Protocol

In ISAKMP, security protocol refers to a specific security scheme to protect network traffic. Examples of security protocol include IPSEC ESP, IPSEC AH, even TLS. It usually implies a specific security service such as authentication (e.g., AH), confidentiality (e.g., ESP) or combination of them (e.g., ESP, TLS).

2.4.7 Situation

In ISAKMP, *situation* refers to the set of information that will be used to determine the required security services. Situation in ISAKMP can also be set to help the responder pick a proper SA proposal based on the situation.

2.4.8 Anti-attack Features

The first anti-attack feature to discuss is ISAKMP anti-clogging cookies. The cookies are used to prevent passive attacks that can cause the responder to waste resources on useless negotiations. Cookies (initiator cookie and responder cookie) are also used to identify *SA*s because of their uniqueness.

The second feature is the authentication methods used by the IKE mode, which can defeat man-in-the-middle attacks. The third feature is the Diffie-Hellman key agreement employed by IKE, which has a high degree of cryptographic strength.

ISAKMP/IKE is quite complex and the above only covers the basic constructs in the framework and protocols. For more information, please refer to the specifications [34] [25] [44].

2.4.9 Problems with IKE Version 1

While ISAKMP/IKE provides extensibility and flexibility to applications, it is alleged to be over-complex by a number of publications such as [42], [56] and [51]. It is therefore difficult to understand and hard to implement [24]. After some formal security analysis, [36] points out the ambiguities of the specifications that lead to potentially insecure and incorrect implementation. It is argued that Phase II is not necessary, assuming that a connection over the Internet seldom lasts so long that at least two quick modes would be conducted in one main mode session [56]. Phase II is not significantly cheaper than Phase I if Perfect Forward Secrecy (PFS) is required [43]. Furthermore, each authentication method introduces a new set of messages, which makes the implementation even more complex.

On the other hand, IKE does not provide sufficient ID protection. It is well known that the aggressive mode does not consider ID protection at all (even though the public key encryption variant does provide it). Even in the main mode, it has been documented that there is real ID protection provided for the pre-shared key variant, since it forces the IDs to be IP addresses. [43] notes that IKE should have provided ID protection for the Initiator, instead of the Responder. Finally, IKE v1 does not provide any DoS protection, even though the preceding Photuris offers protection from simple flooding DoS [26].

2.5 IKE Version 2

RFC 4306 [27], or IKE v2, is designed to solve the majority of the aforementioned problems. It is based on the same design philosophy as IKE v1, so the underlying cryptographic format is similar to that of IKE v1, such as using HMAC to prevent ID misbinding, a digital signature to prevent man-in-the-middle attacks, and generating different key sets for each direction. However, several major revisions have been made in order to simplify the protocol definition and improve the efficiency.

First, the number of message rounds is reduced to two, by combining the messages for SA negotiation and key exchange. Second, the protocol is further simplified and hence more efficient by allowing negotiation of child SA (equivalence of Phase II SA in IKE v1) in IKE_AUTH message exchange. Third, the authentication is decoupled from the protocol definition, that is to say, the usage of a specific authentication method (e.g., pre-shared key, public key signature, public key encryption, etc.) would not affect the structure of the protocol messages and the initiator/responder's behaviours. Other improvements include (a) the support of extensible authentication by using EAP (Extensible Authentication Protocol) payload; (b) greater security constraint on primitives such as requiring nonce to be at least 128 bits in size; and (c) the output of PRF (Pseudo Random Function) must be greater than 128 bits, etc.

However, IKE v2 still has some drawbacks. The designer of IKE v2 assumes that the risk of DoS attacks (which IKE v1 suffers from) can be minimized by careful implementation of the responder. IKE v2 specifications therefore do not give much consideration to the design of the protocol in order to prevent DoS attacks, except for including two additional messages that the applications of IKE v2 can choose from only when a potential DoS attack is detected. This may worsen the situation when the responder is already under DoS attack. Another weakness of IKE v2 has been documented, which is vulnerability to ID probing [27].

2.6 SSL

SSL (Secure Sockets Layer) [20] is a protocol for secure information transmission from a client to a server over the Internet between layer 3 and layer 4. SSL, which sits in the application layer and has four components, namely, (a) *Handshake Protocol* for key exchange and authentication;

(b) *Change Cipher Specification* for one party to indicate to the other that the chosen key will be used; (c) *Alert* to signal error and session closure; and (d) *Application Data Protocol* to transmit and receive encrypted data.

SSL uses Diffie-Hellman in its pre-master key based key exchange mechanism as well; server authentication, client authentication, and anonymous authentication are provided to secure the key exchange. *CipherSuite*, which is similar to SA, is the vehicle to convey the security parameters of the connection about to be established. However, the purpose of SSL is to create a secure connection over which a certain amount of data can be transferred between client and server. It does not explicitly provide an extensible architecture suitable for all applications, so its potential to become an infrastructure of secure communication protocols that apply to most of the applications is limited.

2.7 JFK

Just Fast Keying (JFK) [2], which is based on ISO 9798-3 and SIGMA, provides an alternative at layer 3 that can be used in the IP security architecture. There are two variants in JFK: JFKi and JFKr. The former is appropriate for the client/server scenario, when more ID protection on the Initiator is provided; the latter is appropriate for a peer-peer scenario, when the Responder's ID is also protected. JFK's design goals include simplicity, efficiency, resistance to memory and computation DoS, as well as proper ID protection for communicating parties. JFK is apparently intended to address the major deficiencies of IKE v1, yet follows a different approach from IKE v2.

Like IKE v2, JFK finishes SA negotiation, key exchange and authentication within two round trips in order to simplify the protocol and make it more efficient. What it has done better than IKE v2 is to provide more DoS protection by keeping the computation level low in the first round, and not generating states in the Responder entity in the first two rounds [2]. To make the protocol more efficient while providing reasonable Perfect Forward Secrecy (PFS), several remedies have been added to the design. First, the designer of JFK argues that PFS or absolute Forward Secrecy (FS) is neither feasible (considering the high cost) nor necessary for all scenarios. Instead, the amount of FS should be a configurable engineering parameter. The parameter can be configured through *forward secrecy interval*, which is a period of time when FS is considered unnecessary and is therefore not provided. Since the *forward secrecy interval*

allows the same secret Diffie-Hellman to be used for multiple exchanges in a given period, many expensive modular exponentiation calculations are saved. This rule applies to both the Initiator and the Responder when the D-H value is generated. Second, JFK avoids complex calculations, particularly for the Responder. For example, when the Responder replies to requests, the only cryptographic calculation involved is HMAC, which is inexpensive.

JFK becomes efficient by removing negotiation in general. As a result, it loses the extensibility which is available in IKE v2, so it does not have the potential of becoming an infrastructure for security protocol. JFK does not support multiple authentication options either; instead, JFK relies on the authentication being done by outboard protocols. Finally, JFK provides two different protocols, just because it wants to handle two different ID protection requirements from two scenarios. One is client/server and the other is peer-to-peer. This could have been avoided.

2.8 Other Key Exchange/Authentication Procotols

Besides ISAKMP/IKE, SSL and JFK, many other protocols have been proposed to solve key establishment and maintenance issues in different applications or environment settings. An overall survey is given in [18] on application layer key agreement protocols, including the protocols that involve two or more parties among which secure communication channels are required for information exchanges over insecure networks. According to the underlying cryptosystems used, the protocols are categorized as certificated-based and ID-based. The former assumes the Diffie-Hellman key pair is employed and each party knows the public key of the other, so the the public key is authenticated with a certificate issued by CA (Certificate Authority). The latter assumes a user ID can function as a public key in an environment where there is a PKG (Private Key Generator)/KGC (Key Generation Center) that is trusted by all parties. The authentication apparently replies on the trustworthiness of the PKG/KGC.

In contrast, a protocol can be categorized by the number of parties involved in the secure communication, because the interaction between multiple parties in a secure information exchange would be different from the interaction between two parties [10]. Furthermore, a protocol for multiple parties with dynamic peers is more complex than a multiple-party protocol since it requires that the security of the communicating group is not comprised when a peer steps out and/or gets into the communicating group [18]. A survey on security requirements and models for group key exchange has recently been carried out [33].

A number of security protocols have been proposed. Two party protocols include MTI, MQV, Jeong (a variant of Diffie-Hellman), Diffie-Hellman, and some ID-based protocols such as Chen & Kudla, McCullagh & Barreto. Three-party protocol examples would be Joux, Zhang, Liu & Kim (ID-based). Several multi-party protocols have also been developed [18]. A typical example is CLIQUES, a key agreement among peers of a dynamic group [52].

This thesis focuses only on two-party security protocols.

2.9 Other Related Work

2.9.1 Single Sign-on

Single Sign-on (SSO) is proposed and matured in Sun's open standard initiative on identification management [54], which lay down the foundation of an advanced user/client authentication that allows the user to "login once, but sign in everywhere," and is authorized automatically by the resources of multiple security domains. Many variants of SSO exist at different levels of applications, from the Internet (Web-SSO) to the enterprise business environment (enterprise SSO).

As discussed in Section 1.2.1, SSO does not completely solve the unrecognizable subject ID problem.

2.9.2 Other Related XML Security Specifications

XML Digital Signature is the building block for many other XML security technologies such as the aforementioned XACML, Web service security, XKMS and SAML that will be introduced in this section. The XML Digital Signature (DS) provides non-repudiation, authentication and data integrity service for XML-compliant data. XML Digital Signature itself is expressed in XML and hence more XML-aware or XML-friendly than the traditional Digital signature. For example, XML DS is composed from hierarchical XML elements such as <Signature>, which is composed from <SignatureValue> and <SignedInfo>, which is, in turn, composed from <CanonicalizationMethod>, <SignatureMethod>, and so on. XML DS allows part of the XML

document to be signed, providing more flexibility for the applications. For more information, please refer to the XML-Signature specification [62].

XML Encryption is another building block for XML security. It is a standardized way to incorporate encryption service into the XML world. Like XML DS, it is implemented by a set of hierarchical XML elements such as <EncryptedData> that consists of <EncryptionMethod>, <KeyInfo> and <CipherData>. XML Encryption allows partial encryption of XML data; specifically, it can be used to encrypt an XML element, or its content, or arbitrary data that can be in any form. For more information, please refer to the XML Encryption specification [61].

Another XML specification that needs to be mentioned is XKMS, or XML Key Management Specification [60]. It is well known that PKI (Public Key Infrastructure) is designed for key distribution. In order not to expose the complexity of PKI to a web-service application, XKMS has been invented. XKMS, which is actually a web service implemented as request/reponse protocols layered on SOAP, and offers a standardized interface for complicated PKI services underneath. The interface is composed of two services, Key Registration Service Specification (X-KRSS) and Key Information Service Specification (X-KISS). X-KRSS is used to establish and manage public key credentials while X-KISS allows applications to locate and validate public key credentials. With XKMS, a thin client as simple as a mobile client can use a PKI service without knowing much about it.

Security Assertion Markup Language (SAML) [39], as its name implies, is the markup language to standardize assertions that are passed among security domains. Assertion allows one security domain to assert to the other any attributes related to an entity or subject, such as authentication results or authorization decisions on a specific subject, and other attributes of specified objects. Assertions are denoted as statements. For example, an attribute is expressed as an attribute statement or <AttributeStatement>. SAML also defines the request/response protocols that allow a security domain to query assertions from the other. For example, a security domain can use <AttributeQuery> to obtain the requested attributes for a specified subject. SAML plays an important role in the distributed environment that involves multiple security domains. For example, Single Sign On (SSO) is usually implemented with SAML.

2.10 Summary

This chapter included an explanation of the terminology that is used in the thesis. Next, a brief overview was given of a number of works that are related to security access control infrastructure and key establishment protocols, such as XACML, SSO, IKE v1/v2, JFK, SSL, SAML and other XML security-related specifications. The next chapter will discuss the XACML-based architecture aiming at solving the seven problems mentioned in Chapter 1.

Chapter 3 ARCHITECTURE

Many applications of eXtensible Access Control Markup Language (XACML) [37] have been found in security-application solutions, yet few of them succeed in solving authorization problems common in typical business and leisure scenarios that involve mobile users. These scenarios include identification management in a mobile environment, issuing a proper authorization request to a domain where the security model is unknown, locating all the applicable policies for an unknown requester, finding a proper service provider that will not compromise the requester's data confidentiality, integrity and privacy, and the issue of applicability of reputation data (See Section 1.2).

An XACML-based architecture is proposed to tackle the above issues. A Subject ID mapping service is the foundation of the architecture, upon which a Meta Policy Server (MPS) is designed to locate the policies for the requester and provide guidelines for overall security management, while reverse authorization and an optional privacy proxy server are used to guarantee the requester's privacy. In addition, a private reputation attribute authority (AA) handles reputation data applicability problems.

A security handshake protocol for secure communication between the MPS and subject attribute authorities is also an important part of the solution. It is described in Chapter 4.

3.1 Overview

The proposed architecture is built on top of the ABAC system [64], which is derived from the XACML data model [37] (as defined in the standardized XACML and SAML specifications, see Section 2.2.2). As shown in Figure 3-1, the Meta Policy Server (MPS), PDP (Policy Decision Point), and PEP (Policy Enforcement Point) interact with specialized Attribute Authorities (AA) [64] from which attributes required to create access requests and make authorization decisions are retrieved. The Policy Authority is identical to the PAP (Policy Administration Point) in the XACML data model. However, our model extends the standard infrastructure with the following new concepts and system components, which examine the aforementioned issues.

- **Subject ID and External Subject Identity**

  Subject ID is the ID of a requester recognizable by the local domain to which the access request is sent. A mobile requester sends his/her initial request with an external subject identity that may be recognizable by the local domain. If it is not recognizable, the local domain will trigger an autonomous registration process (See Section 3.2) for the requester. Once the registration is complete, the requester will get a subject ID that is recognizable by the local domain. A mapping between the subject ID and the external subject identity is created during the registration.

  The recognizable subject ID will then be used *alone* in the access request, or without subject attributes that are unknown to the requester anyway, because the mobile requester has no knowledge of the access model implemented in the local domain. To process the request, PEP in the local domain forwards the subject ID to a Meta Policy Server (MPS) in the system, which consults with an independent subject attribute authority (subject AA), which is usually external to the local domain, to get the subject attributes required by PDP for the decision rendering. The PEP then uses these subject attributes along with relevant environmental information to form a proper request for the requester, and finally sends it to the PDP, which will in turn get the applicable policies for the access request and make the decision.

  Because of the subject ID mapping service, autonomous registration and MPS in the local domain system, the mobile requester does not need to have a recognizable subject identity or know the access control model implemented by the local domain in order to form a valid request. Problem 1 (Section 1.2.1) is partially addressed by using a recognizable subject ID. Please refer to Section 3.3 and 3.5.5 for further discussion.

- **Meta Policy Server (MPS)**

  Meta policies (MPs), which are optionally written in XACML, are organization-wide IT security regulations, guidelines, and rules from security Service Level Agreements. MPs may also include pointers to regional laws, as well as to vocabulary (dictionary) and subject ID mapping (conversion) services. MPS addresses Problems 1, 2 and 3.

- **Reverse Authorization**

As discussed in Section 1.2.4, a mere comparison between security parameters (the local domain) and preferences (the mobile requester) may result in an inability to find appropriate services because of incompatibility between security parameters and preferences. A negotiation procedure that involves a reverse authorization triggered by the requester is proposed to address Problem 4. This negotiation procedure is in fact a reverse authorization procedure in which the local domain's access request, which never exists, is evaluated against the policies of the security domain from where the requester comes.

- **Privacy Proxy Server**

  As discussed in Section 1.2.5, it is impossible for an individual requester to assure his/her privacy not be compromised. A privacy proxy server is an agent where the privacy information and privacy preferences are maintained and the usage of privacy information is tracked. Considering one subject AA is de facto a centric data source for subject attributes, it makes sense to have subject AA, or a parent of subject AAs that sits at a higher level than the children subject AAs in a hierarchically organized group of subject AAs (just as DNS servers work together), and is able to redirect any request to its child subject AA, to perform as a privacy proxy server for service requesters. Please refer to section 3.6 for further discussion.

- **Private (Local) Reputation AA Versus Public (3rd party) Reputation AA**

  The reputation server is extended in the model by introducing the concept of *Private Reputation AA* and *Public Reputation AA* to overcome Problem 6, stated in Section 1.2.6.

- **Communication protocol between MPS & Subject AA**

  Driven by the system requirement that an MPS has to securely exchange info with a subject AA that is unknown to the MPS, a key exchange infrastructure and protocol called KEAML/KEAML-KE, which can be adopted for any application level peer-to-peer secure communications, is proposed as part of this model.

- **Authorization rendering procedure**

  The authorization rendering procedure in Figure 3-1 can be roughly divided into several stages.

A. Registration of Unknown Requester (Steps 0, 0a) to create a subject ID and the mapping between the subject ID and the external subject identity.

B. Discover desired services. Resource IDs of services are returned (Steps 1-2).

C. Use the resource IDs of services to perform reverse authorization (Step 3).

D. The mobile requester sends an access request that contains only the subject ID; this triggers the whole authorization request/response sub-procedure (Steps 4-12).

E. Reputation data collection (Steps 13-14).

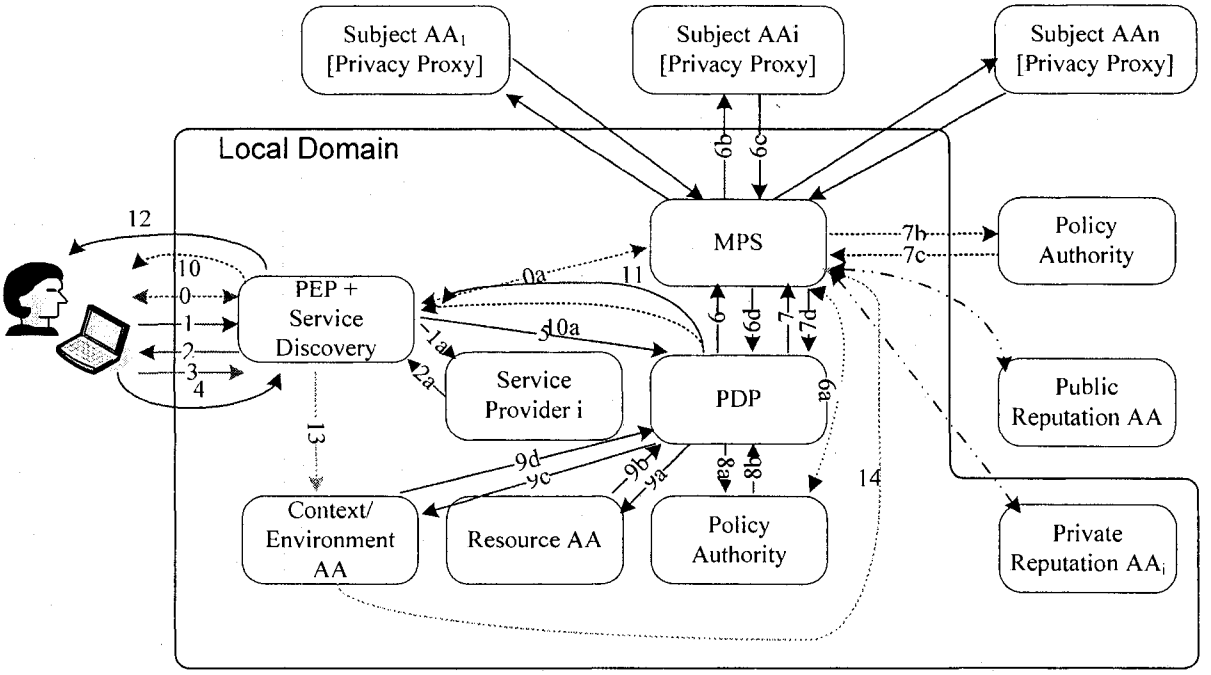

Figure 3-1 Architecture

3.2 Autonomous Registration

Bob, or the mobile requester, can use any subject identity for his/her access request. The subject identity can vary from traditional IDs such as a Hilton Gold Membership Card number or a driver's license number to electronic IDs such as the employee's digital certificate issued by New

Age Business Inc. (which Bob works for), or some globally acceptable digital ID certificate that happens to be recognizable by both the home domain and the local domain. If the subject identity is recognizable by the local domain, the requester will be authenticated before the authorization rendering procedure starts.

However, as mentioned in Section 3.1, an *autonomous registration* procedure is triggered if the external subject identity of the mobile requester is not recognizable by the local domain. The local domain first sends an autonomous registration request to the mobile requester, asking the requester to fill out the information in which the local domain is interested, which is governed by the *meta policies* of the local domain. Some fields, such as subject name and the subject AA location (where valid subject attributes can be retrieved) are mandatory.

Which external subject identity is acceptable, and which fields (except subject name and subject AA location) are mandatory in the autonomous registration request are all defined in corresponding *meta policies*. Furthermore, meta policies can be defined so that traditional manual registration is also permitted. For example, the mobile user can use his/her driver's license and credit card to register in a hotel.

Once the autonomous registration is successful, the local domain issues a subject ID, which is associated with a certain expiration period, to the mobile requester, and a mapping between the subject ID and the external subject identity is created. The mobile requester can then be authenticated and entitled to send an authorization request. However, once the subject ID has expired, the mapping becomes invalid as well, meaning the requester has to re-register before s/he can send another authorization request. Imposing an expiration period on the mapping and subject ID is desirable under some circumstances. For example, a mobile requester should not be allowed to use any services after checking out from a hotel, which is the local domain. Nevertheless, an implementation of the system can choose to deactivate the mapping temporarily instead of cancel it, so next time when the same mobile requester checks in, the mapping can simply be re-activated instead of having to re-enter all the information.

There are two reasons to store the external subject identity of the request and create a mapping to the subject ID issued by the local domain. The first reason is to allow the requester to continue to use an external subject identity to authenticate him/herself if the external identity can be electronically authenticated. It is not recommended, since it will obviously introduce more administrative overhead. The second reason is much more important. The external subject

identity is de facto the subject ID that is recognizable in a subject AA where the subject attributes of the requester can be retrieved. Therefore, it is possible for a mobile requester or principle to have multiple subject identities, each of them associated with separate subject AAs. The meta policies of the local domain can be configured to allow or disallow multiple subject identities mapping to one subject ID, based on specific security considerations.

3.3 Reverse Authorization

Reverse authorization refers to the negotiation, which is actually a requester-triggered reverse authorization procedure that allows the requester's domain (which owns the data) to decide whether to expose the data to the other party or the service provider. Since the negotiation involves a policy-driven decision rendering process, it is much more flexible than a pure security parameters/references matching scheme, and provides a higher level of security.

The decision whether to conduct a reverse authorization is configurable through the meta policies of the requester's domain (home domain). Assuming that a service discovery, which returns the requester a set of resource IDs of services, is performed before the reverse authorization, the reverse authorization procedure can be described as follows:

i) The requester takes the resource ID as the subject identity of service provider.

ii) If the subject identity is recognizable by the requester, then it automatically becomes the subject ID of the service provider in the requester's domain; otherwise, an autonomous registration is conducted to get the subject ID and the mapping between the subject ID and the subject identity (Section 3.2).

iii) Based on the meta policies in the requester's domain, the *action* that the service provider will conduct on the about-to-send data is determined. The action can be determined either by referring to the security SLA between the home domain and local domain, or the value predetermined by the home domain.

iv) The PEP in the requester's domain sends the request to the PDP.

v) The authorization decision is rendered (Refer to Steps 4-9 in Figure 3-1). The decision does not have to be sent to the service provider.

vi) If the rendered result is "Permit," the data can be sent to the service provider for the services, otherwise the data cannot be sent and the reverse authorization fails.

An example is given to explain the details of each step, as follows. Assuming Bob wants to print a corporate confidential document in the hotel. The reverse authorization procedure would be:

1) The reverse authorization is triggered by the MPS (meta policy server) before Bob sends out the confidential document for printing.

2) The home domain takes the printing service's resource ID as the subject identity. If the home domain does not understand the subject identity, it will initiate an autonomous registration procedure for the printing service provider. If the printing service provider does not support the autonomous registration or does not want to register to the home domain where Bob is from, for security reasons, the reverse authorization will fail and stop.

3) Once the home domain gets the subject ID of the printing service, the home domain determines the action that the printing service will conduct on the confidential document based on the meta data configuration. For example, the action can be "read" for printing a document.

4) PEP in the system forms the authorization request with the subject ID (the printing service), action ID ("copy"), and the resource ID (the document), and forwards it to the PDP.

5) With the subject ID of the printing service, PDP gets the printing service's attributes (subject attributes) from an external subject AA. Similar to the authorization process stated in Section 3.4, a session ID is sent to context AA to collect the context attributes. The printing service's attributes could be anything that the home domain is interested in (this is also configurable through meta policy configuration), such as if the printer is under monitoring, the brand name of the printer, and the connectivity scheme of the printer (e.g., wireless, wired, LAN, remote connection over VPN, etc.).

6) With subject ID, resource ID, and action ID and their attributes, the PDP is able to find the most applicable policies from policy authorities in the home domain, and then make the decision.

7) Bob does not have to send the reverse authorization decision back because the printing service is not usually concerned about the decision. If it failed, Bob would simply stop sending the confidential document to the printing service.

3.4 Authorization Rendering Procedure

This authorization rendering process (Figure 3-1) differs from the traditional XACML authorization process in the following aspects:

1) An authorization request does not include subject attributes and resource attributes. Instead, subject ID, resource ID, and action ID are required.

   When PEP forms the authorization request for the requester, a session ID is created and associated with the time when the request is formed. The session is used to retrieve the proper context/environment attributes for the request. The session ID lives until the authorization procedure and the requested operation is finished.

   The basic action ID in the proposed system includes but is not limited to *create*, *delete*, *read*, *write*, *append*, *execute*, *list* and *search*. This list covers the common operations in many access-control models. For example, sending a printing request can be translated as "append the document x to the printer y."

2) PDP and MPS will fill out the subject attributes and other attributes as required. This enables a mobile requester to send an authorization request without knowing the access model that the target domain is using.

For a typical authorization procedure, from the beginning to the end, the following steps are followed. They apply to all three scenarios described in Section 1.1:

Stage A. Registration of Unknown Requester.

Step 0 – 0a. If the requester's subject identity is unknown, an autonomous registration request is sent to the mobile requester, asking to provide the values of attributes. The attributes of interest is configurable and set in the *meta policies* of the local domain. After the autonomous registration is completed, a subject ID (with an expiration period) is issued to the mobile requester and a

mapping between the subject ID and the external subject identity is created by MPS (Meta Policy Server)

Stage B. Discover desired services.

Step 1. The mobile requester searches for desired services based on the home domain's service discovery criteria, which do not include any security preferences of the mobile requester.

Step 2. A list of service providers or resources that meet the criteria is returned. Also, for each service/resource, a service ID or resource ID is provided.

Stage C. Use the resource IDs of services to perform reverse authorization. This stage is configurable by MPS.

Step 3. For each service provider, the mobile requester conducts a reverse authorization (Please see Section 3.3) and chooses the first one that passes the reverse authorization.

Stage D. Authorization request/response sub-procedure

Step 4. The mobile requester sends the authorization request to the service provider, and the request is forwarded to the PEP module.

Step 5. The PEP receives the authorization request, generates and inserts the session ID into the request, and forwards it to a PDP.

Step 6. Neither the PDP nor the local policy authority knows the subject attributes required, considering that a decision must be rendered based on the combination of the applicable local policies, the company guidelines and the relevant security SLA. Therefore, the PDP simply embeds the subject ID/resource ID/Action ID into the subject attribute request and sends it over to the Meta Policy Server (Section 3.5).

Step 6a. If a list of subject attribute IDs has not been calculated for the combination of subject ID/resource ID/Action ID, the MPS will consult with the security SLA, or company security guidelines to get subject attribute IDs – whichever is applicable – and send a request to the local policy authority to get the subject attribute IDs required by the local policies. These subject attribute IDs can then be cached in the MPS. Once they are cached, this step will be skipped for the next subject attributes query.

Step 6b – 6d. The Meta Policy Server sends the request to one or more appropriate subject AAs, gets the subject attributes, combines them if necessary, and sends them to the requesting PDP (Section 3.5, 3.6 and Chapter 4).

Step 7 – 7d. The PDP always sends a request to the MPS to get the external policies such as the regional laws for the requester (Step 7), or the high-level policy, such as security SLAs. If the MPS determines that it does not need any such policies, it simply returns nothing; otherwise, it sends a request to an external policy authority for the external policies whenever it is applicable, then bundles these policies and sends them to the requesting PDP.

Step 6b – 6d. The PDP finds out that the external policies, such as regional laws on movie ranking, which require some subject attributes that have not been retrieved yet, such as the age of the subject. The PDP then initiates another subject attribute request, but this time with subject ID and subject attributes that are missing. Similarly, the MPS obtains the subject attributes and forwards them to the PDP.

Step 8a – 8b. The PDP sends a request to the local policy authority to get the policies.

Step 9a – 9d. The PDP collects the resource attributes from local resource AA; it uses the session ID to get context attributes as well from the context/environment AA.

Step 10 – 10a. If applicable, the PDP asks extra information that could not be obtained from AAs other than from the original requester him/herself (Section 3.9).

Step 11. Now that the PDP gets all applicable policies and all attributes, it renders the decision and sends the result to the PEP.

Step 12. The PEP sends the result to the mobile request. However, the PEP can choose not to grant the access if the decision is made conditional on specific obligations, assuming that the local context AA can track where the obligations are complete or not. To find out if the obligations are complete, the PEP has to register on the local context AA, and listen to the occurrences of the obligations. If the obligations are required but not a prerequisite for the decision, the access can be granted immediately. Whether or not the requester has completed the obligation after the access, it would be collected anyway for the next decision rendering. Please note some obligations are not trackable by nature. For example, "user must send an email to entity x," the local context cannot know if the user has sent the email to entity x, if entity x is not

part of the domain or is not linked to the domain in any way. How to make an obligation result collectable is out of the scope of this thesis. We recommend that the obligations incorporated into policies should be made such that their results are collectable with the facilities implemented in the domain.

Stage F. Reputation and obligation data collection

Step 13. Reputation and obligation data is collected from the PEP.

Step 14. Context/Environment AA periodically sends the reputation data to MPS, which forwards the information to internal private reputation AA or sends it to external public reputation AA.

3.5 Meta Policy Server (MPS)

Some functions of MPS have been mentioned in the preceding sections. In general, as the core of the architecture, MPS provides two categories of service:

- **Category 1** – The management of the interactions between the security domain and outside security entities

  It includes the services that conduct or enable security information exchanges with the outer entities such as subject AAs, external policy authorities, public reputation servers and even another security domain that supports MPS or similar.

- **Category 2** – The management of the security domain itself

  It includes the services that configure the security domain to meet the general security requirements from the service providers and resources in the domain, while offering common facilities as the foundation to support the general security requirements

In this section, we first describe the basic services MPS provides, and then discuss the components of MPS to support these services.

3.5.1 Category 1 Services

- The service to establish the secured communication channel between the security domain and external security entities.

  In order to enable secured information exchanges between the security domain and the other security entities (which may adopt a variety of communication protocols), a standardized protocol or infrastructure is required to establish a secure channel upon which these communication protocols can be bounded. Please see Chapter 4 for details.

- The service to look up the external subject identities and corresponding subject AAs, given a subject ID that is recognizable by the security domain. It may also define the acceptable types of external subject identities.

- The service to locate or provide external national/regional regulations/laws, corporate security guidelines, and security SLA (Service Level Agreement) between security domains or entities.

- The dictionary service that converts the local vocabulary or terms to their equivalences that are recognizable by external security domains or entities, such as subject AAs, policy authorities and public reputation servers.

  An example of utilizing dictionary service is at *Stage D*, prior to sending a subject attribute request to subject AAs, and after receiving the response. Before sending the request out, the MPS replace the terms (such as subject attribute ID) with the ones recognizable by the targeted subject AAs; after receiving the response from the subject AA, it replaces the subject attribute IDs back with the local subject attribute IDs and sends the result to the PDP that originated the subject attribute retrieval request.

3.5.2 Category 2 Services

- The service to allow the domain administrator to configure the security domain.

  The configurations include but are not limited to:

  i) Whether or not to support private reputation AA and if yes, what reputation data to forward to the private reputation AA; similarly, whether or not to support public reputation AAs and what reputation data to forward to external public reputation servers if they support it.

  ii) Whether to check if a registered subject AA is still trustable or not, and when and how frequently the check should be done.

  iii) Which security algorithms such as encryption/decryption, digital signature algorithms are supported; furthermore, which are the minimum security

algorithm requirements on the other security domains or entities with which information exchanges are to be conducted.

iv) Whether or not to support reverse authorization and under what conditions or scenarios to trigger the reverse authorization. The reverse authorization is usually triggered when the going-out data is confidential and/or the service provider is trustable.

v) Whether or not a traditional manual registration is permitted. An autonomous registration is triggered by the security domain if a received subject identity is not recognizable. If a traditional manual registration is permitted, then the registration can happen before the requester sends his/her first authorization request, or after the user chooses to register manually. Since the location of subject AA is not usually retrievable from a manual registration, the security domain may choose to find a most appropriate known subject AA for the requester and manually create the mapping entry in the table.

vi) Which authentication algorithms are supported; the weakest authentication algorithms that are acceptable under certain circumstances are defined. For example, a meta policy may define that password-based authentication as unacceptable.

vii) The service to assign access privileges to a newly registered requester. The privilege assignment is dependent on the type of the access model of the domain; however, the minimum access right assigned to a registered user is to perform a service discovery, as described in the system assumptions.

viii) Whether or not to allow multiple subject identities mapping to one subject ID issued by the security domain. If multiple subject identities per requester are allowed when registering, the requester can provide several subject identities that will be mapped to one subject ID. By providing multiple subject identities, the requester may obtain more access rights than those who only provide one, because multiple subject identities imply the requester has access to more than one category of data in his/her home domain. However, a security domain may choose to disallow multiple subject identities due to security considerations.

- The autonomous registration service.

  The autonomous registration service has the following tasks: (a) generate subject ID for a newly registered principle; (b) assign appropriate expiration period with the subject ID based on relevant policies; and (c) determine the mandatory service attribute that the principle must provide.

- The service to calculate the actions that a service provider will conduct on the about-to-send-out confidential content.

  The simplest way to calculate the actions is to look up in the table where the service provider's action (e.g., printing) is statically mapped to the action that the security domain thinks will be imposed on the confidential information (e.g., read and copy). However, an MPS can also adopt the trustability of the service provider in the calculation. In an extreme example, the MPS may map an action to dummy, i.e., the service provider will never compromise the confidentiality of the data under any conditions.

- The service to combine subject attribute retrieved from subject AAs.

  The combination algorithm for subject attributes retrieved from multiple subject AAs is determined by relevant meta policies. For example, several subject AAs return different credit history records for the same subject. The service, based on the meta policies, takes the worst record as the credit history of the subject.

- The service to create, track an authorization request/response session.

  As mentioned, a session is created for each authorization request. The session is uniquely identified by a session ID, which is created by this service, which also associates corresponding subject information to the session, such as subject ID, the network connectivity used to send the request, obligations for the authorization and the request itself.

  A session ID expires only if the session is finished, which will not complete until the associated obligations are clear.

- The service to determine subject attribute IDs from given subject ID/resource ID/action ID.

  Based on the subject ID/resource ID/action ID, the service consults with relevant security SLA, and corporate security guidelines. Local policies, if applicable, get the subject attribute IDs.

The MPS services are based on the other three components, namely, meta policy repository, dictionaries and mapping tables, as depicted in Figure 3-2.

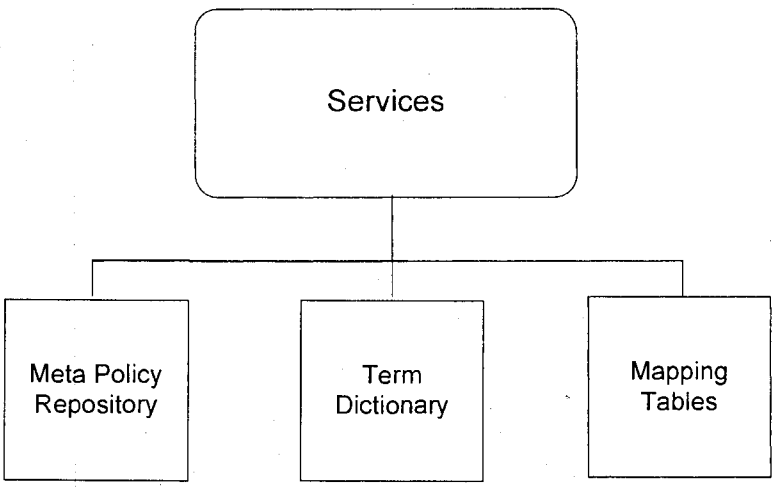

Figure 3-2 MPS Compnents

3.5.3 Meta Policy Repository

Meta Policy Repository is where all meta policies are stored. The meta policy server relies on meta policies to perform its functionality. A meta policy can be written in XACML, but it can also be configured on data that are stored in a database or flat files. There are several categories of meta policies:

- Security domain configuration

  The configuration policies cover all the topics that are related to the configuration of the security domain, such as which external subject identity is acceptable (e.g., traditional ID

systems such as a driver's license, membership #, digital certificate, etc.), which subject attributes are required for a registration, etc.

- Organization-wide IT security regulations, guidelines
- Federal/national/regional laws and regulations

  They can be pointers to the federal/national/regional laws and regulations that are held by external policy authorities (Figure 3-1).

- Security service level agreements

3.5.4 Term Dictionary

The dictionary term of the meta policy server is based on the security service agreement/contract between the local domain and other domains (e.g., home domain of Bob, other third-party attribute authorities, etc.).

A dictionary might be built to translate glossaries of different security systems that are used by other domains and/or authority servers. This may be based on common vocabulary that is set up as an industry standard.

In such a case, the dictionary may just be a dictionary service agent that would simply forward the request to an external standard vocabulary server or similar service.

3.5.5 Subject ID Mapping Tables

A subject AA is added to the table as a result of either autonomous registration of s subject or the setup of security SLA (Service Level Agrement) between MPS and subject AA. Optionally, public key exchange is accomplished before the subject AA is added.

The mapping table keeps:

- Mapping information from subject ID to subject AA's location (URL).

- Mapping information from subject ID to one or more external subject identities that corresponding subject AAs recognize.

3.6 Subject AA and Privacy Proxy Server

Subject AA is a trustable agent or authority from which the attributes of a specified subject can be retrieved. Even though a subject AA can also check the trustability of an agent that asks for the subject attributes, it does not stop it from distributing a subject's private information to some parties which may violate the privacy laws. This issue is stated as Problem 5 (Section 1.2.5). The concept of *privacy proxy server* is introduced to solve these problems.

In general, a privacy proxy server should be agent or authority which:

- Is trustable;
- Holds the private information for subjects; and
- Provides the services or facilities to comply with the five principles of privacy laws [19].

Since a subject AA already owns subject attributes, it makes sense to extend it to provide privacy proxy's functionality as well. The functionality of a privacy proxy server includes:

i) To maintain the user's private information and privacy preferences per type of services, and deliver specific privacy information to service providers based on the privacy preferences of those to whom the privacy info was disclosed and notify the user how and what info was disclosed.

ii) To deliver the privacy information only after the receiver has been authorized; to track the usage of delivered privacy information; the privacy delivery should not cause the compromise of the subject's real identity, or leakage of the information that can be used to infer the subject's real identity.

iii) To inform the subject or principle of how the usage data of privacy information will be reported, e.g., the privacy proxy can deliver electronic reports on the privacy usage.

iv) To answer specific questions from the subject/principle regarding the privacy information disclosure.

3.7 Private Reputation AA versus Public Reputation AA

To overcome the problem stated in Section 1.2.6, a *private reputation* AA is introduced into the infrastructure, as shown in Figure 3-1.

Both private reputation AAs and public reputation AAs maintain reputation data, but a private reputation AA only collects and stores the reputation data of a service consumer or a requester. A public reputation AA collects the reputation data from both parties. The reputation data collected by a private reputation AA is biased towards rather than against the interests of the service provider, since the purpose of it is, after all, to provide a more precise or application-specific way to evaluate how well or badly a requester has behaved when consuming a specific type of service.

3.8 Context/Environment AA

A context/environment AA keeps environment attributes such as time, date, temperature, and even the brightness level of the environment; but more importantly, it keeps the following context attributes:

- Connectivity scheme & status

  For example, if a connectivity scheme matters in policy selection and evaluation, then the PDP should query the context/environment AA to get the connectivity scheme of a device by given session ID that is associated with the device.

In contrast, a context AA can be used to monitor and record the obligation activities and index them by the session ID. If a requester fails to conduct the mandatory actions or obligations, this bad record will be noted in the private (local) reputation server under the requester, and it is cleared only when a penalty is paid by the requester next time when s/he tries to ask for another service. There are two situations where a context AA can be used to collect the obligation status:

- Completion status of obligation action that should be conducted *before* the authorization can be fully granted to the mobile requester; and

- Completion status of obligation action that must be conducted by the requester *after* the authorization is granted.

3.9 Request More Information from the Mobile Requester

Some information may not be obtainable from an attribute authority in the system. For example, some information such as the brand name of the device that initiates the request is not available from an attribute authority due to technology difficulties; or an attribute authority cannot expose requested information to others based on the privacy agreement between the attribute authority and the requester.

In order to get more information from the mobile requester, the PDP involved in the authorization session first initiates and sends the request to PEP, which then forwards a need-more-info request to the mobile requester. For the mobile requester, a possible implementation would be that a visual prompt is presented, asking if s/he is willing to expose the information; if not, the PDP can choose not to provide the service based on the policy.

3.10 System Validation

Although the whole system has originated from XACML and ABAC, several new concepts or logic components such as MDS, reverse authorization and subject ID mapping are incorporated in the system. Some improvements have been introduced into the authorization request/response procedure in order to address the problems presented in Section 1.2. Because of the addition of the new logic components and the systematic improvements, the infrastructure becomes quite different from its origin, even though such old concepts as PDP, PEP, PAP are still adopted. Therefore, it is necessary to have a brief discussion about the validity of the system in terms of how secure and feasible the system is, and how it can be validated.

In the following section, we will discuss the validity of the system from three viewpoints – security, feasibility and overhead. Please note that because all the statements regarding the system

in this thesis are still logical and high-level, it is impossible to achieve a complete analysis for the system without further design and implementation details. As mentioned, the complete analysis and validation can only be done after the whole system has been implemented.

3.10.1 Security Analysis

It can be argued that introducing MPS into the system may weaken the security of the infrastructure. The reason is that not only is it the gateway between the security domain and external systems (Category 1 services), but the security manager of the security domain itself (Category 2 services); a one-point failure in MPS may cause the whole system to crash.

It has been debated for decades whether a distributed system is better than a centric controlled system in almost all domains of computer science, and the debate continues. We believe that whether the system is centric or distributed is really a choice of design or implementation rather than of system level. For example, logically, a system has only one database but the database can be implemented either as distributed or as centric, depending on the concrete requirements of the applications.

In this infrastructure, the MPS or Meta Policy Server is just a system logic component; that is to say, it can be implemented as either a centric controller or a set of distributed functional components that talk to each other in the security domain. When implemented as a centric controller of the infrastructure, the organizational-wide policies, IT security regulations, security SLAs, pointers to regional regulations, and laws are persisted in one centric store, where the category 1 and 2 services are also deployed. The advantage of this implementation is simple and the administration overhead is low. As a result, the security of MPS is easy to handle. It does not scale up for a large system, where a distributed system is more appropriate, the administration overhead is high and the security of MPS is hard to manage.

The security of MPS itself is critical to the security of the whole system, considering MPS owns all the high-level policies and meta policies that govern the security domain, and behaves as the information gateway to external systems. Any compromises to the policies and meta policies will damage the security domain. This includes the compromises to the repositories where they are stored, and the disruption or impairness of services.. Therefore, it is important to assure high quality confidentiality and the integrity of services to MPS as well as to the related components.

A strong authentication method is also highly recommended for the administration system that manages MPS. The discussion cannot be continued without low-level design and implementation details, which will involve future research.

As for the concept of reverse authorization, it is not different from the normal authorization procedure except that it is not triggered by a concrete reverse authorization requester. Since it is initiated internally instead of initiated by an outsider, it is more secure than the normal authorization procedure.

3.10.2 Feasibility Analysis

Based on the statements in all previous sections of this chapter, it seems that the system is logically feasible. However, its feasibility should be validated by examining whether the assumptions (on which the system is based) stand.

Two assumptions need to be examined. One is the interoperatibility and the expressibility of XACML, the other is the equivalency of subject ID and resource ID.

XACML is, by design, one of the most interoperable policy languages. Because it is based on XML, XACML can theorically be used to express any hierarchy and structural information, particulary for the information or instructions that involve subjects (who), actions, objects (data to be protected) and conditions. Each of these is structural, hierarchical, and extensible. XACML may have limitations, but they will be offseted by its extensibility; that is to say, one can always expand the capability of XACML by extending it.

The equivalency of subject ID and resource ID is the basis of reverse authorization. If resource ID is not the equivalency of subject ID, it is much harder to make reverse authorization possible. However, resource ID has a similar structure to subject ID in XACML, so it is possible to take resource ID as a potential subject ID and use it as subject ID in reverse authorization.

Like security analysis, the analysis of feasibility cannot be complete until the design and implementation detail has been determined. For example, the performance of the system will vary with different design choices. Another example involves how a security domain knows whether a subject ID is recognizable or not. A potential design choice would be to leverage the <issuer> sub-element of <subject> in XACML. If the issuer is standardized and/or recognizable, then the

subjectID would be recognizable. The system design choice and its feasibility analysis should be deferred until the design detail is determined.

3.10.3 Overhead Analysis

Compared with XACML or ABAC framework, sevaral extra logic procedures are introduced in the proposal in order to address the problems discussed in Section 1.2. In the following paragraphs, we will conduct the overhead analysis on the major extra processes, namely, autonomous registration, reverse authorization and term conversion/subject ID mapping.

Although autonomous registration is conducted only when a subject identity is unrecognizable by the local domain, it can be argued that the mobile requester may not want to go through the extra registration procedure in order to get the service. However, registration is a common, if not the only, technique to collect information to establish a valid user account, therefore, autonomous registration procedure would be a quite acceptable approach to establish the mapping between external subject identity and internal subject ID. Besides, autonomous registration is performed automatically and transparently to the user; the delay would not be noticeable because the amount of information required is small (e.g. only the location of the subject AA and subject name are mandatory in most cases). Furthermore, an implementation of autonomous registration can choose to cache or keep the mapping for a user for a certain period of time, so next time the same user would not have to go through the registration but simply activate the mapping. Finally, whether or not to support autonomous registration is determined by meta policies; yet not supporting autonomous registration would either force the user to go through traditional manul registration, or compromise the capability of the system in providing services to more customers.

Reverse authorization is introduced to allow the home domain to make a proper decision on whether or not to send out confidential data based on the applicable policies. As mentioned in Section 1.2.4, the simple matching approach, which has less overhead, may result in inability to find appropriate services. However, the home domain can determine not to conduct reverse authorization, or conduct reverse authorization with simplified policy sets, if the performance becomes an issue that has higher priority than data confidentiality. Nevertheless, the overhead of reverse authorization should be measured through the implementation later.

It is quite obvious that an extra term conversion and subject ID mapping procedure is necessary to make proper interactions between different security domains possible. Probably, the performance overhead introduced by this procedure would be low since the major computation involved is just table look-ups.

The overhead analysis above is theoritical and cannot be complete without testing with the implementation, which will be part of future work.

3.11 Problems Vs. Components

The infrastructure proposed addresses Problem 1 to Problem 6 discussed in Section 1.2. The table below shows which components of the infrastucture solve which problems.

However, all the components are still conceptual. Although an initial system validation is done (Section 3.10), a full validation cannot be conducted until the implementation of the system is complete.

Components	Problems
Subject ID Mapping	P1
MPS	P1,2,3
Autonomous Registration	P1
Reverse Authorization	P4
Privacy Proxy Server	P5
Private Reputation AA	P6

- P1 – Indeterminacy of AC model/unrecognizable subject ID
- P2 – Locating Applicable Policies
- P3 – Entities constrainted by different law systems
- P4 – Protecting the requester's data
- P5 – Privacy protection
- P6 – Applicability of reputation data

Table 3-1 Problems Vs. Components

3.12 Summary

This chapter proposed the infrastructure in which the problems raised in Chapter 1 could be addressed. This chapter presented an overview of the system and reviewed the new and important logic components of the system such as MPS, reverse authorization, autonous registration, subject ID mapping, and the revised authorization request/response procedure. Finally, a brief validation analysis was presented.

From the next chapter onward, we will focus on the design and the implementation of the protocol that is used to establish a secure communication channel between two parties over unsecure networks.

Chapter 4 KEAML/KEAML-KE

Secure information exchanges across the unsecured Internet between any two enterprise-level applications or services call for a standardized layer 7 key exchange and authentication protocol that can be deployed in a number of different application environments. An example of such an application environment is given in the XACML-based access control architecture discussed in [46]. Consider a mobile user trying to access a resource in a foreign environment. In order to process the access request from the mobile user, the Meta Policy Server (MPS) of the security domain protecting the requested resource may need to acquire the mobile requester's subject attributes from a Subject Attribute Authority (Subject AA), through a secure channel over the Internet. However, the secure channel to convey the subject attributes cannot be established until the two parties agree upon a key with which to protect their communications. Unfortunately, such a protocol is not addressed by any of the major standards efforts in XML security, including specifications from W3C (World Wide Web Consortium) and OASIS.

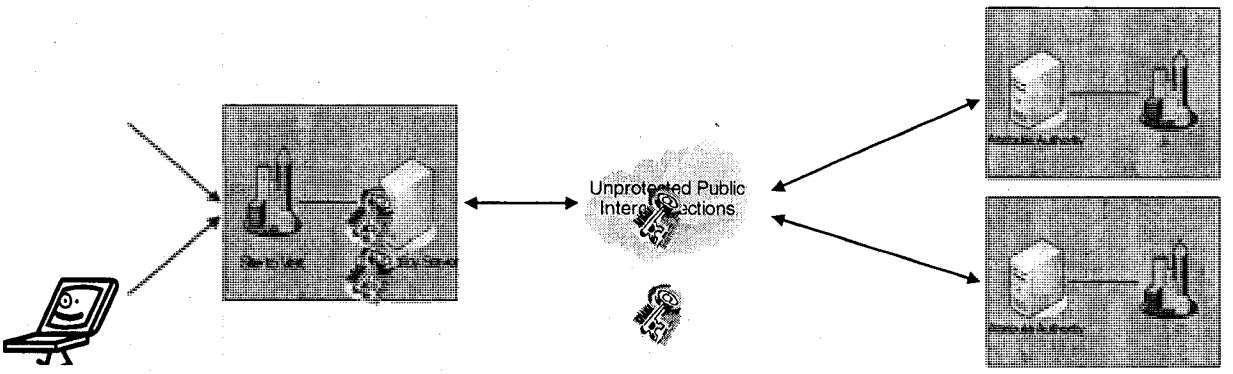

Figure 4-1 Scenario to Use Secure Handshake protocol

This chapter proposes the syntax and semantics for the XML-encoded KEAML/KEAML-KE protocols, which combine the fundamental constructs used by ISAKMP/IKE v1/v2 and JFK from layer 3 to the application layer (or layer 7) to address the aforementioned issues and requirements. Please note this KEAML/KEAML-KE is based on our preliminary work done in [45].

An example of the application of KEAML in a mobile wireless environment is also provided in this chapter.

4.1 Threat Model and Requirements

As discussed by Diffie, et al. [17], a reliable key establishment protocol has to link key exchange with authentication to ensure that the key is shared with the authenticated party rather than an impostor. The protocol should also offer perfect forward secrecy, which means that "disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs" [17]. Furthermore, the protocol should be designed so that:

- Negotiation of security parameters (such as encryption algorithm, key lifetime, and so on) is possible between the two parties;
- High immunity to attacks, particularly man-in-the-middle, replay, etc., should not be effective;
- Scalability is provided (i.e., key exchange can be conducted between any pair of parties that do not share any secrets in advance);
- Message syntax and semantics allow extensibility and flexibility, while guaranteeing the correctness of exchange templates, in addition to leveraging field-level encryption/signing of exchanged data; and
- Finally, the protocol should be standardized, extensible and highly interoperable.

4.1.1 Attacks Overview

A number of attacks on key exchange and authentication protocols have been discussed in many publications. Some are on normal protocols without much security consideration [16], while some are on the protocols (e.g., ISO/IEC 11770-2 and IPSec) that are carefully designed [15] [7]. Attacks on key exchange and authentication protocols are not necessarily attacks on the underlying cryptographic mechanism such as encryption, hash function, MAC, and digital signature. An attack can be successful without breaking the cryptosystems that are used. For example, a successful attack can just be an attack to nonce, which could be as simple as replaying

old messages (i.e., replay attacks), because a system usually has a limited number of memorized nonces that can be used to detect a replay message. Another example would be reflection attacks. A carelessly designed protocol could be vulnerable to a reflection attack, in which messages are simply reflected back at the sender, where there are no encrypted components (e.g., encrypted ID of the intended receiver) in the messages to indicate where the messages intend to go [32]. This is similar to so-called *identity misbinding attacks*, where the victims fail to indicate to the other person with whom they believe they are communicating by including identities of the other in a form that is constructed using the proof of knowledge of a secret, such as digital signature or HMAC. Moreover, some attacks are specifically designed for security protocols. A typical example is the chosen protocol attack mentioned in [29]. The chosen protocol attack is on the interaction between protocols, assuming that multiple protocols can be run parallel by the same principles, and the same keying material is shared between them. For instance, in a chosen protocol attack, a message encrypted with RSA can be sent to a party for signing and to be decrypted; the victim in the example is called RSA signing oracle.

Based on the attack objectives, there are two categories of attacks on security protocols: (a) the attacks on authentication; and (b) the attacks on secret distribution [63]. In an authentication attack, the attacker attempts to persuade the other parties to believe his identity by impersonating the initiator or the responder. An example of an authentication attack is *key compromise impersonation*. When the private key is compromised in an asymmetrically based key agreement protocol, a man-in-the-middle (MITM) can launch such an attack to cause one party to be impersonated in a protocol run [53]. In a secret distribution attack, the secret can be generated once the adversary is believed to be a valid or trusted secret generator such as the initiator, responder or even KGC. The secret can be stolen by the adversary if it impersonates one of the parties or compromises the keying materials successfully, or somehow revives an old secret [63].

Another paper categorizes the attacks into several types from the protocol-design perspective [22], namely: (a) protocol flaws that are vulnerable to reflection attack etc.; (b) key guessing flaws that are vulnerable to dictionary attacks; (c) stale message flaws that fail to offer the refreshing of messages and are vulnerable to replay attacks; (d) parallel session flaws that are vulnerable to oracle session attacks and chosen protocol attacks; and (e) the cryptosystem flaws that are vulnerable to cryptanalysis attacks.

However, the two categorizations (above) do not cover an important type of attack, Denial of Service (DoS). An adversary can launch a DoS attack by flooding the responder with a large volume of false requests to consume the responder's resources (such as bandwidth, memory and CPU) and prevent it from providing services. An adversary can launch such an attack from a number of nodes over the networks that are compromised (Distributed DoS).

4.1.2 Threat Model

The threat model of the security protocol, or KEAML/KEAML-KE, is extracted from the environment setting where adversaries have full powers of launching any attacks discussed in Section 4.1.1. Although the previous section talks only about active attacks, it is understood that passive attacks should also be considered in the threat model. Briefly, the threat model assumes that KEAML/KEAML-KE is deployed in such an environment where the attackers can:

- Eavesdrop whole or part of conversations between peers if necessary. In particular, the eavesdropper understands the pattern of the conversations.
- Listen to the conversation to steal the identities of the peers.
- Collect the information and predict somehow the key or keying materials about to be generated, if the key generation is not sufficiently random.
- Collect almost unlimited numbers of encrypted and non-encrypted messages between the peers, and launch a replay attack if necessary.
- Generate, steal, or revive an old key or keying materials and impersonate the peers and launch a man-in-the-middle attack.
- Generate, steal, or revive an old key or keying materials and decrypt the current and historic conversations. Furthermore, if the key or keying materials are used to generate other session keys, these session keys will also be broken.
- Launch cryptanalytic attacks on the crypto-primitives of the protocols such as digital signature algorithms, HMAC functions, hash algorithms, encryption and decryption algorithms.

- Attempt to launch identity-misbinding attacks if one peer fails to indicate to the other who it thinks it is talking to.

- Pretend to be one of the peers and take an active part in the protocol runs to detect the identities of the peers, regardless of whether the protocol runs fail or not.

- Detect the protocol session flaws and launch the chosen protocols attacks, oracle session attacks etc. on it.

- Launch simple DoS attacks by flooding the peers with a large volume of requests; or launch more complex DoS attacks such as a cookie jar attack, in which the adversary replays the cookies collected to attempt to impersonate the peers.

- Control vulnerable nodes on the networks and use them to launch Distributed DoS attacks on the peers.

The above scenarios cover the majority of the attacks and the design of KEAML/KEAML-KE will be based on them. However, as mentioned in the later session, these scenarios do not enumerate all the cases and the security level of the protocol needs to be verified by employing formal security analysis.

4.1.3 Requirements and Design Guidelines

Secrecy and *correspondence* are believed to be the first two aspects of the security protocol requirements [1]. *Secrecy* is the secrecy of key or keying materials, referring to how badly the key could be comprised. The concept of *Perfect Forward Secrecy* is built on top of *Secrecy*; it requires that the disclosure of long-term key or keying materials must not compromise the secrecy of short-term or session keys that are derived from the long-term one [17]. *Correspondence,* on the other hand, refers to the occurrence and order of protocol messages between the peers. The actions taken by the peers upon receipt of the messages must have a definitive relationship and dependencies so that the matching protocol runs, or matching conversations can be definitive and measured [1].

However, the correspondence does not guarantee that the authentication of key exchange is complete. That is to say, the correspondence property of key-exchange protocol only assures one

peer (Alice) that the protocol runs or conversation conducted with the other peer is matching (i.e., matching protocol runs or matching conversations). It does not assure that the other peer is Bob instead of Carole, or an adversary. Therefore, the third aspect of the requirements is *authentication*, which in turn has two aspects: message authentication and mutual authentication [63]. Message authentication emphasizes the importance of message authentication in key-exchange protocol. Without message authentication, a security protocol is much more vulnerable. Some attacks on the encryption without any message authentication are demonstrated in [7]. Mutual authentication is also important in an authenticated key-exchange protocol. Unidirectional authentication or even anonymous authentication could lead to a situation where an adversary can easily launch an impersonation attack on an unauthenticated peer.

Another important aspect of the requirements, or a high-level design principle to follow, is the *clearness* and *explicitness* of the protocol [1] [4]. First, the message interpretation should only depend on its content instead of context [1], in other words, the property of the messages should be explicitly defined so that the recipient would be able to recover the meaning of a message without any context; that is, any components of the message should not have been inferred from its context. The authentication of the message should depend only on information contained in the message. Furthermore, including a name as identity in the message is essential. First, for example, the message with a secret should show the evidence to the secret receiver who is the initiator and responder [63]. Second, the condition for message to be acted on should be clearly set out (clearness). That is to say, appropriate actions should be defined clearly upon the conditions under which the message is received. Third, the protocol should be clear on what encryption is and how it is used. It should be also clear on how to prove the (re)freshness or timeliness of the message. For example, nonce ensures only temporal succession instead of association. The protocol, therefore, has to be clear that nonce does not imply any association between the messages and the sender. In order to provide the timeliness, the nonce should be unpredictable [1].

A number of tactic-level requirements or principles are also raised to avoid potential design pitfalls [4]. The first design principle is signing before encryption [1] so that the receiver can assume the sender has the knowledge of the data (non-repudiation). Signing before encryption also avoids some attacks to replace the signature over encrypted content [4]. The second design principle is limiting the scope of key [29], or avoiding using the same key for different purpose in a protocol design; otherwise, the protocol will become vulnerable to a number of attacks, including signing oracle attacks mentioned in Section 4.1.1 [4]. The third principle is that the

version and protocol should be associated with a unique identifier in order to avoid attacks such as the chosen-protocol attack. Furthermore, if a key has to be used for a different purpose, the different usage of the same key should have a different identifier.

In summary, secrecy, correspondence, authentication, clearness and explicitness should be the design goals of the protocol, or KEAML/KEAML-KE in this thesis. To guarantee that these goals are satisfied, some tactic-level principles, such as the signing-encryption order, the use of a key, and correct versioning and identification of protocol should be obeyed.

4.2 Comparison of Major Security Protocols

In Chapter 2, some major security protocols, namely, JFK (Just Fast Keying), IKE v1, IKE v2, and SSL, etc., are discussed. KEAML/KEAML-KE design will be derived from some of them by mixing their design ideas and removing the bad properties that have caused drawbacks. Therefore, it is worth giving a comparison between them based on some publications and their specifications [1] [56] [5] [57] [2], in terms of their design, advantages and disadvantages, before we proceed to define the KEAML/KEAML-KE framework and protocols. Table 4-1 depicts the comparison between them. In particular, IKE v2 is much more simplified than IKE v1, yet it is still based on SIGMA as IKE v1 is, in terms of both using HMAC to prevent ID misbinding and a digital signature to provide protection against man-in-the-middle attacks. As for the capability of supporting extensible authentication (XAUTH), it is documented in an Internet draft [5].

The technical comparison between IKE and SSL is based on a technical comparison between IPSec and SSL [1]; some comparisons specific to the Internet Protocol security have been removed.

Finally, KINK, which can be used in IKE v1 and IKE v2, is an inexpensive alternative to Diffie-Hellman that is used to establish/maintain SA using Kerberos authentication. For the details, please refer to [57].

Protocols	Design	Advantages	Disadvantages
JFK	(i) *Forward Secrecy Interval* to improve efficiency while satisfying a certain amount of forward secrecy; (ii) Two protocols to provide ID protection in C/S and peer-to-peer mode; (iii) Small calculation and no states generated in first 2 messages; no new states generated for repeated messages; (iv) Adapts SA but not 2 phases; (v) Unidirectional keying.	(i) DoS protection against flooding request, cookie jar attacks; (ii) Simple without much overhead in negotiation and auth; (iii) Only two round trips; (iv) ID protections against passive attacks; ID protection against active attacks provide to either the Initiator or the Responder.	(i) Two similar but separate protocol to deal with C/S and peer-to-peer scenarios; (ii) Provide ID protection against active attacks to either the Initiator or the Responder; (iii) Not easy to extend (no protocol framework provided); (iv) Removed two phases based on the assumption that specific strong encryption algorithms are used.
IKE v1	(i) 2-phases to establish SAs; (ii) Extensible protocol infrastructure; (iii) 4 Authentications embedded in the protocol design; (iv) No stateless cookie; (v) Main mode to provide better ID protection than quick mode; (vi) Use *DH* or *KINK* for key exchange[57].	(i) Extensible protocol framework; (ii) Separated key exchanges from SA parameter negotiation; (iii) Mutual authentication; (iii) Supported multiple authentication methods; (iv) Standardized D-H groups.	(i) Over complex; (ii) The number of rounds is high iii). Vulnerable to DoS, even no protection from simple flooding attacks; (iv) The ambiguity and complex of the specifications cause interoperability issues between implementations;.(v) ID protection against active attacks provided only to the Responder; (vi) ID protection is flawed.
IKE v2	(i) Simplify the 2-phase design to improve efficiency; (ii) 1 optional round where stateless cookie is used to deal with DoS; (iii) Bidirectional keying; (iv) Authentication becomes independent component to reduce the complexity v). extensible AUTH [5].	(i) All advantages of IKE v1; (ii) Simple and the rounds of messaging reduced to 2; (iii) Simplified two-phase implementation; (iv) More considerations on DoS protection, optional messaging to defeat DoS once it detected; (v) more abstract design of auth and provided extensible authentication.	(i) DoS protection is only optional; the design requires the responder be able to detect DoS attacks in the implementation level; (ii) Some improvements on ID protection but not as good as what JFK offers.
SSL [1]	(i) At the application layer (above layer 4); (ii) Designed for C/S iii). Provides server auth, client auth and anonymous auth with digital signature; (iii) Use CipherSuite (similar to SA); (iv) Pre-master key based keying with DH, RSA, Kerberos and Fortezza.	(i) Supports DH, RSA, Kerberos, and Fortezza in key exchange; (ii) Session oriented so PFS is provided automatically.	(i) Designed as a specific key exchange protocol between Client and Server in the application layer; (ii) Does not offer mutual authentication; (iii) Not bi-directional keying; (iv) Only supports digital signature based authentication (e.g. RSA, DSA); (v) vulnerable to TCP insertion attacks.

Table 4-1 Comparison of Major Key-exchange Protocols

4.3 Design Methodology Overview

We combine ISAKMP/IKE v1, IKE v2 and JFK, which are well-designed security handshake protocols at Layer 3 into one, and transplant it to Layer 7 (the application layer). In detail, we create an XML-encoded format of the protocol instead of using binary format like its origins, IKE v2 and JFK, in the transportation layer, which is called a Key Exchange and Authentication Markup Language (KEAML), in recognition of the fact that many enterprise-level applications and Web services are XML-based.

KEAML/KEAML-KE (Key Exchange) mixes the design concepts of ISAKMP/IKE v1, IKE v2 and JFK to define a simple, efficient key exchange and authentication framework, along with a specific protocol associated with proper ID protection, such that:

- Each of the two parties generates a set of secrets (or keys) shared exclusively between the two parties;
- The key generation algorithm satisfies a different level of forward secrecy (i.e., the compromise of one key set does not lead to a compromise of the other sets), requirements by specific applications. It can be configured to offer perfect forward secrecy if necessary;
- Freshness of the messages is guaranteed (in order to defeat replay attacks);
- Key exchange is scalable (i.e., the key exchange protocol can be conducted between any pair out of a collection of parties that do not share any secrets in advance);
- Resistance to a variety of attacks is provided, which are stated in Section 4.1.1 such as man-in-the-middle attacks, variant DoS attacks, ID misbinding attacks, reflection attacks etc.;
- The choice of the public components for Diffie-Hellman exchange is simplified;
- A two-phased approach is used in order to allow the situations where a Phase II SA is much less expensive than a Phase I SA;

- Extensibility is built-in (KEAML defines a framework, while KEAML-KE is an instance of KEAML, providing the protocol for the first phase of an application-specific security handshake protocol); and

- Multiple authentication methods are supported.

As stated in Section 4.4, many design primitives from IKE/JFK are mixed into KEAML. For example, (a) a simplified two-phase infrastructure (where Phase I produces KEAML SAs while Phase II produces Protocol SAs); (b) Oakley groups that simplify the choice of public Diffie-Hellman components; and (c) the design elements to prevent DoS attacks and offer proper ID protection to the Initiator and/or the Responder.

In detail, the concept of situation is extended in KEAML. The situation in KEAML may include:

- Underlying communication protocol type (s-http, http, https, ipsec, etc.);

- Security classification of content; and

- Connectivity type (wireless, wired, public network, private network).

However, KEAML/KEAML-KE simplifies or removes aspects of IKE v1/v2 that are irrelevant or not required at the application layer. One example is that the security association tunnel mode is removed from the protocol, primarily because a proxy for an application would be in the same domain as the application (and therefore under the same security protection) without being exposed to attacks from public areas. This obviates the need to support the proxy mode as defined for Layer 3. Another example is that *DeletePayload* is removed for the picture; we can simply rely on SA lifetime to expire an SA. Moreover, traffic selectors are discarded since they are specific to the transportation layer. Improvements suggested by some publications [42], such as stateless cookies, and the removal of public encryption key variants, can also be found in Section 4.4.

In summary, KEAML provides a standardized framework for key exchange and authentication between two negotiating enterprise-level entities, while KEAML-KE (Key Exchange) is an instance of KEAML, providing the concrete protocols. In order to address the issues and requirements stated in Section 1.2.7 and Section 4.1, and benefit from the many useful features in IKE and JFK, KEAML/KEAML-KE leverages the major concepts of IKE and JFK: the

framework and protocols such as *security association (SA)* and *SA proposal*, *exchange*, the simplified *two-phase approach*, *Oakley groups, forward secrecy* and applies improvements on top of them.

4.4 Protocol Definition

In this section, we describe how KEAML/KEAML-KE is constructed in order to meet its design features (Section 4.3) within the threat model pictured in Section 4.1.2. Particularly, KEAML/KEAML-KE is defined such that on one hand, it is simply effective, efficient, providing better ID protection and resistant to DoS attacks compared with IKE v1/v2. On the other hand, it is extensible, flexible, supports more authentication methods and is more concise compared with JFK.

In general, KEAML/KEAML-KE is composed of Phase I SA exchange and Phase II SA exchange. Phase I SA exchange is composed of two rounds of messaging. The first round is KEAML SA Initiation that includes the establishment of SA and key exchange. The second round is KEAML Authentication that conducts the explicit authentication for the key exchange. Phase II (or so-called CREATE_CHILD_SA exchange in IKE v2) is dedicated to Phase II SA establishment. It employs the messages that are similar to the first two messages in Phase I, except the payloads are encrypted and authenticated.

First, the notations used to present the messaging in the discussions are introduced Second, KEAML SA Initiation messages, KEAML Authentication and Phase II SA Establishment messages are discussed.

4.4.1 Notation

I	The initiator
R	The responder
HDR	Message header

SA_{xn}	Security Association, where x can be either i (the initiator) or r (the responder), n is either *1* or *2*; 1 is used to denote Phase I SA and 2 is used to denote Phase II SA. For example, SA_{i1} is the Phase I security association or the cryptographic algorithms the initiator supports, while SA_{r2} is the Phase II security association or the cryptographic suite the responder chooses.
KE_x	Key exchange payload from x, where x can be either i or r.
N_x	Nonce, where x can be either i or r, with i indicating the nonce is created by the initiator and r implying that the nonce is generated by the responder.
IDx	Identity of x, where x can be either i (the initiator) or r (the responder).
$\{M\}_{Ke}^{Ka}$	Encryption of M with key K_e, then the MAC authentication of M with key *Ka*. M can be a message or components of a message.
$\{M\}$	The encrypted or secret form of M, where M could be a message or a component of a message. The secret form refers to the form that can only be understandable or interpretable by the two parties.
[X]	*[]* is used to indicate the component X is optional.
$Anchor_x$	The anchor of certificate authority for x, where x can be either i or r.
CERTREQ	Certificate request
CERT	Certificate
Info	Information payload
AUTH	Authentication payload
$S_x(M)$	Digital signature of message or component M with x's private key, where x can be either the initiator or the responder.
$X<x_1, x_2, x_3, \ldots, x_n>$	A component X that consists of sub components x_1, x_2, …, and x_n.

Ck_x	The cookie generated by x, where x can be either i or r.
Flg	The flags
SPI_x	Security Parameter Indexes of x, where x can be either i or r. SPI_x is the same with Ck_x.
Ver	Version numbers
$msgID$	A 32-bit integer that represents message ID.
$Secret_x$	The secret of x, where x can be either i or r.
$Hmac(M, K)$	The keyed hash of message or component M using key K. $Hmac$ is also referred as Pseudo Random Function or PRF.
IP_x	IP address of x where x can be either i or r.
SK_{seed}	The seed of session key
g^x	The exponential of x, where x can be either i or r.
SK_d	The session key to derive Phase II session keys.
SK_{ax}	The authentication key of x where x can be either i or r.
SK_{ex}	The encryption key of x where x can be either i or r.
SK_{px}	The key used to generate AUTH payload for x, where x can be either i or r.
$AuthValue_x$	The authentication value of x, where x can be either i or r.

4.4.2 KEAML SA Initiation

4.4.2.1 Overview

$I \rightarrow R$: $HDR<Ver, msgID, Flg, Cki, 0>, SA_{il}, KE_i, N_i, [\{ID_i\}], [Anchor_r]$ (1)

$R \rightarrow I: HDR<Ver, msgID, Flg, Ck_i, Ck_r>, SA_{rl}, KE_r, N_r, [CERTREQ], [PubKey_r], [S_r(KE_r|SA_{rl}|N_r|N_i)], [Info]$ (2)

The first round of the exchange is KEAML SA Initiation, which is composed of two messages (1) (2). The initiator sends all the cryptographic suites (i.e., the SA proposals, in the order of its preference) it supports (SA_{il}), the Diffie-Hellman value (KE_i) for the first cryptographic suite, and the nonce (N_i), as indicated in message (1). As an option, the initiator can send over its identity that is in a secret form ($[[ID_i]]$), which can be either a representation of the initiator's ID (interpretable only by the initiator and the responder), or the encryption of the ID with the responder's public key. If the initiator chooses to encrypt its identity using the responder's public key, the anchor ($Anchor_r$) of the CA (Certificate Authority) must exist to indicate which public key of the responder is used for the encryption, assuming the responder is a well-known public service provider, or a security SLA (Service Level Agreement) has been reached prior to this exchange.

Once the responder receives the request from the initiator, it chooses one of the proposed cryptographic suites from SA_{il} and puts it into SA_{rl} payload, generates a corresponding Diffie-Hellman value (KE_r), the nonce (N_r), and sends them over to the initiator. If the responder does not choose the first cryptographic suite from SA_{il}, the flag's proposed bit is set to true to indicate that another SA proposal instead of the first one is chosen. No matter which proposal is chosen, the responder must include the digital signature over the D-H value, the chosen cryptographic suite, the nonces, and the ID of the initiator ($S_r(KE_r, SA_{rl}, N_r, N_i, ID_i)$), if the initiator's ID is provided in the message (1). Particularly, if a secret form of ID_i (instead of encrypted form) is received, the responder can optionally send over a public key payload ($[PubKey_r]$) that contains the public key used to verify the digital signature, assuming the initiator does not have the responder's public key yet. The responder can also send over a list of certificate request payload(s) ($[CERTREQ]$) to ask for the certificate(s) of the initiator.

4.4.2.2 Message Header – Flags and Cookies

As in IKE, the header of message ($HDR<Ver, msgID, Flg, Ck_i, Ck_r>$) contains version numbers, SPIs (Security Parameter Indexes) or cookies, and flags. The cookies are generated by the initiator (Ck_i) and the responder (Ck_r), respectively. Please note that the flag property of the

KEAML message header, unlike the flag attribute in IKE, is not used to indicate whether the message is encrypted and/or authenticated.

The flags (*Flg*), which are different from the flags in IKE, include:

- One error flag to indicate if one message is an error message or a normal message; and
- One proposal flag to indicate if one response message is proposing a new SA.

For example, in the case when the responder does not select the first proposal in SA_{i1}, the responder sets the error flag in the message (2) to "false." This indicates that the message is a normal message, yet the proposal flag is set to "true" to indicate that the responder chose another SA instead of the first one in the proposals from the initiator.

The cookie generated by the initiator could be a random number, which is later used as SPI_i; while the cookie by the responder (Ck_r) is calculated as $Hmac(KE_r|N_r|N_i|IP_i|Ck_i, Version\_of\_Secret_r|Secret_r)$ or $Hmac(KE_r|N_r|N_i|ID_i|Ck_i, Version\_of\_Secret_r|Secret_r)$ if the ID_i is known to the responder at the time the cookie is computed.

4.4.2.3 KEAML Key Generation

Like the key generation of IKE, a separate set of keys, namely, SK_{ai}, SK_{ei}, SK_{pi} for I → R and SK_{ar}, SK_{er}, SK_{pr} for R → I are generated for each direction. First, the seed of session key or SK_{seed} is generated:

$$SK_{seed} = H_{mac}(g^{ir}, N_i|N_r)$$

Then, SK_d, which is used to derive Phase II keys, is generated:

$$SK_d = H_{mac}(SK_{seed}, N_i|N_r|SPI_i|SPI_r|0x01)$$

Similarly, the authentication key, encryption key, and the key to generate the *AUTH* payload for the initiator and the responder are:

$$SK_{ai} = H_{mac}(SK_{seed}, SK_d|N_i|N_r|SPI_i|SPI_r|0x02)$$

$SK_{ar} = H_{mac}(SK_{seed}, SK_{ai}|N_i|N_r|SPI_i|SPI_r|0x03)$

$SK_{ei} = H_{mac}(SK_{seed}, SK_{ar}|N_i|N_r|SPI_i|SPI_r|0x04)$

$SK_{er} = H_{mac}(SK_{seed}, SK_{ei}|N_i|N_r|SPI_i|SPI_r|0x05)$

$SK_{pi} = H_{mac}(SK_{seed}, SK_{er}|N_i|N_r|SPI_i|SPI_r|0x06)$

$SK_{pr} = H_{mac}(SK_{seed}, SK_{pi}|N_i|N_r|SPI_i|SPI_r|0x07)$

The methodology of key generation of JFK is similar to that of IKE. Please see [2] for the details.

4.4.3 KEAML Authentication

4.4.3.1 Overview

$I \rightarrow R$: $HDR<Ver,msgID,Flg,Ck_i,Ck_r>$, N_i, N_r, $KE_i^{'}$, KE_r, SA_{i1}, SA_{r1}, $\{ID_i\}_{SK_{ei}}^{SK_{ai}}$, $\{[CERT], [CERTREQ], [ID_r], AUTH, [SA_{i2}]\}_{SK_{ei}}^{SK_{ai}}$, $[\{Info\}_{SK_{ei}}^{SK_{ai}}]$ (3)

$R \rightarrow I$: $HDR<Ver,msgID,Flg,Ck_i,Ck_r>$, $\{ID_r, [CERT], AUTH, [SA_{r2}]\}_{SK_{ei}}^{SK_{ai}}$, $[\{Info\}_{SK_{ei}}^{SK_{ai}}]$ (4)

Upon the completion of the first round (i.e., messages (1) and (2)), the encryption key, authentication and the keys for AUTH calculation are generated for both directions. Therefore, both messages (3) and (4), including the optional information payloads, are encrypted and authenticated. However, to establish the SAs, an explicit authentication has to be conducted to guarantee to one party that the SAs are only shared by the other party that it thinks it is sharing with.

In message (3), the nonces, the D-H public components (i.e. $KE_i^{'}$, KE_r) of both the initiator and the responder, and SA_{i1}, SA_{r1} are echoed back to the responder, so the responder does not need to cache the state information for the session in order to verify the cookie (see Section 4.4.2.2 for the details). Similarly, in message (4), the payloads are encrypted and authenticated by the key established. The most important thing that occurs at this stage is that by inserting their individual

identities (i.e., ID_i in message (3) and ID_r in message (4)) into this round of exchanges that is protected by the encryption keys and the authentication keys, both parties prove their knowledge of the secrets corresponding to their identities. Moreover, the authentication payloads (AUTH) provide further integrity protection to the content of message (3) and (4), and their preceding messages (i.e. message (1) and (2)).

The initiator and the responder may include certificates in *[CERT]* payloads to offer the public key used to verify the authentication payload (AUTH). The initiator can also contain ID_r to indicate which identity of the responder it wants to talk to, if it has not done so by using $Anchor_r$ in message (1). Finally, the initiator can optionally start the negotiation of Phase II SA using SA_{i2} payload.

4.4.3.2 Calculation of Authentication Payload

There are two ways to calculate the authentication value:

- Sign over a block of data; or
- MAC over a block of data with a pre-shared secret.

The first method assumes that a public key is available for signing data, while the second assumes that a pre-shared secret can be used to MAC the sequence of data. It is flexible enough for a party in the exchange to choose its own authentication method. That is to say, the two parties do not have to use the same authentication method; using MAC or a digital signature can be specified in its own AUTH payload that is protected by SK_a and SK_e. Nevertheless, one party can choose to include AUTH algorithm as part of SA proposal in order to force the other party to use the same (strong) algorithm for security considerations.

When the initiator chooses a digital signature as its authentication method, it calculates the authentication value with the following formula:

$$AuthValue_i = S_i(SPI_i|0|N_i|g^i|g^r|[ID_r]|SA_{i1}|\ SA_{r1}|N_r|Hmac(SK_{pi},\ IDi))$$

When the responder chooses a digital signature as the authentication method, the formula is:

$$AuthValue_r = S_r(SPI_r|SPI_i|N_r|g^r|g^i|ID_i|\ SA_{r1}|SA_{i1}|N_i|Hmac(SK_{pr},\ IDr))$$

When the initiator employs a pre-shared key instead of a digital signature as the authentication method, a MAC formula is used:

$$AuthValue_i = Hmac\ (Pre\text{-}shared\_Secret,\ SPI_i|0|N_i|g^i|g^r|[ID_r]|SA_{i1}|\ SA_{r1}|N_r|Hmac(SK_{pi},\ IDi))$$

Similarly, when the responder employs a pre-shared key as the authentication method, the MAC formula is:

$$AuthValue_r = Hmac(Pre\text{-}shared\_Secret,\ SPI_r|SPI_i|N_r|g^r|g^i|ID_i|SA_{r1}|\ SA_{i1}|N_i|Hmac(SK_{pr},\ IDr))$$

Apparently, the authentication based on a pre-shared secret is weaker than the digital signature-based method, because of the relative predictability of the pre-shared secret. The pre-shared secret derived from a password is even weaker due to its vulnerability to dictionary attack. If a pre-shared secret is the only option available (e.g., the entity does not have a public/private key pair), it is highly recommended to avoid using a password-deriving secret; a safe out-band channel should be used to share the secret, whose size should be at least 128 bits.

4.4.4 Phase II SA Establishment

4.4.4.1 Overview

$$I \rightarrow R:\ HDR<Ver,msgID,Flg,Ck_i,Ck_r>,\ \{SA_{i2},\ [KE_i],\ N_i\}^{SK_{ai}}_{SK_{ei}},\ [\{Info\}^{SK_{ai}}_{SK_{ei}}] \qquad (5)$$

$$R \rightarrow I:\ HDR<Ver,msgID,Flg,Ck_i,Ck_r>,\ \{\ SA_{r2},\ [KE_r],\ N_r\}^{SK_{ai}}_{SK_{ei}},\ [\{Info\}^{SK_{ai}}_{SK_{ei}}] \qquad (6)$$

Phase II SA Establishment exchange is much simpler than its Phase I equivalence, because it is conducted in the secure channel that has already been established by Phase I SA. Yet the structure of the protocol is designed intentionally so that it is similar to messages (1) and (2), except that every payload is protected and the cookies are used to indicate the Phase I SA to be used in the Phase II negotiation. The cookies here, with the message ID, are used to uniquely identify the about-to-be-established Phase II SA, or the cookies concatenated with the message ID are served as SPIs in Phase II per direction. For example, in Phase II, $Ck_i \mid msgID$ is SPI_i while $Ck_r \mid msgID$ is SPI_r.

As shown in the messages, the request from the initiator may contain extra KE payload for stronger forward secrecy of Phase II SA. If KE_i payload exists in the request, the responder must respond with a KE_r payload that contains the corresponding Diffie-Hellman public value it generates.

4.4.4.2 Phase II Key Generation

As mentioned in Section 4.4.2.3, Phase I results in a key SK_d, from which Phase II keys can be created.

From SK_d, the keying material or the seed key for Phase II (SK_{seed2}) is generated:

$SK_{seed2} = H_{mac}(SK_d, N_i|N_r)$

Where N_i and N_r are the nonces from Phase I SA Initiation exchange, or the nonces from Phase II Key Establishment exchange.

If additional Diffie-Hellman values are found in the Phase II exchange, then the formula for the seed key is:

$SK_{seed2} = H_{mac}(SK_d, g^{ir}|N_i|N_r)$ where g^{ir} is the additional D-H value

As in Phase I, multiple SAs and keys can be created from one Phase II negotiation. If this is the case, then the following rules (similar to IKE v2) should be obeyed:

- All keys used on the data stream from the initiator to the responder must be created prior to any creation of the keys for the reverse direction; and
- If both encryption and authentication keys exist in the protocol, then the encryption key is created before the authentication key.

For example, assume we have one SA for each direction, and each SA has both an encryption key and an authentication key. Then,

$SK_{ei} = H_{mac}(SK_{seed2}, [g^{ir}]|N_i|N_r|0x01)$

$SK_{ai} = H_{mac}(SK_{seed2}, SK_{ei}|N_i|N_r|0x02)$

$SK_{er} = H_{mac}(SK_{seed2}, SK_{ai}|N_i|N_r|0x03)$

$SK_{ar} = H_{mac}(SK_{seed2}, SK_{er}|N_i|N_r|0x04)$

4.5 Review of KEAML/KEAML-KE Protocol Design

Two aspects of design goals need to be reviewed. One is related to the normal features of a protocol regarding the simplicity, efficiency, extensibility and standardization; the other is about the security features of the protocol, which are the greatest concern yet the most difficult portion of the design. In the first subsection, we briefly go over how the normal features are achieved. In the second subsection, we discuss how the security features are satisfied. Finally, an introduction is given to the formal security analysis, which consists of future research for KEAML/KEAML-KE.

4.5.1 Normal Features

4.5.1.1 Simplicity

KEAML/KEAML-KE is much simpler than IKE v1 and v2. Compared with IKE v1's three major modes for both Phase I and Phase II (Main mode, Aggressive Mode for Phase I, and Quick Mode for Phase II), KEAML only has six messages. Four of them are for Phase I and two of them for Phase II negotiation. IKE v1, in contrast, has 24 messages for the main mode and 12 messages for the aggressive mode, as well as three messages for the quick mode in Phase II. Although IKE v2 reduces the number of messages to six (four for Phase I and two for Phase II), by extracting the authentication methods from the complex messaging of IKE v1, IKE v2 introduces two optional messages in order to make the protocol resistant to being flooded by DoS attacks. Even compared with JFK, KEAML achieves similar security goals yet is more compact compared with JFK's requiring two sets of protocols - One set is for the client/server while the other is designed for peer-to-peer secure communication.

KEAML inherits the concept of the two-phase approach from IKE v1, and like IKE v2, it simplifies the two-phase design by (a) allowing Phase II SA negotiation in Phase I authentication exchange; and (b) using similar constructs in Phase I and Phase II messaging. For example, the recursive key-generation formula is used in both Phase I and Phase II key generation.

The design to support multiple authentication methods is simplified by eliminating the constructs of messaging that are dependent on concrete authentication methods. Like IKE v2, the public key encryption authentication method is discarded, since it does not add much value but is just another way to share a secret between two parties, which can later be used for authentication. The direct result of decoupling the authentication methods and the messaging construction is the great reduction in the number of messages, or the number of rounds required by the SA negotiations.

4.5.1.2 Efficiency

Simplification of the protocol improves the efficiency of the protocol. Like JFK, in order to save the entities or the parties from costly modular exponential calculations, KEAML allows the reusage of Diffie-Hellman values across several SA negotiating sessions within the initiator's or the responder's forward secrecy interval. This interval is exposed to the application as a configurable runtime parameter that can be adjusted based on the specific application forward secrecy requirement. For example, if the application wants perfect forward secrecy, the interval can be adjusted to 0 so that the Diffie-Hellman value will not be reused, but the entities have to generate the expensive exponentials for every key exchange.

KEAML adopts compact messaging design to enhance the efficiency of communication. In a situation where the responder does not choose the cryptographic suite that the initiator prefers, instead of sending an error message and starting over, the responder can propose a new SA. The responder sends it back to the initiator with the new exponential value. When the initiator receives the proposal, and if it accepts it, the initiator can silently accept it and send back its new Diffie-Hellman value corresponding to the new SA, without using another message to reach the key agreement.

As like in IKE v2, KEAML allows a piggyback setup of Phase II SAs to further compact the messaging. Finally, the new group exchange is discarded. Instead, the security association is defined so that it can be customized.

4.5.1.3 Extensibility and Standardization

While simplifying the design of the messaging, KEAML stays as extensible as its IKE orgins by retaining (a) the concept of the separation of key exchange and SA parameter negotiation; and (b) the concept of the two-phase approach. Some publications argue that two-phase approach should be discarded because of (a) the assumption that applications now use more powerful encryption algorithms and stronger keys than before; and (b) frequent keying makes less sense. However, based on the belief that not all applications (many of them are still employing less powerful encryption algorithms) are using strong keys, and a generic key exchange protocol should not rely on specific strong encryption algorithms, we keep the two-phase approach as IKE v2 does. Besides, the two-phase approach offers a platform to decouple the protocol for Phase II from the the protocol for Phase I, which is a feature that a good protocol framework should provide.

Several features of KEAML support it to become a potential standard framework and protocol for key exchange. First, standardized Diffie-Hellman groups or Oaklay groups are supported. Second, its flexibility expands the domains where it can be used. Third, it originates from well-designed key exchange protocols such as IKE, JFK etc., which have been proven highly secure.

4.5.2 Security Features

4.5.2.1 Perfect Forward Secrecy Vs. Adjustable Forward Secrecy

As mentioned, KEAML offers adjustable forward secrecy for applications, instead of providing perfect forward secrecy (PFS) all the time. This design concept is taken from JFK based on the reasoning that applications do not always require PFS at all costs. Instead, perfect forward secrecy is not always required and a certain amount of forward secrecy can be sacrificed in order to gain efficiency or free the entity from heavy calculation burdens. In KEAML, the forward secrecy interval is used to allow the application to have adjustable forward secrecy instead of absolute forward secrecy or PFS. It is up to the application to decide the amount of forward secrecy that is required, without compromising the confidentiality of the keys and the data protected by the keys.

4.5.2.2 Authentication

It is well known that pure Diffie-Hellman key exchange does not provide explicit authentication. That is why, like its orgins in IKE and JFK, KEAML uses HMAC to prevent identity misbinding, and a digital signature or HMAC (when the two parties prefer to use a preshared key for authentication) to prevent man-in-the-middle attacks and provide integrity protection at the same time.

The protection can be observed in the second round of exchanges in Phase I or the key authenticion between the two parties. The calculation of the authentication payload includes the formula $Hmac(SK_{px}, IDx)$, where x can be i (or the initiator) or r (or the responder). This formula associates the identity with the session key that is known only to the two parties (the secrecy of the session key is guaranteed by the Diffie-Hellman exchange). Therefore, it provides protection against identity misbinding.

The calculated result of $Hmac(SK_{px}, IDx)$ is concatenated with other components of the messages and signed or MACed to provide integrity protection to the messages of the first round message exchange, and prevent man-in-the-middle attacks. To provide integrity protection, all the components which appeared in the previous messages (including SAs, SPIs, nonces and Diffie-Hellman values) are signed or MACed to provide the message with integrity. SPIs and nonces are included in the data block to provide uniqueness and "refreshness" of the signature or MAC values. This will avoid the signature or MAC value being used by potential replay attacks. Moreover, the calculation of authenticion for one party takes into account the identity of the other party to *explicitly* authenticate that the SA is established between itself and the other party. Finally, the order of SPIs and Diffie-Hellman values are reverted in the two authentication calculations for the two parties, to eliminate the possibility of being reflection attacked.

The messages in the second round of the Phase I exchange, or message (3) and (4), are authenticated by the authentication keys (SK_{ai} *and* SK_{ar}).

4.5.2.3 Resistance to DoS Attacks

As mentioned before, IKE v1 does not offer the facility in protocol design to avoid denial-of-service (DoS) attacks. IKE v2 provides resistance to simple flooding DoS attacks by using stateless cookies, which was suggested in Photuis, yet at the cost of two additional messages.

KEAML incorporates resistance to simple flooding DoS attacks into the key exchange and authentication exchange without adding more messages except a one-time calculation of the stateless cookie. The initiator first sends the SA initiation request to the responder with a random number $cookie_i$, the responder then calculates $cookie_r$ as $Hmac(KE_r|N_r|N_i|IP_i|Ck_i, Version\_of\_Secret_r|Secret_r)$ or $Hmac(KE_r|N_r|N_i|ID_i|Ck_i, Version\_of\_Secret_r|Secret_r)$ if the secrecy form of ID_i is sent to the responder in the message (1). In the formula, KE_r, N_r, N_i, Ck_i, ID_i, it will be echoed back to the responder in message (3); IP_i can be obtained from the underlying network API. Therefore, the responder does not have to cache any of these locally, for the ongoing session. $Secret_r$ is the local secret that the responder uses to gurantee $cookie_r$ only producable by the responder itself. Since $Secret_r$ changes periodically, the version of the secret needs to be accounted for in the calculation.

Upon receipt of the message (3), without keeping any state information (including $cookie_r$) for the session, the responder re-calculates the $cookie_r$ and compares it with Ck_r in message (3), which is echoed back by the initiator. If the two match, it indicates that the initiator has the ability to reply instead of being a virtual node with a fake IP address. The second calculation of the cookie takes the IP address that is obtained from the underlying Network API, or the ID of the initiator (assuming the initiator sends it over in message (1)), which cannot be forged (IP address) or is confidential to whoever sends the message (1) (ID of the initiator). The responder can therefore be sure that the initiator who sends message (1) is replying with message (3) instead of an adversary who has a cookie jar to replay the cookie. That is to say, if the two calculations of $cookie_r$ do not match, the responder is under either a simple flooding or a cookie-jar attack.

Besides detecting potential DoS attacks, avoiding computation bandwidth and memory space being consumed and exhausted by adversaries are also important goals. Like JFK, KEAML is designed to avoid and/or defer heavy calculations and state caching. As mentioned before, the forward secrecy interval is applied to make the forward secrecy configurable. This action reduces the amount of calculation in messages (1) and (2). In message (2), the only crypto calculation is MAC in order to generate the stateless cookie or SPI_r, and no or little state is generated and kept

in the responder side; similarly, in message (3), no state is generated since the initiator echoes back all the information that is required to calculate $cookie_r$ or SPI_r.

4.5.2.4 Identity Protection

Attacks against the identities of two parties are categorized as *passive* and *active*. Passive is refered to as eavesdropping to learn who is communicating. Active is known as identity probing that is usually launched by a man-in-the-middle.

As described in Section 4.4, the initiator can choose whether to include $\{ID_i\}$ and $Anchor_r$. Including or not including them will result in two different identity protection services.

If both $\{ID_i\}$ and $Anchor_r$ are included, or if the initiator sends ID_i that is encrypted with the responder's public key indicated by $Anchor_r$, the responder has to include the $S_r(KE_r|SA_{r1}|N_r|\ N_i)$ in message (2) to authenticate to the initiator that it is the responder that the initiator intends to talk to. If the initiator wants to include only $\{ID_i\}$, a security SLA has to be established outboard prior to the exchange. The secret form (e.g., a random number mapping to ID_i) and the identifier of the public key (that the initiator will use) for the responder will therefore be determined. Since the public key identifier is known in this case, $S_r(KE_r|SA_{r1}|N_r|\ N_i)$ can also be used for authentication purposes. Because the authentication field in message (2) cannot be created without knowning the responder's private key and ID_i transmitted in encrypted or secret form, the initiator's identity is protected from both passive and active attacks.

On the other hand, the responder is protected against both passive and active attacks if only $\{ID_i\}$ is contained in message (1), because the identity of the responder is sent in cipher in message (4), and the responder can determine not to send its identity if it fails to verify the initiator as a legimate party. Therefore, it is protected against a man-in-the-middle identity prober as well. However, if $Anchor_r$ is included in message (1), it is possible for an eavesdropper to learn the identity of the responder that is used in the exchange. Yet, including $Anchor_r$ implies that the idenitity of the responder is public anyway and does not need to be protected.

If neither $\{ID_i\}$ nor $Anchor_r$ is present in message (1), the responder is protected from both passive and active attacks, because its identity is never sent in clear and is only sent after verifying a valid identity for the initiator. The initiator is also protected from passive attacks but becomes vulnerable to a man-in-the-middle prober. This is because a man-in-the-middle can

pretend to be the responder and would be able to decrypt ID_i from message (3). However, it will be detected by the initiator to be an adversary in message (4) since the man-in-the-middle will definitely fail in providing an *AUTH* field without knowing the private key of the responder or the pr-shared secret.

In summary, the initiator is protected from both passive and active attacks when $\{ID_i\}$ presents in message (1), but is only protected from passive attacks otherwise; the responder is protected from both passive and active attacks in either case. However, when $Anchor_r$ presents message (1), the responder may not be identity protected at all.

4.5.2.5 Avoiding Protocol Design Pitfalls

Section 4.1.3 discusses the design principles such as explicitness, and clearness, with which the key scope of the design of protocols should comply. In this sub-section, we discuss how KEAML satifies some of these principles.

First of all, KEAML is XML-based and much more self-descriptive than its binary origins such as IKE and JFK. Therefore, it is much more explicit than any binary encoded security protocols. Nevertheless, special attention has been paid to the protocol design, the design of each component of the protocol in order to improve the completion and accuracy of the information, and make them more explicit and clearer. For example, if message (2) is proposing a new cryptographic suite or SA, a bit in the message header flag will be set to "true" to indicate a new proposal is included in the message. A KE_r under the new proposal presents in the message, and a bit of the flag is set if the information payload is included. In the message header of KEAML, a version field presents to indicate the version of the protocol. As one of the by-the-thumb rules, KEAML has the version number mandated and contained in the message header.

The design of key scope is to minimize the use of a specific key as much as possible. The bottom line is that a key should not serve for multiple purposes, if possible. In KEAML, one specific key is used for only one purpose; particularly, a key can be defined such that it is used for only one direction instead of both. For example, in an established KEAML SA, three keys per direction can be found – one is for encryption, the seond is for authentication, and the third is for AUTH payload, so there are six keys altogether.

Moreover, KEAML is structurally asymmetric in order to be immune to reflection attacks. For example, the calculation for the AUTH payload from the initiator to the responder is $S_i(SPI_i|0|N_i|g^i|g^r|[ID_r]|SA_{i1}|SA_{r1}|N_r|Hmac(SK_{pi}, IDi))$ while the calculation for the reverse direction is $S_r(SPI_r|SPI_i|N_r|g^r|g^i|ID_i|SA_{r1}|SA_{i1}|N_i|Hmac(SK_{pr}, IDr))$. They are not structurally equivalent; note that the order of g^i and g^r is swapped.

Besides complying with the major design principles discussed in Section 4.1.3, other security constraints are imposed on KEAML in order to improve the security of the protocol. For example, the size of nonces must be at least 128-bits; the output of HMAC must be greater than 128 bits. Some fine tuning is also optional to increase the security of the protocol. For example, forward secrecy could be enhanced by additional Diffie-Hellman values in Phase II.

4.5.3 Security Analysis Introduction

In this section, we give a brief introduction of security analysis on key exchange and authentication protocols. The security analysis of KEAML will entail further research.

BAN logic [11] discusses the reasoning on the establishment of the beliefs of communicating parties, or the reasoning about how to determine whether the exchanged message or information between parties is trustworthy.

BAN logic contains a set of logic postulates that concern such security protocol relevant matters as the interpretation of messages and the freshness of messages etc. Here we introduce only the three postulates that are most relevant to this thesis:

1. The *message meaning* rule, or the rule about under what conditions A can believe that B once said X (denoted as $A \mid\equiv B \mid\sim X$);

2. The *nonce-verification* rule, or the rule about the conditions under which A can believe that B still believes what it once said (denoted as $A \mid\equiv B \mid\equiv X$); or the statement X is still fresh, and

3. The *jurisdiction* rule, or the rule for A to believe X (denoted as $A \mid\equiv X$).

In the next pargraph we briefly discuss the three rules.

In the *message meaning* rule, BAN logic states that if A receives a message X in cipher, which is encrypted with the key shared by A and B ($A \leftrightarrow^{K} B$), then A is able to believe that B once said X.

The second rule, or the *nonce-verification* postulate, gives the reasoning about the freshness of the message. In this rule, if A believes that B once said X (the reasoning result of the first rule), and X is still fresh ($A \mid\equiv \# X$), then A will believe that B still believes X. The freshness of X can be proved by a timestamp, nonce or some other indicator contained in X, to show that it is produced in the current run of the protocol.

Finally, the *jurisdiction* postulate gives the reasoning for A to believe X. If A believes B still believes what it once said (the reasoning result of the second rule), and B is the authority on X or B is trusted on X ($A \mid\equiv B \mid\Rightarrow X$), then A will believe X.

The three postulates can be denoted by the following formula. The notation on the top is the premises and the notation on the bottom is the result.

$$\frac{A \mid\equiv A \leftrightarrow^{K} B, A \triangleleft \{X\}_{K}}{A \mid\equiv B \mid\sim X} \tag{1}$$

$$\frac{A \mid\equiv \# X, A \mid\equiv B \mid\sim X}{A \mid\equiv B \mid\equiv X} \tag{2}$$

$$\frac{A \mid\equiv B \mid\Rightarrow X, A \mid\equiv B \mid\equiv X}{A \mid\equiv X} \tag{3}$$

Besides BAN logic, a number of formal analysis methodologies have been developed in an attempt to describe both the security requirements and a corresponding model of the protocol that would capture the security properties. One of the best known is a general approach for analysis and verification of authentication properties built on top of the general Communicating Sequential Processes' (CSP) semantic framework [50]. The CSP approach expresses security protocols in a natural and precise way, and can formally reason about the properties described. The Needham-Schroeder public key protocol is illustrated by this approach in both single run and multiple concurrent run examples. In [35], it presents how to define requirements with CSL, and the NRL analyzer is used to define the logic connectives between the events of the protocols. TheNRL

analyzer uses an analysis model specified as communicating state machines, or a set of honest participants and one intruder, each of which has a set of local state variables whose definitions are based on the cryptographic protocol being analyzed.

Research has been carried out specifically on the security analysis of AK and AKC [9]. The formal definitions of AK/AKC protocols are proposed in a model of distributed computing, which is a variant of the Bellare-Rogaway model. In this model, the formal definitions of the requirements of AK/AKC, and the corresponding practical, provably secure solutions in the random oracle model are provided.

The other efforts regarding formal security analysis include, but are not limited to:

(a) A symbolic logic-based formal method to analyze the correctness of key exchange protocols, presented in [23];

(b) A knowledge and tense modalities-based logic where many "important security properties of protocols" can be expressed [48], which is used to express the security properties of Needham-Schroeder protocol and even BAN logic;

(c) A spi-calculus and type-checking-based method to analyze the authenticity properties of security protocols [21];

(d) A modular approach to designing and analyzing key agreement protocols [6]; and

(e) A formalized analysis of key exchange authentication protocols, which provides a security definition of the protocol that simplifies security analysis on such protocols [12].

Verification of the protocol is another important aspect of security analysis. Some verification tools are developed to detect both design and implementation flaws. For example, in [8], verification tools are provided for both implementation and design. The implementation can be either symbolic or concrete. The concrete implementation is used for interoperability testing, while the symbolic implementation is used for formal verification of the protocol. The verification tools can also automatically derive verifiable models from code.

Some more specific formal security analyses are conducted on IKE and similar systems. In [13], the signature-based key exchange protocols (instead of pre-shared key-based) of IKE were

analyzed with a key-exchange model and an unauthenticated-links model, where an intruder can launch an MITM attack and have access to secret information such as session state, session key, and all the information in the memory. The NRL protocol analyzer is also used to analyze IKE v1 [36]. The goal of the analysis is to determine harmful interactions between the IKE subprotocols and whether the protocols satisfy the secrecy, authentication and PFS in Phase II. Modifications and improvements are made on NRL to handle the analysis on IKE. Fortunately, except for the ambiguities of the IKE v1 specifications, no harmful subprotocol interactions were found.

Further, security analysis has been conducted on group key establishment protocols. A systematic way to analyze the security properties of group key agreement protocols is developed from Bresson et al.'s work [41]. As a result, some attacks were noted against several protocols such as A-GDH.2 and its extension, SA-GDH.2.

4.6 Basic Syntax and Semantics

The KEAML schema is carefully designed to provide all the features its binary-encoded original provides, including extensibility and flexibility, while guaranteeing the correctness of exchange templates, in addition to leveraging partial encryption/signing of the XML messages.

The specification defines the syntax and semantics for XML-encoded Key Exchange and Authentication Markup Language (KEAML) basic elements, and protocols. These constructs and their contents can be encrypted with XML Encryption specification [XML-ENC] and signed with XML Signature specification [XML-SIG]. Files containing the most up-to-date of the KEAML schemas [KEAML-XSD] and protocol schemas [KEAMLP-XSD] are available through the author. However, the current schemas are also available in the Appendices of this thesis.

The following sections introduce the important primitve elements used in KEAML. For detailed information about the syntax and semantics of these elements, please refer to Appendices A and B.

4.6.1 Schema Organization and Namespaces

Conventional XML namespace prefixes are used throughout the listings in this specification to stand for their respective namespaces, as follows:

- The prefix keaml: stands for the KEAML namespace.
- The prefix keaml-ke: stands for the KEAML-KE namespace.
- The prefix ds: stands for the *W3C XML Signature* namespace.
- The prefix xenc: stands for the *W3C XML Encryption* namespace.

The KEAML schema is imported into the KEAML-KE schema. The schemas of both XML Signature and Encryption are also imported.

4.6.2 Restrictions and Extensibility

- Restrictions on Constructs

  Restrictions on constructs are dealt with using XML complexType structures. For example, if an exchange message can contain only certain types of elements but not others, these element types are explicitly specified in the complex type definition for that exchange message.

- Extensible Framework

  In order to define KEAML as the framework that is flexible and extensible and make KEAML-KE object-oriented, XML's abstract, restriction and extension mechanisms are employed. For example, many elements in KEAML are defined as abstract so they can be extended by elements in KEAML-KE.

- Borrow Existing Elements

Existing elements from other XML security specifications (such as *ds:keyValue*, *ds:X509Certificate*, *ds:SignatureValue*, *ds:KeyInfo*, and *ds:DigestValue*, etc.) are imported and reused in the KEAML/KEAML-KE specifications.

4.7 KEAML-KE Basic Elements

Some basic elements are briefly introduced in this section. For details, please see Appendix A and Appendix B.

4.7.1 SA Data Attributes

Two categories of SA data attributes exist - the attributes for Phase I and the attributes for Phase II. In KEAML, only Phase I attributes are defined and described. The definition of Phase II SA attributes belongs to a specific protocol that applies KEAML and is beyond the scope of this paper. An example of such a document would be KEAML 1.0 Profile of SAML for Subject Attributes Exchange.

4.7.2 Diffie-Hellman Public Values and Oakley Group

DH public values must be specified either by using a defined group description (Oakley) or by defining all attributes of a group using New Group Mode.

DH public values are defined in *keaml-ke:GroupType*.

The elements *Type* and *Desc* are mandatory. When the predefined Oakley Group is adopted, the *Type* of the group can only be one of MODP (modular exponentiation group), ECP (elliptic curve group over GF[P]), and EC2N (elliptic curve group over GF[2^N]); correspondingly, the *Desc* of the group is one of MODP_768 (default, 768-bit MODP group), MODP_1024 (1024-bit MODP group), EC2N_GP_155 (EC2N group on GP [2^155]), and EC2N_GP_185 (EC2N group on GP [2^185]).

4.7.3 SA LifeType

Two basic categories of SA life types exist – the length of the data being protected and the length of time an SA is used, yet it can be extended to add more categories.

4.7.4 Pseudo Random Function (PRF)

PRF is one of the mandatory elemenets in SA data attributes, because it is used by both parties to calculate the keys.

4.7.5 Situation

As mentioned, an application level environment in the scenarios described in Section 1.1 may consider detectable environment parameters as the content of *situation*, such as connectivity type, the security level of underlying communication channels and the security classification-level , etc.

4.7.6 EncryptedPayloadType

keaml:EncryptedPayloadType derives from the basic payload type *keaml: PayloadType*, which is abstract. Encrypted Payload Type itself is also an abstract type, which in turn is the parent of other encrypted payload types such as *keaml:EncryptedIDPayloadType*, *keaml:EncryptedAuthnReqPayloadsType* and *keaml:EncryptedAuthnRespPayloadsType* etc.

4.7.7 Keaml:KEPayloadType and Keaml-ke: KEPayloadType

Keaml:KEPayloadType is defined as the base for all key exchange payload types. For example, *Keaml-ke:KEPayloadType*, which is derived from it, contains DH public component values, which is represented by *hexBinary*.

4.8 KEAML Protocol

KEAML protocol consists of a set of key-exchange and authentication messages whose definitions are based on the framework provided by KEAML.

KEAML protocol generates one Phase I SA, but with two sets of unidirectional keys. One is for the direction from the Initiator to the Responder, and the other from the Responder to the Initiator. KEAML protocol can also be used to generate two Phase I SAs for each direction. In order to generate the I2R (from the Initiator to the Responder) SA, the initiator has to propose a list of proposals to the Responder. For the R2I SA, the Responder has to behave as the Initiator does to the Responder, who is the Initiator for the I2R SA.

The definition of the protocol has been detailed in Section 4.4. For the syntax of KEAML/KEAML-KE, please refer to the KEAML/KEAML-KE schemas in Appendix A and Appendix B.

4.9 Example

Here we demonstrate, with simplified XML representations of messages, how KEAML-KE protocols are used in ad hoc secure communication between a Meta Policy Server (MPS) in one domain and a Subject Attribute Authority in another domain [46]. KEAML-KE with a public key signature (Phase I) is used. Please note that <*KE*> contains one party's Diffe-Hellman public component, not the shared key itself. For the sample messages with complete syntax, please refer to Appendix D.

First, MPS sends a list of proposals with the pre-generated KE payload to the subject AA, and the subject AA chooses one proposal and returns it with generated KE to MPS, as shown in the list below:

```
<SAKERequestMsg>
 <Header MajorVersion="1" MinorVersion="0">
  <InitiatorCookie>D4861B4F</InitiatorCookie>
   <ResponderCookie>00</ResponderCookie>
    <MessageID>1</MessageID>
     <Flags>4</Flags>
 </Header>
 <Body>
```

```
  <SA>
   <Proposal number="1">
    < ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-type:keaml</ ProtocolID>
     <Transform number="1">
      <TransformType>urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
      <TransformID>urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
       <DataAttribute>
       <EncryptionAlg Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
       <AuthnAlg type="urn:oasis:names:tc:keaml-ke:1.0:authnmethod-type:publickey-signature">
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       </AuthnAlg>
       <Group><Type><Oakley>MODP</Oakley></Type><Desc><Oakley>MODP_1024</Oakley>
         </Desc></Group>
       <PRF Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
       <Integ Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
       <LifeType><Type>KILOBYTES</Type></LifeType>
       </DataAttribute>
      </Transform>
     </Proposal>
     <Proposal number="2">
       < ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-type:keaml</ ProtocolID>
        <Transform number="1">
          <TransformType>urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
           <TransformID>urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
           <DataAttribute>
             <EncryptionAlg Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
             <AuthnAlg type="urn:oasis:names:tc:keaml-ke:1.0:authnmethod-type:publickey-signature">
             <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
             </AuthnAlg>
             <Group><Type><Oakley>EC2N</Oakley></Type><Desc>
<Oakley>EC2N_GP_185</Oakley></Desc></Group>
             <PRF Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
             <Integ Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
             <LifeType><Type>KILOBYTES</Type></LifeType>
           </DataAttribute>
          </Transform>
     </Proposal>
     <keaml:Situation xsi:type="keaml-ke:SituationType">
       <SituationID>urn:oasis:names:tc:keaml:1.0:situation-id:security-classification-
level</SituationID><Value>C</Value></keaml:Situation>
    </SA>
    <KE><DHPublicComp>3082 ... 7010943</DHPublicComp></KE>
    <Nonce>47B700E7DE17ACB6</Nonce>
  </Body>
</SAKERequestMsg>
<!-- SA KE Response Message -->
< SAKEResponseMsg>
 <Header MajorVersion="1" MinorVersion="0">
  <InitiatorCookie>D4861B4F</InitiatorCookie>
  <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
  <MessageID>1</MessageID>
  <Flags>0</Flags>
 </Header>
 <Body>
  <SA>
   <Proposal number="1">
```

```
    <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-type:keaml</keaml:ProtocolID>
    <Transform number="1">
    <TransformType>urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
     <TransformID>urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
      <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
        <EncryptionAlg Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
        <AuthnAlg type="urn:oasis:names:tc:keaml-ke:1.0:authnmethod-type:publickey-signature">
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        </AuthnAlg>
        <Group><Type><Oakley>MODP</Oakley></Type><Desc><Oakley>MODP_1024</Oakley>
         </Desc></Group>
        <PRF Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
        <Integ Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
        <LifeType><Type>KILOBYTES</Type></LifeType>
      </DataAttribute>
     </Transform>
    </Proposal>
   </SA>
   <KE><DHPublicComp>3082012430819...AC</DHPublicComp></KE>
   <Nonce>9358BE10CBE6B574</Nonce>
  </Body>
</SAKEResponseMsg>
```

Then, MPS and subject AA finish Phase I negotiation by performing mutual authentication, as shown below. As a result of these steps, a KEAML SA or Phase I SA is established and, as in IKE, the keys are generated and shared exclusively between MPS and the subject AA (pleaes see Section 4.4.2.3 for the detail).

```
<!-- Key authentication request message  -->
<KeyAuthnRequestMsg>
 <Header MajorVersion="1" MinorVersion="0">
  <InitiatorCookie>D4861B4F</InitiatorCookie>
  <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
  <MessageID>1</MessageID>
  <Flags>0</Flags>
 </Header>
 <Body>
  <InitiatorNonce>47B700E7DE17ACB6</InitiatorNonce>
  <ResponderNonce>9358BE10CBE6B574</ResponderNonce>
  <InitiatorKE><DHPublicComp>30..10943</DHPublicComp></InitiatorKE>
  <ResponderKE><DHPublicComp>3082..18AE7AC</DHPublicComp></ResponderKE>
  <InitiatorSA> .. </InitiatorSA>
<ResponderSA> .. </ResponderSA>
<EncryptedID><xenc:EncryptedData><xenc:CipherData>
<xenc:CipherValue>fQ .. ypvK8q1EVeYAnRPnl00=</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
<ds:DigestValue>IQKPli9hPUzzxOdA5XHcN0usgC4=</ds:DigestValue>
</EncryptedID>
<EncryptedReq><xenc:EncryptedData><xenc:CipherData>
<xenc:CipherValue>3ngUu .. J2lBDNE2Fkndh/VrljpFqEh3k=</xenc:CipherValue>
</xenc:CipherData></xenc:EncryptedData>
<ds:DigestValue>IT76MbUaxDPmnKuBvmOiQ+kycM4=</ds:DigestValue>
</EncryptedReq>
```

```
</Body>
</KeyAuthnRequestMsg>
<!—Key authentication response message -->
< KeyAuthnResponseMsg>
 <Header MajorVersion="1" MinorVersion="0">
  <InitiatorCookie>D4861B4F</InitiatorCookie>
  <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
  <MessageID>1</MessageID>
  <Flags>0</Flags>
 </Header>
 <Body>
 <EncryptedResp><xenc:EncryptedData><xenc:CipherData>
 <xenc:CipherValue>26Bp .. I++Vn4nACWE=</xenc:CipherValue>
 </xenc:CipherData>
 </xenc:EncryptedData>
 <ds:DigestValue>bEsoQBC1UY+qbluwHHqqBRxRGQ8=</ds:DigestValue>
 </EncryptedResp>
 </Body>
</keaml:KeyAuthnResponseMsg>
```

In Phase II, MPS starts by sending a request that contains an encrypted payload, which has (i) a proposal (the security protocol now is SAML, because the SA is for SAML attribute request/response messages), (ii) a nonce, and (iii) optionally the public key component (e.g., a predefined Diffie-Hellman component). The subject AA returns similar information with the selected SA.

```
<! -- SA2 request  -->
< SA2RequestMsg>
 <Header MajorVersion="1" MinorVersion="0">
 <InitiatorCookie>D4861B4F</InitiatorCookie>
 <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
 <MessageID>2</MessageID>
 <Flags>0</Flags>
 </Header>
 <Body>
  <EncryptedReq><xenc:EncryptedData><xenc:CipherData>
  <xenc:CipherValue>fPFx .. Lx6jXI=</xenc:CipherValue>
  </xenc:CipherData></xenc:EncryptedData>
  <ds:DigestValue>vhdw/cFVtW1srhLX/WffyU8xFx8=</ds:DigestValue>
  </EncryptedReq>
 </Body>
</SA2RequestMsg>
<!-- SA2 response message -->
< SA2ResponseMsg>
 <Header MajorVersion="1" MinorVersion="0">
  <InitiatorCookie>D4861B4F</InitiatorCookie>
  <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
  <MessageID>0</MessageID>
  <Flags>0</Flags>
 </Header>
 <Body>
 <EncryptedResp><xenc:EncryptedData><xenc:CipherData>
  <xenc:CipherValue>C5tl .. 5U4ANP0P</xenc:CipherValue>
```

```
  </xenc:CipherData></xenc:EncryptedData>
 <ds:DigestValue>MnBkj6wkDEktL9JuRiRwlvIDWN0=</ds:DigestValue>
 </EncryptedResp>
</Body>
</ SA2ResponseMsg>
```

Upon completion of Phase II, an SA for SAML 2.0 information exchange (SAMLAttributeQuery and SAMLAttributeAssertion messages) is established. A set of keys is generated for each direction, each of which is the result of a pseudo random function applied to keys derived from Phase I, the nonces of both parties, and optionally, the Diffie-Hellman public components established in Phase II messages. Once the SAML SA has expired, the KEAML SA can be used in another Phase II negotiation to re-establish another SAML SA. When the KEAML SA has expired, MPS and the subject AA have to go through both of the Phase I and II negotiations before they can make more secure SAML attribute exchanges.

4.10 Summary

This chapter describes the design and major constructs of KEAML/KEAML-KE as the framework and protocols to provide the standard method of establishing security communication channels between two application-level entities. In Chapter 5, we will discuss the implementation detail of KEAML/KEAML-KE.

Chapter 5 KEAML IMPLEMENTATION

A simple yet extensible implementation of the KEAML/KEAML-KE engine is presented in this chapter, not only to confirm the feasibility of the protocol, but also to initially verify the performance feature of this protocol.

The implementation is written in Java and run on Windows 2000/XP, or any other platforms that support JRE 1.4.2 or later versions.

5.1 Related Work

RFC2401 IKE [30], or Security Architecture for the Internet Protocol, is an effort to standardize the external aspects of IKE v1 implementation. It recommends having Security Policy Database (SPD) to determine the inbound and outbound IP traffic in a host, and a Security Association Database to manage the SAs themselves.

P.-C. Cheng et al. gave a detailed description of how IKE v1 can be implemented [14]. In detail, architecture for IKE v1 framework/protocol is given in the layer, which contains the security policy management (VPN policy administration tools), the tunnel management to generate the request to the IKE engine, a certificate proxy, and the IKE engine itself.

N. Hallqvist et al. describe Open BSD IKE implementation [24], which is claimed to be quite configurable and can interoperate with other mainstream products.

The implementation of the KEAML/KEAML-KE engine is designed more for demonstration purpose rather than as a real product. The architecture, however, has been laid down so that it contains the modules to handle the states of the entities (i.e. the initiator and the responder), the interfaces to the external module to determine the SA proposals, and the configuration module for the protocol itself.

5.2 Overview

On one hand, the purpose of the implementation is to prove the feasibility of the KEAML/KEAML-KE framework/protocol and conduct an initial performance analysis based on

its implementation. On the other hand, it should be designed so that the implementation is flexible, and can be extended to a practical KEAML/KEAML-KE protocol stack and corresponding message process engine without significant changes.

We first depict the architecture of KEAML/KEAML-KE and then the major classes. Their relationships are discussed in class diagrams.

5.2.1 Architecture

As shown in Figure 5-1, the initiator and the responder share a similar infrasturcture. That is to say, the initiator or the responder, or an entity with a role (i.e. Initiator or Responder) receives the process, and sends out the xml based messages in a similar fashion.

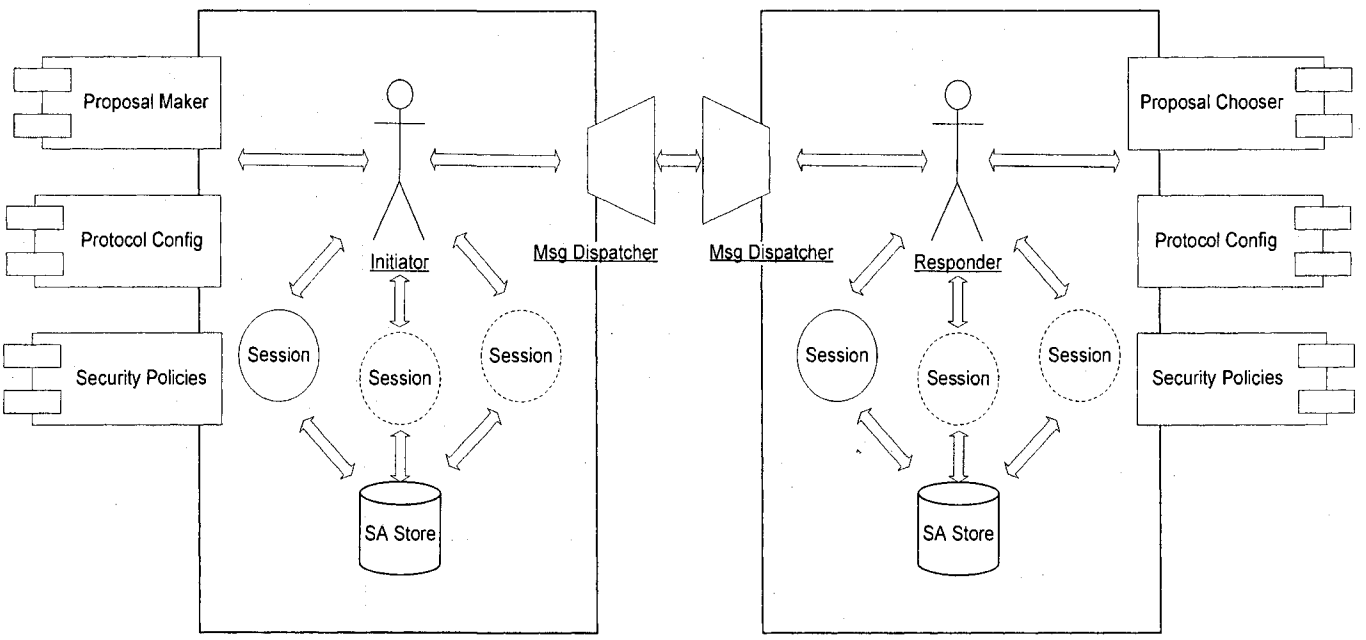

Figure 5-1 KEAML Engine Architecture

An entity (either the initiator or the responder) receives and sends out KEAML-KE messages through message dispatcher instances, which are dependent on an underlying communication channel. Typically, a message dispatcher is bound to an HTTP request/responder handler (e.g., Java servlet) or a SOAP message handler.

The initiator will start a KEAML (either Phase I or Phase II) SA negotiation process by starting a session in which all the information about the status of the negoitation is kept. In contrast, the responder will not start the session until it receives message (3) or the key authentication request from the initiator (please see Section 4.5 for the details). During the negotation, either the initiator or the responder will consult with external components, namely, the proposal maker (for the initiator only), the proposal chooser (for the responder only), the protocol configuration agent, and the security setting (policies) agent, to process the incoming messages and form the outgoing messages. Section 5.3.7 will describe these external components, which are exposed to the KEAML engine by Java interfaces.

Upon completion of the negotiation, the SA is established, along with the calcuated keys are persisted in the SA store, which can be either a protected DB or a file. For simplicity, we implement it only as an XML file.

5.2.2 Class Diagrams

Entity is the base class for both Initiator and Responder, and contains the major logic of processing and sending negotiation messages. Entity can have mutliple session instances; each of them keeps an on-going session's status. Entity can be used as a message listener by extending the interface *IMesssageListener*, which will be notified each time a message comes in. Entity also refers to the message dispatcher instance in order to send outgoing messages.

Session keeps the status of its host entity (*IStateEntity*), the actions to conduct and the exchange type. The actions are determined by the status of the entity, its role, the exchange type of the negotiation, as well as the security setting of the system and any protocol configurations that entity can obtain.

Engine provides the system level services, such as starting up and shuting down the whole system or KEAML engine, exposing the interface to configure forward secrecy intervals.

The classes above belong to *façade* package, which is the core of the engine and exposes the API to use the engine. The other packages under the dotted line in the figure provides other important services, such as bytes concatenation, encrypting/signing (*util* package), and the message syntax checking, messaging forming and verification (*keaml*, *keamlke*, *ds*, and *xenc* packages). The package *sastore* contains the basic functions to support SA loading and storing.

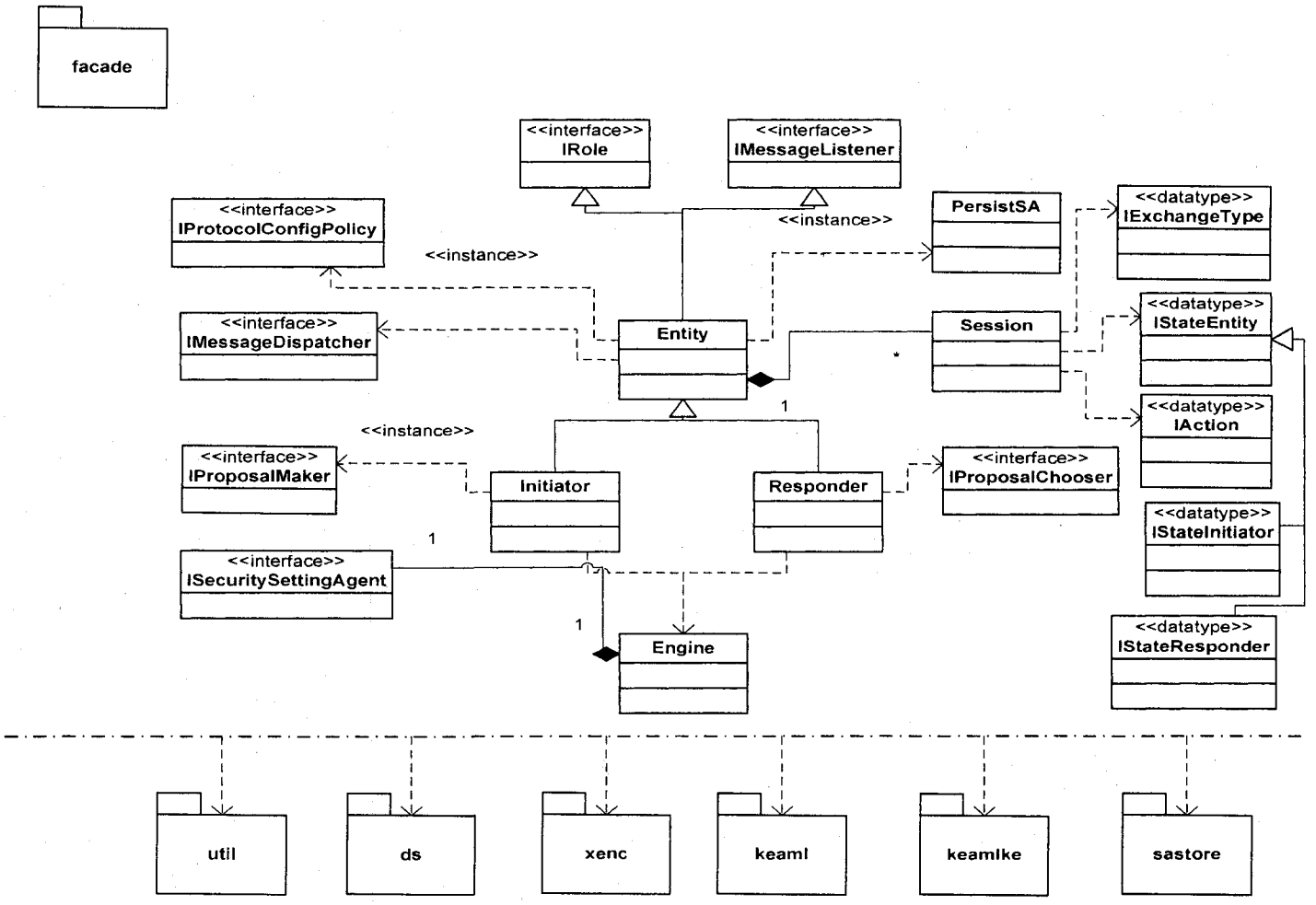

Figure 5-2 KEAML Implementation Class Diagrams

The next section will discuss the engine, entity, initiator, responder, session and the SA store in detail. The external interfaces, JBind, which is used to generate the Java code to handle XML documents and encryption/signature, are also covered in the discussion.

5.3 Introduction to Detailed Design

5.3.1 Engine

The major responsibility of *Engine* is to initialize and uninitialize the engine. It includes

- The configuration and initialization of JBind that provides XML validation, marshal/unmarshal services
- The initialization of the interfaces to the external services required for the engine, such as the security setting, policies, and protocol configuration.
- To uninitialze the engine; and
- To terminate the engine if critical error occurs

Moreover, it provides the local security configuration services for the engine. It includes

- Setting the forward secrecy interval;
- Managing the global secret that is used to generate the responder cookie
- Generating the next message ID upon request and
- An error/information logging service.

5.3.2 Entity

Entity hosts the common logics of the initiator and the responder. These common logics are:

- Initializing/inserting a new session;
- Ending a session; and
- Sequencing the processing of an incoming message

  The sequence can be expressed by the peudo code below:

  ```
  Validate incoming message;
  Process the message and update the state of enitity accordingly;
  Determine the current action;
  Do action;
  ```

- Basic message header validation; and
- Basic error handling

5.3.3 Initiator

Based on the logics of Entity, *Initiator* specifies the concrete logics for the initiator. They are

- Triggering Phase I and Phase II negotiations;
- Generate the initiator cookie for the session and the messages;
- Validating incoming responses. Particulary, the SAKE response's second cookie (the responder cookie must be 0)
- Processing incoming Phase I or Phase II responses from the responder and updating the states accordingly

  In detail, in Phase I, it looks for and verifies the signature in the SAKE response if the ID in a secret form has been sent out; in Phase II, it verifies the authentication payload in the key authentication response message.

- Determining the current action based on the state, the payloads received from the responder, and protocol configuration as well as other security settings.

For example, in the condition that the SAKE request has been sent, and the corresponding SAKE response is received, the initiator will determine the current action as *SEND_AUTHN_REQ* or sending the key authentication request; according to the protocol configuration setting, if any of the certificate requests, prefered ID of the responder, and SA2 payload (for Phase II) should be included, it will set the corresponding action flags accordingly.

- Consulting with the security management system through external interfaces and propose the proper Phase I or Phase II SAs
- Sending Phase I or Phase II requests, or error notification if required;
- Generating the keys for the established SA; and
- Storing the SA upon the completion of the negotiation.

5.3.4 Responder

Similarly, *Responder* specifies the concrete logics for the responder. They are:

- Reacting to the new session only by sending a response, if necessary

  The responder will not initiate a session unless it verifies the initiator is a legimate user, instead of a DoS attacker. The responder will initiate a session only upon the receipt of the key authentication message, or message (3) in Phase I.

  However, in Phase II, it will establish the session upon receiving the SA2 request message from the initiator.

- Generating the responder cookie based on the initiator cookie and the local secret;
- Validating incoming responses. Particulary, it verifies the echoed-back responder cookie by re-calculating it;
- Processing incoming Phase I or Phase II requests from the initiator and updating the states accordingly;

- Determining the current action based on the state, the payloads received from the initiator, and protocol configuration as well as other security settings;

  Particularly, if it has received the initiator's ID in the SAKE message, the responder will includes the signature payload that is calculated on the public component of the D-H key agreement, the SA payload and the two nonces.

- Consulting with the security management system through external interfaces and choosing the Phase I or Phase II SAs that satisfies the security criteria;

- Sending Phase I or Phase II responses, or error notification if required;

- Generating the keys for the established SA; and

- Storing the SA upon completion of the negotiation

5.3.5 Session

As mentioned, *Session* keeps the state information of its host entity, the exchange type of the negotiation, and the current action. Session also contains a list of transient items that is required for key generation and verification. The following is the list of major items kept in a session:

- Key generator used to generate the public DH value and the shared secret;

- Nonces sent and received;

- DH values sent and received;

- Cookies sent and received; and

- SA payloads sent and recevied

5.3.6 SA Store

A simple xml-based SA store is implemented. An SA store is identified by a store name, which in this case will be part of the xml file name.

In a SA store, each Phase I SA is indexed by SPII or the cookie the entity sent out. For example, a Phase I SA is indexed by the initiator's cookie in the initiator's SA store, while in the responder's store the same Phase I SA is indexed by the responder's cookie. Each Phase II SA is indexed in a similar way except that the index itself is SPII concatenated with the message ID of the messages with which the Phase II SA is established. The engine's message ID generation logic will guarantee the uniqueness of the message ID so that it will not make different Phase II SAs to share the same index.

Moreover, each parent SA (Phase I SA) has refereneces to its child SAs, while a child SA has a reference to its parent SA. For the detailed definition of SA store, please refer to Appendix C, which gives the xml schema of SA store.

5.3.7 External Interfaces

Several external interfaces are defined in the implementation, as follows:

- *Proposal maker*

  The initiator uses this to consult with the security management system for the proper SA proposals. The proposal maker is also used by the responder to make a proposal if it failed to choose one from the initiator's SA proposals.

- *Proposal chooser*

  The responder uses this service to choose the prefered SA from a list of SA proposals.

- *Protocol Configuration Policy*

  This interface is used by the entity to get the configuration policy for the protocol. For example, the initiator can ask if it should send an ecrypted ID to a given responder.

- *Security Setting Agent*

  The security setting agent is a major interface used by the entity to get the answers for most of security setting related questions, such as the public key of a given reponder, the secret used to generate the cookie, the private key for a given usage and algorithm etc.

5.3.8 Parsing and Validation of Messages – Jbind

Since the handling of KEAML/KEAML-KE syntax is not trivial, JBind, a third party XML binding solution is used in the implementation. With JBind [59], the Java code to validate, marshal, unmarshal KEAML/KEAML-KE messages and SA store xml files is generated. After generation, some business logics for the messages and SA store handling can be added on top of the generated code.

However, for each type defined in an XML schema, two classes and two interfaces are generated. They are data interface, behavior interface, behavior class, and data class. On one hand, this makes the infrastructure straightforward and simple, but on the other hand, it results in a large number of Java files being generated. For KEAML, about 450 files are generated. This potentially becomes a burden of the performance.

5.3.9 Encryption/Digital Signature

SUN Java Cryptography Extension (JCE) [55] is wrapped in *util* package to provide the encryption and digital signature services. Please see the reference for details.

5.4 Statistics on Performance

An initial performance analysis has been done for KEAML/KEAML-KE messaging in both the size and processing time perspectives.

5.4.1 Message Size

A size comparison between three sets of KEAML/KEAML-KE messages is shown in Table 5-1. The unit of the size is byte.

	SAKE Req	SAKE Resp	Key Authn Req	Key Authn Resp	SA2 Req	SA2 Resp
Instance 1	2406	2255	6507	2056	2915	2673
Instance 2	3411	2255	7512	2056	4315	2673
Instance 3	4418	2253	8517	2056	5715	2673

Table 5-1 KEAML Message Sizes (Unit: Byte)

In the first set or instance 1, the SAKE request and SA2 request both contain one SA proposal, while the two requests in instance 2 contain two and in instance 3 there are three SA proposals. This reflects in the message size for the SAKE request and the SA2 request. It is observed that for all request messages, there is approximatley a one-kilobyte size increment for one additional SA proposal. However, the size of the response messages is the same, because the response only chooses only one SA proposal.

It is noted that the key authentication request message is the largest, because it echoes back the payloads in mesasge (1) and (2), and the encrypted and authenticated payload that contains at least the authentication payload. In detail, when the number of SA proposals reaches to three, the size of key authentication request is close to ten kilobytes, which might be issue for a mobile device. Nevertheless, the message size can be reduced by compression, which will be briefly discussed in Section 5.4.3. Furthermore, in practice, it is reasonable to limit the number of SA proposals to fewer than four.

5.4.2 Processing Overhead

The processing time for instances 1, 2 and 3 are depicted in Table 5-2, Table 5-3, and Table 5-4. The time is measured in milliseconds.

	1st run	2nd run	3rd run	4th run	5th run	Avg
SAKE Req	781	516	500	610	563	594
SAKE Resp	125	93	94	110	94	103.2
Key Authn Req	219	234	188	265	235	190.2
Key Authn Resp	125	188	109	172	125	143.8
Process Key Authn Resp	31	15	0	16	16	15.6
SA2 Req	16	32	31	15	15	21.8
SA2 Resp	93	62	31	94	125	81
Process SA2 Resp	16	16	16	0	32	16

Table 5-2 Instance 1 Processing Time (Unit: ms)

	1st run	2nd run	3rd run	4th run	5th run	Avg
SAKE Req	485	765	766	469	468	590.6
SAKE Resp	78	125	125	93	78	99.8
Key Authn Req	187	235	234	172	203	206.2
Key Authn Resp	125	109	157	157	156	140.8
Process Key Authn Resp	16	16	15	31	16	18.8
SA2 Req	16	47	16	15	16	22
SA2 Resp	47	125	47	47	78	68.8
Process SA2 Resp	15	31	0	0	15	12.2

Table 5-3 Instance 2 Processing Time (Unit: ms)

	1st run	2nd run	3rd run	4th run	5th run	Avg
SAKE Req	516	656	875	500	531	615.6
SAKE Resp	125	125	140	125	109	124.8
Key Authn Req	265	234	250	297	250	259.2
Key Authn Resp	140	188	187	125	250	178
Process Key Authn Resp	15	16	16	15	15	15.4
SA2 Req	16	31	47	32	15	28.2
SA2 Resp	78	78	62	78	63	71.8
Process SA2 Resp	16	16	16	31	16	15.8

Table 5-4 Instance 3 Processing Time (Unit: ms)

Surprisingly, despite the difference in size of the three sets of messages, the corresponding processing time is very similar. Furthermore, the following points are observed:

- The SAKE request message requires the most amount of processing time. This is likely because of the system initialization time, and the Diffie-Hellman keying material generation.

- The processing for key authentication response and SA2 response requires the least amount of time. This indicates SA session key calculation or generation (only for SA2 response processing) and persisting the established SA do not consume significant CPU time.

- The processing time for both SA2 request and response messages is low, probably because the process does not involve heavy calculations. In the example provided, no extra DH public components are generated. The processing time for the SA2 response is

higher than the SA2 request, possibly because the responder needs to read the request message first, validate it, choose one of the proposals and form the response.

- In the Phase I exchange, the processing time on the responder side is lower than that on the initiator side; this is probably due to the asymmetric protocol design, in which the response messages are less complex than the request messages.

A comparison of processing time with another similar protocol cannot be conducted, since there is no other standard layer 7 key exchange and authentication protocol. However, the maximum processing time around half second for a layer 7 application is quite satisfactory, considering the several-second response time is typical and acceptable in most of applications, such as internet browsing.

Although the overall processing time is not significant, it is worth analyzing the distribution of processing time for each step, such as marshalling/unmarshalling XML messages, their validation and D-H keying material generation, cookie generation, nonce generation etc. This will be attempted in the future.

5.4.3 Potential Improvement

It is expected that the performance of XML based protocol is not as good as those that are binary based. However, a number of studies have been done in this area. In Particular, C. Adams and the author have conducted research on XACML document compression by using ASN.1 [47]. Detailed comparison results on XACML documents are presented. The study concludes that wbXML plus Gzip would be the best solution probably because the tag names in XACML are long and the replacement, which is used in wbXML, has much less processing overhead than complex encoding in ASN.1. One of the future works would be to verify how much the compression of wbXML would contribute to the performance of KEAML/KEAML-KE protocols.

5.5 Summary

In summary, a simple but complete implementation of KEAML/KEAML-KE framework and protocols are presented in this chapter. First, some related works on IKE v1 are introduced, then the overivew of the architecture of KEAML/KEAML-KE processing engine and some

implementation details are discussed. This is followed by an initial performance analysis on both message size and processing time. The next chapter will provide the conclusion of this thesis.

Chapter 6 CONCLUSIONS

In Section 1.2 in this thesis, we first state several access-control problems in a mobile environment. They are (1) Unrecognizable Subject ID and Indeterminacy of AC Model; (2) Locate and Apply Applicable Policies During Authorization; (3) Enforce Regulations/Laws and Security SLAs; (4) Protect the Requester's Confidential Data; (5) Privacy Threats in Distributed System; (6) Applicability of Reputation Data; and (7) Set up Secured Communication Between Parties.

These problems are addressed by introducing subject ID mapping, autonomous registration, the reverse authorization process, MPS, privacy proxy server and the concept of private reputation server into an Attribute-based Access Control system.

First, the subject ID mapping service provided by MPS allows the requester's subject ID to be recognizable (problem 1). For the requester to request a service, it just needs to send over the subject ID to the local security domain, which can obtain the attributes with the specified subject ID and use them to render the authorization decision.

Second, the security domain can get all applicable security and privacy policies, laws and regulations from distributed stores through MPS, and employ them under the governing of the configurable meta policies to render the final authorization decision (problem 2 and 3).

Third, in the situation where the requester is allowed to get the service, it can choose to trigger a reverse authorization procedure to guarantee that the security of its data will not be compromised by the service provider (problem 4).

Fourth, applying the privacy proxy server makes privacy protection issues more solvable, since it frees individual service prioviders from the burden of implementing complex privacy protection systems, and the service consumers do not have to deal with a number of service providers in order to track the usage of their privacy information (problem 5).

Fifth, the concept of private reputation server makes the usage of repuation data more appropriate.

Sixth and last, a standardized application level key exchange and authentication protocol, KEAML/KEAML-KE, is proposed to solve the secure communication issue between an MPS in a security domain and a number of subject AAs.

The solutions for the first six problems are described in Chapter 3. After the detailed description of the proposed architecture for the access control in a mobile environment, some intial discussion about the validity of the proposed architecture occurs in Section 3.10, from both security and feasibility perspectives. The section examines new logic components such as (a) MPS; (b) reverse authentication procedure; (c) the assumption of the interoperability/expressibility of XML; and (d) the assumptions about the equivalency of subject ID and resource ID. The discussion indicates that the system is logically secure and feasible. Yet the complete analysis cannot be accomplished before determining further design and implementation details.

The implementation of the architecture (including MPS, autonomous registration, reverse authorization etc.) will be future work. Along with this, more details in the proposed architecture need to be determined, such as:

- The best practices in implementing security SLAs in XACML;
- The best practices in implementing meta policies in XACML; and
- Introducing dynamic attributes into the infrastructure.

Such issues constitute some possibilities for further work in this area.

The solution of the last problem is discussed in Chapter 4. By combining IKE v1/v2 and JFK in the application layer (Layer 7), a new XML-based key exchange and authentication protocol, KEAML/KEAML-KE, is established. It has all the security features that IKE and JFK have in Layer 3 and makes secure communications between two application-level entities easy and extensible.

Directions for future research into KEAML/KEAML-KE include (a) incorporating the feature of extensible authentication (introduced in IKE 2.0) into KEAML/KEAML-KE; (b) further examining performance and bandwidth considerations; (c) validating the system in terms of

security and feasibility; and (d) proposing KEAML/KEAML-KE as a standard application layer key exchange and authentication protocol.

REFERENCES

[1] M. Abadi, R. Needham. *Prudent Engineering Practice for Cryptographic Protocols,* Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy, Nov 1, 1994

[2] W. Aiello, S. M. Bellovin, et al. *Efficient, DoS-resistent, Secure Key Exchange for Internet Protocols,* pages 48 – -58, Proceedings of the 9th ACM Conference on Computer and Communications Security, 2002

[3] A. Alshamsi, T. Saito. *A Technical Comparison of IPSec and SSL,* pages 395 – -398, Proceedings of the 19th International Conference on Advanced Information Networking and Applications – Volume 2, 2005

[4] R. J. Anderson, R. M. Needham. *Robustness Principles for Public Key Protocols,* pages 236 – -247, Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, 1995

[5] S. Beaulieu, R. Pereira. *Extended Authentication within IKE (XAUTH).* Internet Draft, Oct, 2000, available at http://tools.ietf.org/html/draft-beaulieu-ike-xauth-00

[6] M. Bellare, R. Canetti, H. Krawczyk. *A Modular Approach to the Design and "Analysis of Authentication and Key Exchange Protocols,* pages 419 – -428, Proceedings of the 30th Annual Symposium on the Theory of Computing, 1998

[7] S. M. Bellovin. *Problem Areas for the IP Security Protocols,* , Proceedings of the 6th Usenix UNIX Security Symposium, July 22 – 25, 1996, San Jose, CA

[8] K. Bhargavan, C. Fournet, A. D. Gordon, S. Tse, *Verified Interoperable Implementations of Security Procotols,* Computer Security Foundations Workshop, 2006. 19TH IEEE

[9] S. Blake-Wilson, D. Johnson, A. Menezes. *Key Agreement Protocols and Their Security Analysis,* page 30 – 45, Proceedings of the 6th IMA International Conference on Cryptography and Coding, 1997

[10] J. Bohli, M. Vasco, R. Steinwandt. *Secure Group Key Establishment Revisited*, 2005, available at http://eprint.iacr.org/complete/

[11] M. Burrows, M. Abadi, R. Needham. *A Logic of Authentication,* page 18 – 36, ACM Transactions on Computer Systems (TOCS), Volume 8, Issue 1, Feb 1990

[12] R. Canetti, H. Krawczyk. *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels,* page 453 – 474, Proceedings of the International Conference on the theory and Application of Cryptographic Techniques: Advances in Cryptology, 2001

[13] R. Canetti, H. Krawczyk. *Security Analysis of IKE's Signature-Based Key-Exchange Protocol,* page 143 – 161, Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology, 2002

[14] P.-C. Cheng, Thomas J. Watson. *An Architecture for the Internet Key Exchange Protocol.* IBM Systems Journal, page 721 – 746, Volume 40, Issue 3, April. 2003

[15] Z. Cheng, R. Comley. *Attacks on An ISO/IEC 11770-2 Key Establishment Protocol*, Sept 23, 2004, available at http://eprint.iacr.org/2004/

[16] J. Clark. *Attacking Authentication Protocols*, available at http://www-users.cs.york.ac.uk/~jac/papers/newHISJ.ps, Mar. 1996

[17] W. Diffie, P. Van Oorschot, and M. Wiener. *Authentication and Authenticated Key Exchanges*. p 107 – 125, Designs, Codes and Cryptography, vol. 2, Kluwer Academic Publishers, June, 1992

[18] R. Dutta, R. Barua. *Overview of Key Agreement Protocols,* 2005, available at http://eprint.iacr.org/2005/

[19] Federal Trade Commission (FTC). *Fair Information Practice Principles*, http://www.ftc.gov/reports/privacy3/fairinfo.htm

[20] A. O. Freier, P. Karlton, P. C. Kocher. *SSL 3.0 Specification,* March 1996, available at http://wp.netscape.com/eng/ssl3/

[21] A. D. Gordon, A. Jeffrey. *Authenticity by Typing for Security Protocols,* page 451 – 519, Journal of Computer Security, Volume 11, Issue 4, Jan, 2004

[22] S. Gritzalis, D. Spinellis. *Cryptographic Protocols over Open Distributed Systems: A Taxonomy of Flaws and Related Protocol Analysis Tools,* page 123 - 137, 16th International Conference on Computer Safety, Reliability, and Security: SAFECOMP, Sept 1997, York, UK

[23] P. Gupta, V. Shmatikov. *Towards Computationally Sound Symbolic Analysis of Key Exchange Protocols*, page 23 – 32, Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering, 2005

[24] N. Hallqvist, A. D. Keromytis. *Implementing Internet Key Exchange (IKE),* OpenBSD, available at http://www.cis.upenn.edu/~dsl/STRONGMAN/Papers/ikepaper.ps.gz

[25] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE).* Request for Comments RFC 2409, Nov. 1998, available at http://www.ietf.org/rfc/rfc2409.txt

[26] P. Karn, W. Simpson. *Photuris: Session-Key Management Protocol*, Request for Comments RFC 2522, Mar. 1999, www.ietf.org/rfc/rfc2522.txt

[27] C. Kaufman, Ed. *Internet Key Exchange (IKE v2) Protocol*, Request for Comments RFC 4306, Dec. 2005, www.ietf.org/rfc/rfc4306.txt

[28] G. S. Kc, P. A. Karger. *Preventing Security and Privacy Attacks on Machine Readable Travel Documents (MRTDs),* 2005, available at http://eprint.iacr.org/2005/

[29] J. Kelsey, B. Schneier, D. Wagner. *Protocol Interactions and the Chosen Protocol Attack,* page 90 – 104, Proceedings of the 5th International Workshop on Security Protocols, 1997

[30] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol.* Request for Comments RFC 2401, Nov. 1998, available at http://www.ietf.org/rfc/rfc2401.txt

[31] H. Krawczyk. *SKEME: A Versatile Secure Key Exchange Mechanism for Internet,* page 114, Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96), 1996

[32] G. Lowe. *Some New Attacks upon Security Protocols,* page 162, Proceedings of the 9th IEEE Computer Security Foundations Workshop, 1996

[33] M. Malunis. *Survey on Security Requirements and Models for Group Key Exchange*, Nov 2006, available at http://eprint.iacr.org/2006/

[34] D. Maughan, M, Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP).* Request for Comments RFC 2408, Nov. 1998, available at http://www.ietf.org/rfc/rfc2408.txt

[35] Catherine Meadows. *What Makes a Cryptographic Protocol Secure? The Evolution of Requirements Specification in Formal Cryptographic Protocol Analysis.* Proceedings of ESOP 03, Springer-Verlog, April. 2003

[36] C. Meadows. *Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer,* page 216 – 231, Security and Privacy, 1999, Proceedings of the 1999 IEEE Symposium, Oakland, CA, USA, 1999

[37] OASIS. *eXensible Access Control Markup Language (XACML) Version 2.0*, http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-02.pdf.

[38] OASIS. *XACML Profile for Role Based Access Control (RBAC)*, http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile- -01.pdf.

[39] OASIS Security Services TC. *Security Assertion Markup Language (SAML) v2.0.* March. 2005, available at http://www.oasis-open.org/specs/index.php#spmlv2.0

[40] H. Orman. *The Oakley Key Determination Protocol*, Request for Comments RFC 2412, Nov. 1998, www.ietf.org/rfc/rfc2412.txt

[41] O. Pereira. *Some Attacks Upon Authenticated Group Key Agreement Protocols*, page 555 – 580, Journal of Computer Security, Volume 11, Issue 4, 2004

[42] R. J. Perlman, C. Kaufman. *Analysis of the IPSec Key Exchange Standard,* page 150 – 156, Proceedings of the 10th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001

[43] R. Perlman, C. Kaufman. *Key Exchange in IPSec: Analysis of IKE,* page 50 – 56, IEEE Internet Computing, Volume 4, Issue 6, Nov/Dec 2000

[44] D. Piper. *The Internet IP Security Domain of Interpretation for ISAKMP.* Request for Comments RFC 2407, Nov. 1998, available at http://www.ietf.org/rfc/rfc2407.txt

[45] Xuebing Qing and Carlisle Adams. *KEAML – Key Exchange and Authentication Markup Language.* p 1539 – 1543, Proceedings of the 19th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'06), May 7 – 10, 2006, Ottawa, Ontario, Canada

[46] Xuebing Qing and Carlisle Adams. *XACML-Based Policy Driven Access Control for Mobile Environments.* p 1548 – 1551, Proceedings of the 19th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'06), May 7 – 10, 2006, Ottawa, Ontario, Canada

[47] Xuebing Qing and Carlisle Adams. *A Comparison of Compression Techniques for XML-based Security Policies in Mobile Computing Environments.* Ottawa New Challenges for Access Control Workshop, 27 April, 2005. Available at http://lotos.site.uottawa.ca/ncac05/fisler_18500059.ppt

[48] R. Ramanujam, S. P. Suresh. *Deciding Knowledge Properties of Security Protocols,* Proceedings of TARK X (2005), July 10 – 12, 2005, Singapore

[49] E. Rescorla. *Diffie-Hellman Key Agreement Method,* Request for Comments RFC 2631, Jun. 1999, www.ietf.org/rfc/rfc2631.txt

[50] S. Schneider, *Verifying Authentication Protocols in CSP,* page 741 – 758, IEEE Transactions on Software Engineering, Volume 24, Issue 9, Sept 1998

[51] W. A. Simpson. *IKE/ISAKMP Considered Harmful,* Usenix ;login, Dec 1999, Volume 24, Number 6

[52] M. Steiner, G, Tsudik, M. Waidner. *Key Agreement in Dynamic Peer Groups,* page 769 – 780, Parallel and Distributed Systems, IEEE Transactions, Volume 11, Issue 8, Aug 2000

[53] M. A. Strangio. *On the Resilience of Key Agreement Protocols to Key Compromise Impersonation,* Proceedings of the 3rd European PKI Workshop, EuroPKI06, Jul 2006

[54] SUN. *Sun Identity Management: Identity Standards - OASIS*, http://www.sun.com/software/products/identity/standards/index.xml.

[55] SUN. *Java Cryptography Extension (JCE),* available at http://java.sun.com/products/jce/

[56] Y. P. Swami. *Design of IPSec and IKE Version 1 and 2,* a living document, available at http://www.yogesh-swami.org

[57] M. Thomas, J. Vilhuber. *Kerberized Internet Negotiation of Keys (KINK).* Internet Draft, Jan 21, 2003, available at http://www.ietf.org/ids.by.wg/kink.html

[58] United States Department of Defence. *Trusted Computer System Evaluation Criteria (DOD-5200.28-STD), or the Orange Book,* published in 1983, available at http://www.dynamoo.com/orange/ or http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html

[59] S. Wachter. *JBind 1.2 A Java-XML Data Binding Framework,* available at http://www.jbind.org

[60] World Wide Web Consortium. *XML Key Management Specification (XKMS),* March 2001, available at http://www.w3.org/TR/xkms/

[61] World Wide Web Consortium. *XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002,* Dec 2002, available at http://www.w3.org/TR/xmlenc-core/

[62] World Wide Web Consortium. *XML Signature Syntax and Processing, W3C Recommendation 12 Februrary 2002,* Feb 2002, available at http://www.w3.org/TR/xmldsig-core/

[63] C. Xu, G. Kedem, F. Gong. *Categorizing Attacks on Cryptographic Protocols Based on Intruders' Objectives and Roles*, 2000, available at http://citeseer.ist.psu.edu/xu00categorizing.html

[64] Eric Yuan, Jin Tong. *Attribute Based Access Control – A New Access Control Approach for Service Oriented Architectures (SOA)*, NCAC (New Challenges for Access Control) 2005. Ottawa, ON, Canada

APPENDIX A KEAML SCHEMA

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:keaml="urn:names:tc:KEAML:1.0"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
targetNamespace="urn:names:tc:KEAML:1.0" elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.0">
        <import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-
schema.xsd"/>
        <import namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-
schema.xsd"/>
        <annotation>
                <documentation>
      Document identifier: keaml-schema-1.0
    </documentation>
        </annotation>
        <simpleType name="CookieType">
                <restriction base="hexBinary"/>
        </simpleType>
        <!-- HeaderFlagValue -->
        <!-- bit 0: information payload contained; bit 1: error; bit 2: SA proposed; -->
        <simpleType name="HeaderFlagValue">
                <restriction base="byte"/>
        </simpleType>
        <!-- SituationType -->
        <complexType name="SituationType" abstract="true"/>
        <element name="Situation" type="keaml:SituationType"/>
        <!-- ProtocolIDType -->
        <simpleType name="ProtocolIDType">
                <restriction base="anyURI"/>
        </simpleType>
        <!-- ProtocolID -->
        <element name="ProtocolID" type="keaml:ProtocolIDType"/>
        <!-- SPIType -->
        <simpleType name="SPIType">
                <restriction base="hexBinary"/>
        </simpleType>
        <!-- SPI -->
        <element name="SPI" type="keaml:SPIType"/>
        <!-- Used to identify the transfrom type -->
        <simpleType name="TransformTypeType">
                <restriction base="anyURI"/>
        </simpleType>
        <!-- TransformIDType -->
        <!-- Used to identify the transfrom -->
        <simpleType name="TransformIDType">
                <restriction base="anyURI"/>
        </simpleType>
        <!-- DataAttributeType-->
        <!-- The generic DataAttributeType which can be overwritten -->
        <complexType name="DataAttributeType">
                <sequence>
                        <any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
```

```
            </sequence>
            <anyAttribute namespace="##any" processContents="lax"/>
        </complexType>
        <!-- global DataAttribute -->
        <element name="DataAttribute" type="keaml:DataAttributeType"/>
        <!-- IdentificationDataType -->
        <complexType name="IdentificationDataType">
            <sequence>
                <element name="IDType" type="anyURI"/>
                <element name="ID" type="anyURI"/>
            </sequence>
        </complexType>
        <element name="IdentificationData" type="keaml:IdentificationDataType"/>
        <!-- AnchorType -->
        <complexType name="AnchorType">
            <sequence>
                <element name="IssuerName" type="string"/>
                <choice>
                    <element name="KeyName" type="string"/>
                    <sequence>
                        <element ref="ds:DigestMethod" minOccurs="0"/>
                        <element ref="ds:DigestValue"/>
                    </sequence>
                </choice>
            </sequence>
        </complexType>
        <!-- NotificationCategoryType -->
        <simpleType name="NotificationCategoryType">
            <restriction base="anyURI">
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
payload-type"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
cookie"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
major-version"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
minor-version"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
exchange-type"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
flags"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
spi"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:no-
proposal-chosen"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-ke-
payload"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:attributes-
not-supported"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
syntax"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-id-
information"/>
                <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:invalid-
certificate"/>
```

```
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-
type:authentication-failed"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:address-
notification"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:certificate-
unavailable"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:required-
payload-missing"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-
type:unexpected-msg-received"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:notification-type:session-
terminated-upon-error-received"/>
                </restriction>
        </simpleType>
        <!-- NotficationCategory -->
        <element name="NotificationCategory" type="keaml:NotificationCategoryType"/>
        <!-- Message Components and Payloads etc. -->
        <!-- MessageHeaderType -->
        <complexType name="MessageHeaderType">
                <sequence>
                        <element name="InitiatorCookie" type="keaml:CookieType"/>
                        <element name="ResponderCookie" type="keaml:CookieType"/>
                        <element name="MessageID" type="integer" minOccurs="0"/>
                        <!-- bit 0: error ; bit 1: warning; bit 2: info bit 3: propose new SAs -->
                        <element name="Flags" type="keaml:HeaderFlagValue"/>
                </sequence>
                <attribute name="MajorVersion" type="unsignedByte" use="required"/>
                <attribute name="MinorVersion" type="unsignedByte" use="required"/>
        </complexType>
        <!-- MessageHeader -->
        <element name="Header" type="keaml:MessageHeaderType"/>
        <!-- PayloadType -->
        <complexType name="PayloadType" abstract="true">
                <sequence>
                        <any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
                <anyAttribute namespace="##any" processContents="lax"/>
        </complexType>
        <!-- Payload -->
        <element name="Payload" type="keaml:PayloadType"/>
        <!-- EncryptedPayloadType is a payload that contains encrypted payload -->
        <complexType name="EncryptedPayloadType" abstract="true">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element ref="xenc:EncryptedData"/>
                                        <element ref="ds:DigestValue" minOccurs="0"/>
                                </sequence>
                                <attribute name="Type" type="anyURI" use="optional"/>
                                <attribute name="Mapping" type="boolean" use="optional"/>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- MessageBodyType -->
        <complexType name="MessageBodyAbstractType" abstract="true">
```

```
                <sequence>
                        <any namespace="##any" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
                </sequence>
        </complexType>
        <!-- MessageBody -->
        <element name="Body" type="keaml:MessageBodyAbstractType"/>
        <!-- KEPayloadType -->
        <complexType name="KEPayloadType" abstract="true">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <any namespace="##any" maxOccurs="unbounded"/>
                                </sequence>
                                <anyAttribute namespace="##any" processContents="lax"/>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- AnchorPayloadType -->
        <complexType name="AnchorPayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="Anchor" type="keaml:AnchorType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- PubKeyPayloadType -->
        <complexType name="PubKeyPayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element ref="ds:KeyValue"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SAPayloadType -->
        <complexType name="SAPayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="Proposal"
type="keaml:ProposalPayloadType" minOccurs="0" maxOccurs="unbounded"/>
                                        <element ref="keaml:Situation" minOccurs="0"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SAPayload -->
        <element name="SA" type="keaml:SAPayloadType"/>
        <!-- ProposalPayloadType -->
        <complexType name="ProposalPayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
```

```
                                        <sequence>
                                                <element ref="keaml:ProtocolID"/>
                                                <element name="Transform"
type="keaml:TransformPayloadType" maxOccurs="unbounded"/>
                                        </sequence>
                                        <attribute name="number" type="integer"/>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- ProposalPayload -->
                <element name="Proposal" type="keaml:ProposalPayloadType"/>
                <!-- TransformPayloadType -->
                <complexType name="TransformPayloadType">
                        <complexContent>
                                <restriction base="keaml:PayloadType">
                                        <sequence>
                                                <element name="TransformType"
type="keaml:TransformTypeType"/>
                                                <element name="TransformID"
type="keaml:TransformIDType"/>
                                                <element name="DataAttribute"
type="keaml:DataAttributeType" maxOccurs="unbounded"/>
                                        </sequence>
                                        <attribute name="number" type="integer" use="optional"/>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- TransformPayload -->
                <element name="Transform" type="keaml:TransformPayloadType"/>
                <!-- IdentificationPayloadType -->
                <complexType name="IdentificationPayloadType">
                        <complexContent>
                                <restriction base="keaml:PayloadType">
                                        <sequence>
                                                <element name="IdentificationData"
type="keaml:IdentificationDataType"/>
                                        </sequence>
                                        <anyAttribute namespace="##any" processContents="lax"/>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- IdentificationPayload -->
                <element name="IdentificationPayload" type="keaml:IdentificationPayloadType"/>
                <!-- AuthnPayloadType -->
                <complexType name="AuthnPayloadType" abstract="true">
                        <complexContent>
                                <restriction base="keaml:PayloadType">
                                        <sequence>
                                                <any namespace="##any" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                                        </sequence>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- CertificatePayloadType -->
                <complexType name="CertificatePayloadType">
```

```
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <choice>
                                                <element name="X509Data"
type="ds:X509DataType"/>
                                                <element name="PGPData"
type="ds:PGPDataType"/>
                                                <element name="SPKIData"
type="ds:SPKIDataType"/>
                                                <any namespace="##other" processContents="lax"
minOccurs="0"/>
                                        </choice>
                                </sequence>
                                <anyAttribute namespace="##any" processContents="lax"/>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- CertificateRequestPayloadType -->
        <complexType name="CertificateRequestPayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="CertificateType" type="anyURI"/>
                                        <choice>
                                                <element name="X509IssuerSerial"
type="ds:X509IssuerSerialType" maxOccurs="unbounded"/>
                                                <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                                        </choice>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SignaturePayloadType -->
        <complexType name="SignaturePayloadType">
                <complexContent>
                        <restriction base="keaml:AuthnPayloadType">
                                <sequence>
                                        <element ref="ds:SignatureValue"/>
                                        <element ref="ds:CanonicalizationMethod" minOccurs="0"/>
                                        <element ref="ds:SignatureMethod"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <complexType name="NoncePayloadType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="Nonce" type="hexBinary"/>
                                </sequence>
                                <anyAttribute namespace="##any" processContents="lax"/>
                        </restriction>
                </complexContent>
        </complexType>
```

```
        <!-- EncryptedPayloadType is a payload that contains encrypted payload -->
        <!-- If DigestValue does not exist, then it is only encrypted -->
        <complexType name="EncryptedIDPayloadType">
                <complexContent>
                        <restriction base="keaml:EncryptedPayloadType">
                                <sequence>
                                        <element ref="xenc:EncryptedData"/>
                                        <element ref="ds:DigestValue" minOccurs="0"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:identity"/>
                                <attribute name="Mapping" type="boolean" use="optional"/>
                        </restriction>
                </complexContent>
        </complexType>
        <complexType name="EncryptedAuthnReqPayloadsType">
                <complexContent>
                        <restriction base="keaml:EncryptedPayloadType">
                                <sequence>
                                        <element ref="xenc:EncryptedData"/>
                                        <element ref="ds:DigestValue"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:authn-request"/>
                        </restriction>
                </complexContent>
        </complexType>
        <complexType name="AuthnReqPayloadsType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="CERT"
type="keaml:CertificatePayloadType" minOccurs="0"/>
                                        <element name="CERTREQ"
type="keaml:CertificateRequestPayloadType" minOccurs="0"/>
                                        <element name="IDr"
type="keaml:IdentificationPayloadType" minOccurs="0"/>
                                        <element name="AUTH" type="keaml:AuthnPayloadType"/>
                                        <element name="SAi2" type="keaml:SAPayloadType"
minOccurs="0"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:authn-request"/>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="AuthnReqPayloads" type="keaml:AuthnReqPayloadsType"/>
        <complexType name="EncryptedAuthnRespPayloadsType">
                <complexContent>
                        <restriction base="keaml:EncryptedPayloadType">
                                <sequence>
                                        <element ref="xenc:EncryptedData"/>
                                        <element ref="ds:DigestValue"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:authn-response"/>
```

```
                    </restriction>
                </complexContent>
            </complexType>
            <complexType name="AuthnRespPayloadsType">
                <complexContent>
                    <restriction base="keaml:PayloadType">
                        <sequence>
                            <element name="IDr"
type="keaml:IdentificationPayloadType"/>
                            <element name="CERT"
type="keaml:CertificatePayloadType" minOccurs="0"/>
                            <element name="AUTH" type="keaml:AuthnPayloadType"/>
                            <element name="SAr2" type="keaml:SAPayloadType"
minOccurs="0"/>
                        </sequence>
                        <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:authn-response"/>
                    </restriction>
                </complexContent>
            </complexType>
            <element name="AuthnRespPayloads" type="keaml:AuthnRespPayloadsType"/>
            <!-- EncryptedNotificationType -->
            <complexType name="EncryptedNotificationPayloadType">
                <complexContent>
                    <restriction base="keaml:EncryptedPayloadType">
                        <sequence>
                            <element ref="xenc:EncryptedData"/>
                            <element ref="ds:DigestValue"/>
                        </sequence>
                        <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:notification"/>
                    </restriction>
                </complexContent>
            </complexType>
            <!-- NotificationPayload -->
            <complexType name="NotificationPayloadType">
                <complexContent>
                    <restriction base="keaml:PayloadType">
                        <sequence>
                            <element ref="keaml:SPI"/>
                            <element ref="keaml:ProtocolID"/>
                            <element ref="keaml:NotificationCategory"/>
                            <element name="NotificationData" type="string"/>
                        </sequence>
                    </restriction>
                </complexContent>
            </complexType>
            <element name="NotificationPayload" type="keaml:NotificationPayloadType"/>
            <!-- ExchangeMessageAbstractType -->
            <complexType name="ExchangeMessageAbstractType" abstract="true">
                <sequence>
                    <element name="Header" type="keaml:MessageHeaderType"/>
                    <element name="Body" type="keaml:MessageBodyAbstractType"/>
                </sequence>
            </complexType>
            <!-- ExchangeMessageAbstract -->
```

```
        <element name="ExchangeMessageAbstract" type="keaml:ExchangeMessageAbstractType"/>
        <!-- SAKERequestMsgType -->
        <complexType name="SAKERequestMsgType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body"
type="keaml:SAKERequestMsgBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SAKERequestMsg" type="keaml:SAKERequestMsgType"/>
        <!-- SAKERequestMsgBodyType -->
        <complexType name="SAKERequestMsgBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="SA" type="keaml:SAPayloadType"/>
                                        <element name="KE" type="keaml:KEPayloadType"/>
                                        <element name="Nonce" type="keaml:NoncePayloadType"/>
                                        <!-- Initiator's ID, no MAC, mapping could be true -->
                                        <element name="EncryptedID"
type="keaml:EncryptedIDPayloadType" minOccurs="0"/>
                                        <element name="Anchor" type="keaml:AnchorPayloadType"
minOccurs="0"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SAKEResponseMsgType -->
        <complexType name="SAKEResponseMsgType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body"
type="keaml:SAKEResponseMsgBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SAKEResponseMsg" type="keaml:SAKEResponseMsgType"/>
        <!-- SAKEResponseMsgBodyType -->
        <complexType name="SAKEResponseMsgBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="SA" type="keaml:SAPayloadType"/>
                                        <element name="KE" type="keaml:KEPayloadType"/>
                                        <element name="Nonce" type="keaml:NoncePayloadType"/>
                                        <element name="CertReq"
type="keaml:CertificateRequestPayloadType" minOccurs="0"/>
```

```
                                        <element name="PubKey"
type="keaml:PubKeyPayloadType" minOccurs="0"/>
                                        <element name="Sig" type="keaml:SignaturePayloadType"
minOccurs="0"/>
                                        <element name="Info"
type="keaml:NotificationPayloadType" minOccurs="0"/>
                                    </sequence>
                                </restriction>
                            </complexContent>
                        </complexType>
                        <!-- Key Authentication Exchange -->
                        <!-- KeyAuthnRequestMsgType -->
                        <complexType name="KeyAuthnRequestMsgType">
                            <complexContent>
                                <restriction base="keaml:ExchangeMessageAbstractType">
                                    <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body"
type="keaml:KeyAuthnRequestMsgBodyType"/>
                                    </sequence>
                                </restriction>
                            </complexContent>
                        </complexType>
                        <element name="KeyAuthnRequestMsg" type="keaml:KeyAuthnRequestMsgType"/>
                        <!-- KeyAuthnRequestMsgBodyType -->
                        <complexType name="KeyAuthnRequestMsgBodyType">
                            <complexContent>
                                <restriction base="keaml:MessageBodyAbstractType">
                                    <sequence>
                                        <element name="InitiatorNonce"
type="keaml:NoncePayloadType"/>
                                        <element name="ResponderNonce"
type="keaml:NoncePayloadType"/>
                                        <element name="InitiatorKE"
type="keaml:KEPayloadType"/>
                                        <element name="ResponderKE"
type="keaml:KEPayloadType"/>
                                        <element name="InitiatorSA"
type="keaml:SAPayloadType"/>
                                        <element name="ResponderSA"
type="keaml:SAPayloadType"/>
                                        <!-- This encrypted id must have MAC value -->
                                        <element name="EncryptedID"
type="keaml:EncryptedIDPayloadType"/>
                                        <element name="EncryptedReq"
type="keaml:EncryptedAuthnReqPayloadsType"/>
                                        <element name="EncryptedInfo"
type="keaml:EncryptedNotificationPayloadType" minOccurs="0"/>
                                    </sequence>
                                </restriction>
                            </complexContent>
                        </complexType>
                        <!-- KeyAuthnResponseMsgType -->
                        <complexType name="KeyAuthnResponseMsgType">
                            <complexContent>
```

```
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body"
type="keaml:KeyAuthnResponseMsgBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="KeyAuthnResponseMsg" type="keaml:KeyAuthnResponseMsgType"/>
        <!-- KeyAuthnResponseMsgBodyType -->
        <complexType name="KeyAuthnResponseMsgBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="EncryptedResp"
type="keaml:EncryptedAuthnRespPayloadsType"/>
                                        <element name="EncryptedInfo"
type="keaml:EncryptedNotificationPayloadType" minOccurs="0"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- **********************
InformationalExchange
    ********************** -->
        <!-- InformationExchangeMessageType -->
        <complexType name="InformationExchangeMessageType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body"
type="keaml:InformationExchangeMessageBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="InformationExchangeMsg" type="keaml:InformationExchangeMessageType"/>
        <!-- InformationExchangeMessageBodyType -->
        <complexType name="InformationExchangeMessageBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="EncryptedNotification"
type="keaml:EncryptedNotificationPayloadType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- InformationExchangeClearMessageType -->
        <complexType name="InformationExchangeClearMessageType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
```

```
                                        <sequence>
                                                <element name="Header"
type="keaml:MessageHeaderType"/>
                                                <element name="Body"
type="keaml:InformationExchangeClearMessageBodyType"/>
                                        </sequence>
                                </restriction>
                        </complexContent>
                </complexType>
                <element name="InformationExchangeClearMsg"
type="keaml:InformationExchangeClearMessageType"/>
                <!-- InformationExchangeMessageBodyType -->
                <complexType name="InformationExchangeClearMessageBodyType">
                        <complexContent>
                                <restriction base="keaml:MessageBodyAbstractType">
                                        <sequence>
                                                <element name="Notification"
type="keaml:NotificationPayloadType"/>
                                        </sequence>
                                </restriction>
                        </complexContent>
                </complexType>
</schema>
```

APPENDIX B KEAML-KE SCHEMA

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:keaml-ke="urn:names:tc:KEAML-
KE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" targetNamespace="urn:names:tc:KEAML-KE:1.0"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="1.0">
        <import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-
schema.xsd"/>
        <import namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-
schema.xsd"/>
        <import namespace="urn:names:tc:KEAML:1.0" schemaLocation="keaml-schema-1.0.xsd"/>
        <annotation>
                <documentation>
      Document identifier: keaml-ke-schema-1.0
    </documentation>
        </annotation>
        <!-- SituationIDValues -->
        <simpleType name="SituationIDValues">
                <restriction base="anyURI">
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:situation-id:comm-
protocol"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:situation-id:security-
classification-level"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:situation-id:connectivity-
type"/>
                </restriction>
        </simpleType>
        <!-- KKESituationType -->
        <complexType name="SituationType">
                <complexContent>
                        <extension base="keaml:SituationType">
                                <sequence>
                                        <element name="SituationID" type="keaml-
ke:SituationIDValues"/>
                                        <element name="Value" type="string"/>
                                </sequence>
                        </extension>
                </complexContent>
        </complexType>
        <element name="Situation" type="keaml-ke:SituationType"
substitutionGroup="keaml:Situation"/>
        <!-- ProtocolIDType -->
        <simpleType name="ProtocolIDType">
                <restriction base="keaml:ProtocolIDType">
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:protocol-type:keaml"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:protocol-type:keaml-ke"/>
                </restriction>
        </simpleType>
        <element name="ProtocolID" type="keaml-ke:ProtocolIDType"
substitutionGroup="keaml:ProtocolID"/>
        <!-- TransformTypeType -->
        <!-- The only transform type used in Phase I is sa-attr; the other are for Phase II -->
        <simpleType name="TransformTypeType">
```

```
                <restriction base="keaml:TransformTypeType">
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:encr"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:prf"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:integ"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:dh"/>
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:authn"/>
                </restriction>
        </simpleType>
        <element name="TransformType" type="keaml-ke:TransformTypeType"/>
        <!-- this is the only transform in KEAML-KE; the type is urn:oasis:names:tc:keaml:1.0:transform-
type:sa-attr -->
        <!-- in Phase II, a transformID reuses Enc and Dig IDs. -->
        <simpleType name="TransformIDType">
                <restriction base="keaml:TransformIDType">
                        <enumeration value="urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke"/>
                </restriction>
        </simpleType>
        <element name="TransformID" type="keaml-ke:TransformIDType"/>
        <simpleType name="AuthnMethodTypeType">
                <restriction base="anyURI">
                        <enumeration value="urn:oasis:names:tc:keaml-ke:1.0:authnmethod-
type:preshared-key"/>
                        <enumeration value="urn:oasis:names:tc:keaml-ke:1.0:authnmethod-
type:publickey-signature"/>
                </restriction>
        </simpleType>
        <complexType name="AuthnMethodType">
                <sequence>
                        <element ref="ds:SignatureMethod"/>
                </sequence>
                <attribute name="type" type="keaml-ke:AuthnMethodTypeType" use="required"/>
        </complexType>
        <!-- AuthnPayloadType -->
        <complexType name="AuthnPayloadType">
                <complexContent>
                        <restriction base="keaml:AuthnPayloadType">
                                <sequence>
                                        <element name="Type" type="keaml-
ke:AuthnMethodType"/>
                                        <element name="AuthnData" type="hexBinary"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="AuthnPayload" type="keaml-ke:AuthnPayloadType"/>
        <!-- PRFType -->
        <complexType name="PRFType">
                <sequence>
                        <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="Algorithm" type="anyURI" use="required"/>
        </complexType>
        <!-- IntegType; usually only HMAC function -->
        <complexType name="IntegType">
                <sequence>
```

```
                <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
<complexType name="GroupType">
        <sequence>
                <element name="Type" type="keaml-ke:GroupTypeType"/>
                <element name="Desc" type="keaml-ke:GroupDescType"/>
                <element name="Prime" type="hexBinary" minOccurs="0"/>
                <element name="GeneratorOne" type="decimal" minOccurs="0"/>
                <element name="GeneratorTwo" type="decimal" minOccurs="0"/>
                <element name="CurveA" type="hexBinary" minOccurs="0"/>
                <element name="CurveB" type="hexBinary" minOccurs="0"/>
                <!-- the group order of elliptical curve group -->
                <element name="Order" type="hexBinary" minOccurs="0"/>
                <!-- field size of DH group in bits -->
                <element name="FieldSize" type="integer" minOccurs="0"/>
        </sequence>
</complexType>
<!-- GroupDesc -->
<complexType name="GroupDescType">
        <choice>
                <element name="Oakley">
                        <simpleType>
                                <restriction base="string">
                                        <enumeration value="MODP_768"/>
                                        <enumeration value="MODP_1024"/>
                                        <enumeration value="EC2N_GP_155"/>
                                        <enumeration value="EC2N_GP_185"/>
                                </restriction>
                        </simpleType>
                </element>
                <any namespace="##other" minOccurs="0"/>
        </choice>
</complexType>
<!-- GroupTypeType -->
<complexType name="GroupTypeType">
        <choice>
                <element name="Oakley">
                        <simpleType>
                                <restriction base="string">
                                        <enumeration value="MODP"/>
                                        <enumeration value="ECP"/>
                                        <enumeration value="EC2N"/>
                                </restriction>
                        </simpleType>
                </element>
                <any namespace="##other" minOccurs="0"/>
        </choice>
</complexType>
<!-- LifeTypeType -->
<complexType name="LifeTypeType">
        <choice>
                <element name="Type">
                        <simpleType>
                                <restriction base="string">
```

```
                                        <enumeration value="SECONDS"/>
                                        <enumeration value="KILOBYTES"/>
                                    </restriction>
                                </simpleType>
                            </element>
                            <any namespace="##other" minOccurs="0"/>
                        </choice>
                </complexType>
                <!-- SADataAttributeType, Phase I attributes; Phase II are defined in profile, e.g. SAML Profile of
KEAML-KE -->
                <!-- This attribute is for the transform type urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr -->
                <complexType name="SADataAttributeType">
                        <complexContent>
                                <restriction base="keaml:DataAttributeType">
                                        <sequence>
                                                <element name="EncryptionAlg"
type="xenc:EncryptionMethodType"/>
                                                <!-- for Authn field -->
                                                <element name="AuthnAlg" type="keaml-
ke:AuthnMethodType"/>
                                                <!-- info about a group over which to do Diffie-Hellman-->
                                                <element name="Group" type="keaml-ke:GroupType"/>
                                                <!-- for key calculation -->
                                                <element name="PRF" type="keaml-ke:PRFType"/>
                                                <!-- for the message integrity check using HMAC function -->
                                                <element name="Integ" type="keaml-ke:IntegType"/>
                                                <!-- default="MODP_768" -->
                                                <!-- Life Type, Key length, Field Size, Group Order, .. -->
                                                <element name="LifeType" type="keaml-ke:LifeTypeType"
minOccurs="0"/>
                                                <!-- key length in bits -->
                                                <element name="KeyLen" type="integer" minOccurs="0"/>
                                                <!-- pseudo random function, by default, is the HMAC version
of the negotiated hash algorithm -->
                                                <element name="HashAlg" type="ds:DigestMethodType"
minOccurs="0"/>
                                        </sequence>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- global SADataAttribute -->
                <element name="SADataAttribute" type="keaml-ke:SADataAttributeType"/>
                <!-- EncryptedSA2ReqPayloadsType - {SAi2, [KEi], Ni}-->
                <complexType name="EncryptedSA2ReqPayloadsType">
                        <complexContent>
                                <restriction base="keaml:EncryptedPayloadType">
                                        <sequence>
                                                <element ref="xenc:EncryptedData"/>
                                                <element ref="ds:DigestValue"/>
                                        </sequence>
                                        <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml-ke:1.0:payload-type:sa2-request"/>
                                </restriction>
                        </complexContent>
                </complexType>
                <!-- SA2ReqPayloadsType - {SAi2, [KEi], Ni}-->
```

```
        <complexType name="SA2ReqPayloadsType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="SAi2" type="keaml:SAPayloadType"/>
                                        <element name="KEi2" type="keaml-ke:KEPayloadType"
minOccurs="0"/>
                                        <element name="Ni2" type="keaml:NoncePayloadType"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml-ke:1.0:payload-type:sa2-request"/>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SA2ReqPayloads" type="keaml-ke:SA2ReqPayloadsType"/>
        <!-- EncryptedSA2RespPayloadsType - {SAr2, [KEr], Nr} -->
        <complexType name="EncryptedSA2RespPayloadsType">
                <complexContent>
                        <restriction base="keaml:EncryptedPayloadType">
                                <sequence>
                                        <element ref="xenc:EncryptedData"/>
                                        <element ref="ds:DigestValue"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:sa2-response"/>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SA2RespPayloadsType - {SAr2, [KEr], Nr} -->
        <complexType name="SA2RespPayloadsType">
                <complexContent>
                        <restriction base="keaml:PayloadType">
                                <sequence>
                                        <element name="SAr2" type="keaml:SAPayloadType"/>
                                        <element name="KEr2" type="keaml-ke:KEPayloadType"
minOccurs="0"/>
                                        <element name="Nr2" type="keaml:NoncePayloadType"/>
                                </sequence>
                                <attribute name="Type" type="anyURI"
fixed="urn:oasis:names:tc:keaml:1.0:payload-type:sa2-response"/>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SA2RespPayloads" type="keaml-ke:SA2RespPayloadsType"/>
        <complexType name="KEPayloadType">
                <complexContent>
                        <restriction base="keaml:KEPayloadType">
                                <sequence>
                                        <!-- DH public value passed in a KE payload -->
                                        <element name="DHPublicComp" type="hexBinary"/>
                                </sequence>
                                <anyAttribute namespace="##local" processContents="lax"/>
                        </restriction>
                        <!-- DH public components ???? -->
                </complexContent>
        </complexType>
```

```
        <element name="KE" type="keaml-ke:KEPayloadType"/>
        <!-- SA2RequestMsgType -->
        <complexType name="SA2RequestMsgType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body" type="keaml-
ke:SA2RequestMsgBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SA2RequestMsg" type="keaml-ke:SA2RequestMsgType"/>
        <!-- SA2RequestMsgBodyType -->
        <complexType name="SA2RequestMsgBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="EncryptedReq" type="keaml-
ke:EncryptedSA2ReqPayloadsType"/>
                                        <element name="EncryptedInfo"
type="keaml:EncryptedNotificationPayloadType" minOccurs="0"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <!-- SA2ResponseMsgType -->
        <complexType name="SA2ResponseMsgType">
                <complexContent>
                        <restriction base="keaml:ExchangeMessageAbstractType">
                                <sequence>
                                        <element name="Header"
type="keaml:MessageHeaderType"/>
                                        <element name="Body" type="keaml-
ke:SA2ResponseMsgBodyType"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
        <element name="SA2ResponseMsg" type="keaml-ke:SA2ResponseMsgType"/>
        <!-- SA2ResponseMsgBodyType -->
        <complexType name="SA2ResponseMsgBodyType">
                <complexContent>
                        <restriction base="keaml:MessageBodyAbstractType">
                                <sequence>
                                        <element name="EncryptedResp" type="keaml-
ke:EncryptedSA2RespPayloadsType"/>
                                        <element name="EncryptedInfo"
type="keaml:EncryptedNotificationPayloadType" minOccurs="0"/>
                                </sequence>
                        </restriction>
                </complexContent>
        </complexType>
</schema>
```

APPENDIX C SA STORE SCHEMA

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:store="urn:names:tc:SA-STORE:1.0"
xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
targetNamespace="urn:names:tc:SA-STORE:1.0" elementFormDefault="unqualified"
attributeFormDefault="unqualified" version="1.0">
        <import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-
schema.xsd"/>
        <import namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-
schema.xsd"/>
        <import namespace="urn:names:tc:KEAML:1.0" schemaLocation="keaml-schema-1.0.xsd"/>
        <import namespace="urn:names:tc:KEAML-KE:1.0" schemaLocation="keaml-ke-schema-
1.0.xsd"/>
        <annotation>
                <documentation>
      Document identifier: sa-store-1.0
    </documentation>
        </annotation>
        <complexType name="SAType" abstract="true">
                <sequence>
                        <element name="SPI1" type="keaml:SPIType"/>
                        <element name="SPI2" type="keaml:SPIType"/>
                        <element name="ProtocolID" type="keaml-ke:ProtocolIDType"/>
                </sequence>
        </complexType>
        <element name="SA" type="store:SAType"/>
        <complexType name="Phase1SAType">
                <complexContent>
                        <extension base="store:SAType">
                                <sequence>
                                        <element name="ChildSARefs"
type="store:ChildSARefsType"/>
                                        <element name="AuthnAlg" type="keaml-
ke:AuthnMethodType"/>
                                        <element name="EncryptionAlg"
type="xenc:EncryptionMethodType"/>
                                        <element name="Group" type="keaml-ke:GroupType"/>
                                        <element name="PRF" type="keaml-ke:PRFType"/>
                                        <element name="Integ" type="keaml-ke:IntegType"/>
                                        <element name="LifeType" type="keaml-ke:LifeTypeType"
minOccurs="0"/>
                                        <!-- key length in bits -->
                                        <element name="KeyLen" type="integer" minOccurs="0"/>
                                        <element name="HashAlg" type="ds:DigestMethodType"
minOccurs="0"/>
                                        <element name="SKseed" type="hexBinary"/>
                                        <element name="SKd" type="hexBinary"/>
                                        <element name="SKai" type="hexBinary"/>
                                        <element name="SKar" type="hexBinary"/>
                                        <element name="SKei" type="hexBinary"/>
                                        <element name="SKer" type="hexBinary"/>
                                        <element name="SKpi" type="hexBinary"/>
```

```
                                        <element name="SKpr" type="hexBinary"/>
                                    </sequence>
                                </extension>
                            </complexContent>
                        </complexType>
                        <element name="Phase1SA" type="store:Phase1SAType"/>
                        <complexType name="Phase2SAType">
                            <complexContent>
                                <extension base="store:SAType">
                                    <sequence>
                                        <element name="EncryptionAlg"
type="xenc:EncryptionMethodType"/>
                                        <element name="Group" type="keaml-ke:GroupType"
minOccurs="0"/>
                                        <element name="PRF" type="keaml-ke:PRFType"/>
                                        <element name="Integ" type="keaml-ke:IntegType"/>
                                        <element name="LifeType" type="keaml-ke:LifeTypeType"
minOccurs="0"/>
                                        <!-- key length in bits -->
                                        <element name="KeyLen" type="integer" minOccurs="0"/>
                                        <element name="HashAlg" type="ds:DigestMethodType"
minOccurs="0"/>
                                        <element name="SKseed2" type="hexBinary"/>
                                        <element name="SKei2" type="hexBinary"/>
                                        <element name="SKai2" type="hexBinary"/>
                                        <element name="SKer2" type="hexBinary"/>
                                        <element name="SKar2" type="hexBinary"/>
                                        <element name="ParentSARef" type="keaml:SPIType"/>
                                    </sequence>
                                </extension>
                            </complexContent>
                        </complexType>
                        <element name="Phase2SA" type="store:Phase2SAType"/>
                        <complexType name="ChildSARefsType">
                            <sequence minOccurs="0" maxOccurs="unbounded">
                                <element name="ChildSARef" type="keaml:SPIType"/>
                            </sequence>
                        </complexType>
                        <complexType name="SAsType">
                            <sequence maxOccurs="unbounded">
                                <element name="SA" type="store:SAType"/>
                            </sequence>
                        </complexType>
                        <element name="SAs" type="store:SAsType"/>
</schema>
```

APPENDIX D SAMPLE MESSAGES

```
<!—SA KE Request -->
<keaml:SAKERequestMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
        <Header MajorVersion="1" MinorVersion="0">
                <InitiatorCookie>D4861B4F</InitiatorCookie>
                <ResponderCookie>00</ResponderCookie>
                <MessageID>1</MessageID>
                <Flags>4</Flags>
        </Header>
        <Body>
                <SA>
                        <Proposal number="1">
                                <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
                                <Transform number="1">
                                        <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                                        <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                                        <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                                                <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                                                <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                                        <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                                </AuthnAlg>
                                                <Group>
                                                        <Type>
                                                                <Oakley>MODP</Oakley>
                                                        </Type>
                                                        <Desc>
                                                                <Oakley>MODP_1024</Oakley>
                                                        </Desc>
                                                </Group>
                                                <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <LifeType>
                                                        <Type>KILOBYTES</Type>
                                                </LifeType>
                                        </DataAttribute>
                                </Transform>
                        </Proposal>
                        <Proposal number="2">
                                <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
```

```
                                        <Transform number="1">
                                                <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                                                <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                                                <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                                                        <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
                                                        <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                                                <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
                                                        </AuthnAlg>
                                                        <Group>
                                                                <Type>
                                                                        <Oakley>EC2N</Oakley>
                                                                </Type>
                                                                <Desc>

        <Oakley>EC2N_GP_185</Oakley>
                                                                </Desc>
                                                        </Group>
                                                        <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                        <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                        <LifeType>
                                                                <Type>KILOBYTES</Type>
                                                        </LifeType>
                                                </DataAttribute>
                                        </Transform>
                                </Proposal>
                                <keaml:Situation xsi:type="keaml-ke:SituationType">
                                        <SituationID>urn:oasis:names:tc:keaml:1.0:situation-id:security-
classification-level</SituationID>
                                        <Value>C</Value>
                                </keaml:Situation>
                        </SA>
                        <KE xsi:type="keaml-ke:KEPayloadType">

        <DHPublicComp>308201233081990609 2A864886F70D01030130818B02818100FFFFFFFFFFF
FFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A0879
8E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42
E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFF
FFFFFFFFFFFFFF020102020203FF038184000218078D4CC2A9F0474D153D3C7E032077D292833B05
F26643704EF8A29C677AA313B222AB3A970ED0521CCE746F9FE00E82D58AE9BC94488AC59D201
B46B89B1C46B7D8DD062CFB7A197187AAED8DEA7EC0EE493197B8403A490C417813835D9ACF
D96905F357643FCC062B88DA6B1981DD7BDCA2D52BADDB6877A33008BC7010943</DHPublicCo
mp>
                        </KE>
                        <Nonce>
                                <Nonce>47B700E7DE17ACB6</Nonce>
                        </Nonce>
        </Body>
</keaml:SAKERequestMsg>
<!-- SA KE Response Message -->
```

```
<keaml:SAKEResponseMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
        <Header MajorVersion="1" MinorVersion="0">
                <InitiatorCookie>D4861B4F</InitiatorCookie>
                <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
                <MessageID>1</MessageID>
                <Flags>0</Flags>
        </Header>
        <Body>
                <SA>
                        <Proposal number="1">
                                <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
                                <Transform number="1">
                                        <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                                        <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                                        <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                                                <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                                                <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                                        <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                                </AuthnAlg>
                                                <Group>
                                                        <Type>
                                                                <Oakley>MODP</Oakley>
                                                        </Type>
                                                        <Desc>
                                                                <Oakley>MODP_1024</Oakley>
                                                        </Desc>
                                                </Group>
                                                <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <LifeType>
                                                        <Type>KILOBYTES</Type>
                                                </LifeType>
                                        </DataAttribute>
                                </Transform>
                        </Proposal>
                </SA>
                <KE xsi:type="keaml-ke:KEPayloadType">

        <DHPublicComp>308201243081990609 2A864886F70D01030130818B02818100FFFFFFFFFFFF
FFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A0879
8E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42
E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286651ECE65381FFF
FFFFFFFFFFFFFF020102020203FF038185000 2818100BB9220232D306FDC7A075CDA34C6D07EC587
```

E895F92E617544DE4D47764C93DD184984D8D0D2ECF9742D5E783F8EE2800E4702398DF443CC10
85919324D23194C81D4AD5806902979E4E37069BFE7DB82AFDCEB46057E00B8068911166D6DFCE
3D5392F208B04C5D62404DEBFDE569E2644122FE09B6169C5E215207A18AE7AC</DHPublicComp
>

```
                </KE>
                <Nonce>
                        <Nonce>9358BE10CBE6B574</Nonce>
                </Nonce>
        </Body>
</keaml:SAKEResponseMsg>
<!-- Key authentication request message  -->
<keaml:KeyAuthnRequestMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
        <Header MajorVersion="1" MinorVersion="0">
                <InitiatorCookie>D4861B4F</InitiatorCookie>
                <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
                <MessageID>1</MessageID>
                <Flags>0</Flags>
        </Header>
        <Body>
                <InitiatorNonce>
                        <Nonce>47B700E7DE17ACB6</Nonce>
                </InitiatorNonce>
                <ResponderNonce>
                        <Nonce>9358BE10CBE6B574</Nonce>
                </ResponderNonce>
                <InitiatorKE xsi:type="keaml-ke:KEPayloadType">

        <DHPublicComp>308201233081990609 2A864886F70D01030130818B02818100FFFFFFFFFFF
FFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A0879
8E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42
E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE64928665 1ECE65381FFF
FFFFFFFFFFFFFF0201020202 03FF038184000281807 8D4CC2A9F0474D153D3C7E032077D292833B05
F26643704EF8A29C677AA313B222AB3A970ED0521CCE746F9FE00E82D58AE9BC94488AC59D201
B46B89B1C46B7D8DD062CFB7A197187AAED8DEA7EC0EE493197B8403A490C417813835D9ACF
D96905F357643FCC062B88DA6B1981DD7BDCA2D52BADDB6877A33008BC7010943</DHPublicCo
mp>
                </InitiatorKE>
                <ResponderKE xsi:type="keaml-ke:KEPayloadType">

        <DHPublicComp>308201243081990609 2A864886F70D01030130818B02818100FFFFFFFFFFF
FFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A0879
8E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42
E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE64928665 1ECE65381FFF
FFFFFFFFFFFFFF0201020202 03FF038185000281810 0BB9220232D306FDC7A075CDA34C6D07EC587
E895F92E617544DE4D47764C93DD184984D8D0D2ECF9742D5E783F8EE2800E4702398DF443CC10
85919324D23194C81D4AD5806902979E4E37069BFE7DB82AFDCEB46057E00B8068911166D6DFCE
3D5392F208B04C5D62404DEBFDE569E2644122FE09B6169C5E215207A18AE7AC</DHPublicComp
>
                </ResponderKE>
                <InitiatorSA>
                        <Proposal number="1">
```

```
                    <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
                    <Transform number="1">
                        <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                        <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                        <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                            <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                            <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                            </AuthnAlg>
                            <Group>
                                <Type>
                                    <Oakley>MODP</Oakley>
                                </Type>
                                <Desc>
                                    <Oakley>MODP_1024</Oakley>
                                </Desc>
                            </Group>
                            <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                            <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                            <LifeType>
                                <Type>KILOBYTES</Type>
                            </LifeType>
                        </DataAttribute>
                    </Transform>
                </Proposal>
                <Proposal number="2">
                    <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
                    <Transform number="1">
                        <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                        <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                        <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                            <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
                            <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
                            </AuthnAlg>
                            <Group>
                                <Type>
                                    <Oakley>EC2N</Oakley>
                                </Type>
                                <Desc>

        <Oakley>EC2N_GP_185</Oakley>
```

```
                                                    </Desc>
                                                </Group>
                                                <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <LifeType>
                                                    <Type>KILOBYTES</Type>
                                                </LifeType>
                                            </DataAttribute>
                                        </Transform>
                                    </Proposal>
                                    <keaml:Situation xsi:type="keaml-ke:SituationType">
                                        <SituationID>urn:oasis:names:tc:keaml:1.0:situation-id:security-
classification-level</SituationID>
                                        <Value>C</Value>
                                    </keaml:Situation>
                                </InitiatorSA>
                                <ResponderSA>
                                    <Proposal number="1">
                                        <keaml:ProtocolID>urn:oasis:names:tc:keaml:1.0:protocol-
type:keaml</keaml:ProtocolID>
                                        <Transform number="1">
                                            <TransformType xsi:type="keaml-
ke:TransformTypeType">urn:oasis:names:tc:keaml:1.0:transform-type:sa-attr</TransformType>
                                            <TransformID xsi:type="keaml-
ke:TransformIDType">urn:oasis:names:tc:keaml:1.0:transform-type:keaml-ke</TransformID>
                                            <DataAttribute xsi:type="keaml-ke:SADataAttributeType">
                                                <EncryptionAlg
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                                                <AuthnAlg type="urn:oasis:names:tc:keaml-
ke:1.0:authnmethod-type:publickey-signature">
                                                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                                </AuthnAlg>
                                                <Group>
                                                    <Type>
                                                        <Oakley>MODP</Oakley>
                                                    </Type>
                                                    <Desc>
                                                        <Oakley>MODP_1024</Oakley>
                                                    </Desc>
                                                </Group>
                                                <PRF
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <Integ
Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
                                                <LifeType>
                                                    <Type>KILOBYTES</Type>
                                                </LifeType>
                                            </DataAttribute>
                                        </Transform>
                                    </Proposal>
                                </ResponderSA>
                                <EncryptedID>
                                    <xenc:EncryptedData>
```

<xenc:CipherData>
<xenc:CipherValue>fQRK49+5X4xjbEvjuYGp9I9l
93beNkfVYRB9M7IxtTHDWKeA
DEoTRMtZ8yV/Egiz6VhE6qY/
BVh0dyWk28EYGJVsJ7o8MihB
RBo2skXb7e9EPJTHUviOLD5g
emi1a/CsteI7cBPY4Hb67zcx
yaYv5t17cxCsPDwP1kj6g0cT
Kgd+KN5eekoUCm+/Lh3X13i3
GUeBFOBVi1YjFlETI6gAgk++
DocGOuSerQmc+XHR8hxA/jOx
NbMQv/KnMNrfi85dk9y1dXoS
TJQFP7amSR+Pd0YLROEcfGk5
7byCIu2zG3x2rBX1w1sU9MT6
pT5ufITlWm4gRuQ8o594HSr7
48jZEjwNSBqLPl6N6H6rvQUK
OqaWUu4+wEDs0d8DgN3eeGZc
JBzDIMZid3ahAMJnI5QkfhU1
rVC8FzI+QdEiY1ChR2MESVGw
GIMPsxJH1RGkADnF3y/eza2k
2BGE6FVqXl4Y75087d2B5hKn
YC5FFosoyDFfyB5g98vHcN4l
c/fjoYuLvSX5UopYn0Q2cwzz
PgeG2TXw8yBA4vvl9prkyO71
iZf2+3bxI4O9158WHacmAXn7
4tOJk4FyaNWYcnzoC7eNYea5
ypvK8q1EVeYAnRPnI00=</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
<ds:DigestValue>lQKPIi9hPUzzxOdA5XHcN0us
gC4=</ds:DigestValue>
</EncryptedID>
<EncryptedReq>
<xenc:EncryptedData>
<xenc:CipherData>
<xenc:CipherValue>3ngUuUuf1J+XQMw27YSsvbeS
0grYraaZ7HayMndeHmF+lDp8
NdaCVNdBDiGNN/iUlOhU5WcM
tAZNk4ciQJahYGxQmILuWL6N
XU7fwFko2v9+lDp8NdaCVNdB
DiGNN/iU+05c+aoDgwgSGdE/
uL4yhFTRGZwHYCvmdit7T08r
BQpPcCd1hdON3ESDKT94CQgO
wR6CAjruEQzjv/qo/e0dMIft
1ZG1VBpbkjnR8sF7mfK/dmiI
7u1zA2bQUBgmoZiLOiYUA2vK
/GUV2ALt5SCsBYy5GXQoXtXd
eYIiQfvVN5wMxvJXEQpLsTiI
IrAyxwwm26wwrvuK8cyuSgbX
4MLuDGXvZQ+7AMq+o8ph2vT6
khgZMg/XdGZSy6PSlLJxlr41
SeIpcW0Z/LSaGMQ84dcZRyiW
SRb/RwjZuWi8yxUJpasE6KBF
iShg9A/p/Kotrw+QaiUWyNlJ
asVYAouE8/KdYD3+L/DBHM3d
ib51ID1JWdY01Bm4lViVviul

```
6Qw9KkaukM4bdZuB9Ee8ufUV
SfdLPgs9GE+bBEuI/tc/hWwv
aL/Oy9mCqYtl9YR523COVnUG
jrfBxPIxBA+g+0P438vWYmt+
mFhAz3Y1HyAJHdsabdx27KP1
Nxc53UgnQQ11TVVvQZxJMLiN
vJCebRNwyfOu1AcApTXunznF
vAF361Hklry5cJwTHWQt73VX
pFX3QskXizuTSOQMp76cvLE9
csBKUaYvTZaWrcUjUJo/R/y1
wIqYRB+bi+k1WK6AWHWpkCtL
y17ICRDjsqnOMquox0JvjwjQ
f0AqQ9nEaeQUvuVTTvWL3o5i
F3TrIfXmyUQRQib3YHThJEfd
vTwNbunCBwgb+7pR29olcLfo
MHl9BGkwWrJhgVulJapfm85T
9xOFArXsy/IXNwwOrpyyhPKZ
dTimpcIcBId4Sbf9yGLVlCKL
odZ1m8bf0aghjD4X/L7hVFyQ
uQyONA/v37sPNXSaAVFIqtes
xcAOWcUQdCNnxDq9mWp+ARte
t26M4UKVc6Rzw0Xiq2SZaY69
GvhuzhhCUSwyuXsUzDHbJQaW
vpKeSVP4b0/uz6GtmjjKM/ka
CYVjEz3t4pSzrF+zqnWat/2W
ewJ2lBDNE2Fkndh/VrljpFqE
h3k=</xenc:CipherValue>
                    </xenc:CipherData>
                </xenc:EncryptedData>
                <ds:DigestValue>IT76MbUaxDPmnKuBvmOiQ+ky
cM4=</ds:DigestValue>
            </EncryptedReq>
        </Body>
</keaml:KeyAuthnRequestMsg>
<!—Key authentication response message -->
<keaml:KeyAuthnResponseMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
        <Header MajorVersion="1" MinorVersion="0">
            <InitiatorCookie>D4861B4F</InitiatorCookie>
            <ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
            <MessageID>1</MessageID>
            <Flags>0</Flags>
        </Header>
        <Body>
            <EncryptedResp>
                <xenc:EncryptedData>
                    <xenc:CipherData>
                        <xenc:CipherValue>26BpE4YJjyngZCVOwoFZ2uvl
QVU9V5To4osH17dtiZDRODjx
axmk2Hhrv8EJEWq2Vh98EbRS
CuzbSKuxv2ZJHfMAyorgCpgr
roPy1vGiLQiApdY0ibho4LSf
```

8pa0igWVIcVcy7XHjCp4+n1r
dzS7sbZPhgJdLMnzKrbUvavn
M0f79rDVjU+Xhg4gT6DhDhVP
vPDR/5geR31Ff6hCyjLbTPC3
VW8l6EkcUw8ufHTLcjiFUblo
j7ygV/ifNORqaXOXHQtXxpUw
f/N8b8Ul+96P+T2VkBn2g/Fq
DTQ8IGHR/FwQc7c07DFv5rlE
GC+gb0XRnIPZJ0uRTEK8Mnpa
U8fUq66OWERd4nPTavg97hRC
FY/BQ0B1ZpRxT8FmKCjvdWNl
MLQma1SoUhhT4sQlRzAc+hqA
I1Ltk+UznVtKqZtDqK070sQD
7RcSEfD3E/ENmAVQ9mM60mIY
inD2uLZb65+lsqjvRsdfR7/o
skW+zPTj9JKoLw3wMse5g2Vd
Wdr4cGAx0pJs2786TDQYCP5y
rU7EtpIimfdL47SDsiGcjiSj
hsOFALsfPFfPy1r7eMo2vKYy
P1oFF0ekbv1j1AcPkfGPinfj
D+X6zD821tn03qpj8fM1Kqq/
SSXnCMfmaCY+YeHzR43EPgsf
eqWw26taMfQl/L69n3yc1m59
hkWun9mgsvitLeQCcrll/2Ta
ruyEJGDG0zKtzhdf4PuRKWCP
UjdxxtU14p8heFAFyd8Kb7uK
dYaWvDtllLQGuwQKD+QoILYq
Pb4JnxGoaVb9/Q5nqmv2vscu
Qyk4egBDlMJQqsYPfZmUtKmv
sZoRoQ8cik3nUGdMEz4iL2y0
IHaTsRIdPM+eOENqqW16Lu3O
hWA0u1fjXqOeHGgHWm3z2UfN
xKPLz0Y3Dirv1oJV3lvns825
st8YnxouLWVMT7iEQXsR7Isn
6dmW+q2UfZdcU7qgDaO2s2HM
VDqb5v7kQ7NZ1qSLs2/7y86R
3h/dJ7xFIIhHi4iq7jWnJXGk
I3WK0hFb1Tz40kzxZmw2xUvX
DAToh1D3saXwh5QVodJTXSOm
AnxRiFzpflsFmthFMOLSxPcM
I3VsCbYHhopEz0EBfPeL6oHd
xOUcai2DP8J3PkI5CYMFeWNH
LK11ANvSfAxVOKuYnAR+oawp
Vtr1IKvDvn3HnFUwiL1K5Nat
rdwkKIFcU5NGAXKWqpzWAL+s
tC9KN03jJuQ+sWGjyMG4/Zzt
v3NLgSa+4wf/kd4iFOEsvKIy
I++Vn4nACWE=</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
<ds:DigestValue>bEsoQBC1UY+qbluwHHqqBRxR
GQ8=</ds:DigestValue>
</EncryptedResp>
</Body>
</keaml:KeyAuthnResponseMsg>
<! -- SA2 request -->

<keaml-ke:SA2RequestMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
<Header MajorVersion="1" MinorVersion="0">
<InitiatorCookie>D4861B4F</InitiatorCookie>
<ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
<MessageID>2</MessageID>
<Flags>0</Flags>
</Header>
<Body>
<EncryptedReq>
<xenc:EncryptedData>
<xenc:CipherData>
<xenc:CipherValue>fPFxKjnqE5qQPCboPizWhxpi
v5XC3j5POlIU3cx2cveaiRrF
MqnJkpk1LzWeoAZC5ylIurG5
Xepq6WLXrKV8arjmCbcH5OJ1
2LrGI6mgWj8EcdwxgCZ8tzr+
/J12tPtDOdxgx37rzMZrJxfK
JO4EJVBRLm5U7+P7EtwOFo7O
V/gu1fqCsQGAWwgCLcAxjurx
vuAViDQmeyvuueLRZCbOSeip
HJ9Fm50JnEYz52Ys6WvwUCCp
hcsLES/jbkUFfe/8OmRIgl8L
UM7DDjfx8oNc9A7i8f0YfIQM
jHblyqUpjL44AIbEFehCM5LI
wfNWjAwcZ25qLjE8dlhZt2LK
oNMX4y4H0t+oikNbUSZZYOrv
qRWjRM0EgJ0SkKkBNrzw/Tcf
dlBm3ax+OhNxeOZzDk0vBfz8
ys4fEM0nffxmPk0eyFX8lYbC
GuYXMS7/lRh8ayffKzXdzHDB
TPShpr9q6Ve7pmogYbm8YjfX
in7+5AmixQB6ymJkgNXVhNeP
EFEG7MammMlxY4WV2je9/6L4
awP8J110mfXCUutdJcyyeT1q
lAQMI112HNIB/vu+0+HPNpbK
hD0WcU7yCuhqQsKcTTdC54FU
zc59X/Ds3zOPMLF57QK2nABu
qWI5BhNH6l8EkZKHn473ork/
nHDRo+hK9P6tEh+P4c54O3Qr
0eS2xruEYPiYTGWeh8UCHl7k
DgwFW9Kglv4/dpJrxB2dWQAL
D0m4zwLMs+QITYwBfShb0Vor
R4Jg+mlPGG3i+FxV1FZgJSe1
ZJwnpgxHa3HETd+Yt+D5AgXP
L/s2ii/5w5yMgR7VFk3eujh7
kXs6Iib93XX8T7T+DrgiojM3
44e6NTxpg48FZZWMmUuDJqNZ
eYSxeD7kqpt22Zo0voIwMtsC
MalrD4rB7LxPIeCnQYUDZBCc
JH28ICfIrFtLWgp2zCjmeb03
Xegt6q5nAQt0ucr5cLO7UKDT

aLgNUFAq8QvJMiz+StrvSvgD
g+PAKoBM9HtQRn8+iRcG0Tty
CcKq6jLXHhx/3Pfqw2tFp6OX
BNwcjxYod1Y//mbgPvFQCg/b
hwKinM2aj7GYQ30qaLnVfrKv
ypmE+OfWenkumDaB3futgebe
wayvITHoxcqeTvNwBDsRYslv
Pii2kQjeTO0UXhZZdLX01qrZ
XD8ZBxTXcVSn8Qx/NodWE4YH
5fEgCZnugmbpd5eyFWIk+DDs
3veXDkGnGWVxVOJbINJ3gF8o
EsOpvB5+L1tVkNvdzzRk/lJM
xrFUsWwgDGdQXCQdqA73/Gvs
kuusD2UFb0ku1hAFCBHwQchR
Idp3ZFuAsFBZL7F0xlhfW2Sw
fxioQqJHUbia/1prg5JijgAB
IK02PxxmvYmW4GUDQXMK2DK0
Ziukb23MiJWY7kJ1WoPdrG8A
nhUKurj6dNEIg1TWkA4UNh2T
yITpi+wq1ofCt8MMHGW6CgcN
rniZRFeAnt8HR4/6gf7z9e72
tHtmSO9m2mrammbnNe29XgsP
MAz/3qoXuOoWPuAoOzAOSpjs
1pwYdyQpx49biD7CxvTXbFm7
GimEhhd3c0BCZO2hNNGCFVRM
rfp61iwsLIpxYvy/BtnnKUi6
sbld6mrpYtespXxqbvhIl2g6
Idn6bx70W9qKyiFeaYCktqtN
FfSX5StBhBtLzU4KLPe/eO8j
hqpkl2dhOB9T9LydYVf4oe4u
tdFz1nnIV4b9eCDWsDXsNIuc
gxII72G1AuPhfSv3dt36Iwk4
wQLoXJ7P8jF+9usxtXcqyTiq
91hIzR8AAYYUpQkOZouGXeVU
RicoJK+xeZe6G6xZn473ork/
nHDRo+hK9P6tEsFnVKm1TewJ
rVrYVUFB73KdD3DelGpdyO/Y
hih9dsXs8z+T0QaCJR1fngG+
eeSJ7adIb1XkRLVeJGk5ll38
OHgdExn2FrQv47Ayv7nDQ/cL
x78Lwle9xrQiXGMos5yP13tB
LF29FK0HkBJSb4EPp2mlrrRK
L2e36O4kw71wufvPA3ShNlbX
DGSw+ROrNqqFGlVQZ9T4rrUz
7rhhaNUiIbyI/g641xcRq9JT
xeOiZbeVN6e6MpuC41ZgolL3
/+G+3F1aByNFwnM6T5D2q3Lf
lmRtIhZh/qwD0qQK8kptsal4
0urh6LirHb5kdeGI30Ud5D4K
rT1aPWcS47/ja+oUCFJqIGG5
vGI314p+/uQJosUAIVzCWHJd
NlKOyqOUJv4EPGl8efXz5D8V
FZZwotKzuemLTrWnhTtIMNZU
Pw41ymbEu5ShKHjqD1AFFFGo
8BUVS33zqOqez+yUVALmLI4o
2TI4HT4lAzNsL2swMylfYJ6h

kDaaVr6xrhRL1+yJPeGBYjt+
/26HiO704ykKqVElFdG1lh3X
BN2uL2mqucwZ+0gQvJ/iGsk8
nLbF3K4ALrJZR80EldwoSX/S
zr4P6Ml7eUC2emlZwwHSL3Gu
NzVbPRXPYNaLcvilunklCEMp
L/+fUWGNVOHV7nKsEoV14FyO
C2XMS3gdD/aGHsOd5ua/m9b0
UErlE7bmrxnHtqj+EbBAwxDE
FuYqSDZZCaSs5VOGSRcRiFiJ
1FPLPMzEpPjb8UIaPk/65s4f
XFuPy7XmUS37oelYROqmPwVY
dHclpNvBGBgQGjvMv1z4bkRc
SYDQYinsr8/PqOhlJls/67Jv
VYEc16WAnwQiCc8vElc7Othc
/+fH6lryA/kUApS5InQB3S75
P3ALbe00szwKQ43h2hybYW9Z
Y+pK7cquYeYHFlA+8ibGKsYe
X1x72148v8dvraD3wmM9GFQj
MtvMFaEtN6TbhVLEMnp3r39I
vektCvWfyVGNy4ujN4WI50O6
Tl8armaOl0QOezzvjhs02PlM
6PusKB2pjFw4FoLMM/m5qgVY
LK5bxsmQpgJ5ZwGhcHLaEOtR
TSdvZohUyYaRU8D0BfgNun6U
Onw11oJU10EOIY03+JSU6FTl
Zwy0Bk2ThyJAlqFgiDKoS+Jg
sxp3YdiwwF75jnKrd/XP3nwH
FVv9yAzkfKTps6aSxMN1UUC0
K/K74HCXeYfmaxs3Lvu1v4eM
PE8SxqabTvSwr2miYh8l0v//
Kaj6934yCqgZ2EDzyLm0WNu+
w4EA/0QecwqMnh6QArKgAvxn
OyGY6olrmTy89DCRGRi5iMbz
2X4VX4pgRegcCqFOdJiKCjyH
/FxYXI+3GE/T9fQu2d5Jaom1
YCbUOO39qHss0OSZ6CjCVZbc
HCCSawvcfqJJT1hnb1NVLmFp
2Q69lBErhoAxxEYkM9oeMOAw
bJ5g48pw+ihBQ9CCxunsowBi
ems3o0V3P9SS4tyJT04zmiwo
JzRC0puvNbCf04XQ8896JMBG
HJUTcJ32d/WZJLssN198APKi
Vo0LNWDfkM+K0y9hknXu4dkQ
lndauICuHGZvz+5XIF4Oo69O
4Qm3cvrottkTkfbhn4RkmLjY
3zonZyC2lIvLAqK7rXDZ8Lx6
jXI=</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
<ds:DigestValue>vhdw/cFVtW1srhLX/WffyU8x
Fx8=</ds:DigestValue>
</EncryptedReq>
</Body>
</keaml-ke:SA2RequestMsg>
<!-- SA2 response message -->

<keaml-ke:SA2ResponseMsg xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns:keaml-ke="urn:names:tc:KEAML-KE:1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:sastore="urn:names:tc:SA-
STORE:1.0" xmlns:keaml="urn:names:tc:KEAML:1.0" xsi:schemaLocation="urn:names:tc:KEAML-
KE:1.0
keaml-ke-schema-1.0.xsd">
<Header MajorVersion="1" MinorVersion="0">
<InitiatorCookie>D4861B4F</InitiatorCookie>
<ResponderCookie>B7F09A733C2F2550CA87C8016021D46A</ResponderCookie>
<MessageID>0</MessageID>
<Flags>0</Flags>
</Header>
<Body>
<EncryptedResp>
<xenc:EncryptedData>
<xenc:CipherData>
<xenc:CipherValue>C5tlA2iZrloqAbvjV0UXWE3k
9HyE3JBNoBLtulCQWFWQN+UU
hHVAMwsKgJ7HltakKkZNCXw+
8jkErnddSjzYZTReQ/zXcFYR
5MWIZD1MVXMcL/6PrWT2I/lQ
w6uGyVcYlEda1OCwVcaSzgJw
9lEPFJ9xZ0fEirnNilwMD7Pw
SQVENXW5jDYyTvGZqby/Jl7x
iYa+jeh9yAJZ51nniZhCc2ha
D3YTD2QfZ5FlAHUSQu/7aJJ9
1ePgrVuvS3TCiBa9DLVITNgw
pUWK1RyJ+eAi9CwYfdkIFkwN
SDT9ny7oB3jcfYz+EdUWeTcK
baTK2PMvLhiE8xEmhZgM+wTw
5Vpedmudp4mtPi+0rKtqPNBQ
mhFonOcp2zCTZ/OM0ihfqRHB
VPRXtZMNrqdqbbUDQJaAAjai
7dBA11HFPOBoGU4dgbr1uDw4
ZKtNDflLEBw0gGH5zMQAfBEU
gQw2cvnj4B0o/QVFOGsvElK5
Mf8OUjD8m1rZIE+Z0iggsDby
xZiZqmRxB6/UtBNqQyrWZ/wp
ytXfYD4rtrkEBXohaamANCAG
VkHqrGjVO9f1+srVHDd6dgUU
yxGsMIkPhg6/C1qeLDZoP+Mv
x4Di3SGIfA7F0r2qYtFeMUqz
/qqY1MRI73DdO8xqnzpxf8eC
qUWlatCAmFmX4EFlv/erGqWN
4fQWZKYlSNXLpvJU9zVWvNyb
T4uhxXXOflN/SYd/F+eqGuYL
7+u7BYc6X4zil7PoQSyEADys
XWflzEWx/o7wc0Xf1Q1gQv5h
BznkGtxvwfuR7dgJBRS9AqdV
xGZkb0TjnxZmuXaCjn7+1uKF
lZIJPZvGv0k4+S3W1tyyA4YY
U+LxzW47ZWLu78E3c8ynoOlO
aYhqdNM4N5wqsyM0dWqnvWh0
g8fJQGPVwB6uuaFPvqQkS1Du
HJZCVq2g7knot3fnZ46mS1dT
ZTqZpXIXqXRuojwc/nGtEZ9z

```
1G6h+UKOndfOOg7/GkQAj0JA
Y2xLY7Cpb5TnpurzqPQjSUPh
k5ZDFzNtD/GCnLGHZJDNBcpN
H5U3wGSbXO50fUiB6hapTLW6
WV6SBO+wiXSBiy/WOsjYM9Bb
b+5lVoJISoxsuGtovaKR5jRu
GJuNYYSNkqjueGclb0HIXo0v
myzu5YCtBDwzTSyKjk9EQW4f
7URXAXvsOa0GAPSsXfC2pHE7
ULtctfq4Cei14YoFAoPx9+Qs
7CvtNPsqj/gyGnqwT0doy9dV
oKpfc+fCf2q5Gto3otaZ8fIi
PAL08gtS8W/tBbZjlWeeaAGm
g5tCjK0wbHJJKfEkbZoxULwO
KfNaSo2v8cz7AqvDvp/1KDL4
e6wr7oJZDt0vMwICE7aBj2wb
DipVNTm2ZcN6/3O5n/VSJQTa
A9q6XWUrcXWRMyDAUv3GmxsX
gTs8PZ/nYZRSoX+2lEtyOwUx
0KNe/vUcKm2mkBnJ55tNnKCN
nmfTCZxYaxcPcF0lZh+aXeH4
qbsxUEWk2coXn0USXltGOb6n
xGPSYZ66wR+6NR1kNsMzC+wv
4ljj9NGkAdOzBa2AzDCnu4+5
kvgKcpFUdho09DQh5KxvQtLL
zoOLJzSZWveEfTPo7pkqRk0J
fD7yOQSud11KPNhlr1vjK/jC
JdyQ+bi8CV4OQJX7jwjc08hd
WS7Q3Nx/2KsI0vaKFY3hIctD
X1keU9qhqv5EFyaZR0tPcNRG
oe1Wh7+89GqdzKtzPM2robwP
tIYbAosuVR+iqIHlH5jQnlsC
TxXZneTAt8oDDlEODqJinug5
rsxPP3AhXpFCH5RPSplhv/Zf
i9eVo76rgb5WeBEkwhK1cICO
ae+JCX7GfFcfcOtHispZtywR
E4Eatc0fj1oJTTN3vYrAcuWb
5U4ANP0P</xenc:CipherValue>
                                    </xenc:CipherData>
                                </xenc:EncryptedData>
                                <ds:DigestValue>MnBkj6wkDEktL9JuRiRwlvID
WN0=</ds:DigestValue>
                            </EncryptedResp>
                    </Body>
</keaml-ke:SA2ResponseMsg>
```

APPENDIX E ACRONYMS

AA – Attribute Authority

ABAC – Attribute-Based Access Control

ISAKMP/IKE – Internet Security Association and Key Management Protocol/Internet Key Exchange

JFK – Just Fast Keying

KEAML – Key Exchange and Authentication Markup Language

KEAML-KE protocol – Key Exchange and Authentication Markup Language – Key Exchange protocol

KEAML/KEAML-KE – Please refer KEAML and KEAML-KE protocol. The term KEAML/KEAML-KE, which is similar to ISAKMP/IKE, emphasizes that KEAML-KE is the protocol that is derived from KEAML framework, which is extensible.

MPS – Meta Policy Server

PFS – Perfect Forward Secrecy

PRF – Pseudo Random Function

RBAC – Role-Based Access Control

SA – Security Association

SSL – Secure Sockets Layer

XACML – eXtensbile Acess Control Markup Language