Multi-party authentication protocols for web services

Zhang, Dacheng

Multi-Party Authentication Protocols for Web Services

Dacheng Zhang

M.Sc. Thesis

Department of Computer Science,
University of Durham
UK

2003

Abstract

The Web service technology allows the dynamic composition of a workflow (or a business flow) by composing a set of existing Web services scattered across the Internet. While a given Web service may have multiple service instances taking part in several workflows simultaneously, a workflow often involves a set of service instances that belong to different Web services. In order to establish trust relationships amongst service instances, new security protocols are urgently needed.

Hada and Maruyama [HAD02] presented a session-oriented, multi-party authentication protocol to resolve this problem. Within a session the protocol provides a common session secret shared by all the service instances, thereby distinguishing the instances from those of other sessions. However, individual instances cannot be distinguished and identified by the session secret. This leads to vulnerable session management and poor threat containment.

In this thesis, we present a new design for a multi-party authentication protocol. In this protocol, each service instance is provided with a unique identifier. The Diffie-Hellman Key Agreement scheme is employed to generate the trust relationship between service instances within the same flow. The Coordinated Atomic Action scheme is exploited for achieving an improved level of threat containment.

The new protocol was implemented in Java and evaluated by a combined use of experiments and model-based analysis. The results show that the time consumption for multi-party authentication increases linearly as the number of service instances that are introduced into a session increases. Our solution is therefore potentially applicable for Web service flow with a large number of participants. Various public key algorithms are also compared and evaluated during the experiments in order to select the most suitable one for our new protocol.

Key words: Atomic actions, authentication, fault tolerance, Internet computing, key exchange, security, Web services

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

1.1 Background

Web services are self-contained, self-describing, modular applications that can be published, located and invoked across the Web [VAS01]. Although this new technology was only proposed several years ago, it has already created great interest within the Computer Science community. The emergence of Web services is promoting the development of dynamic e-businesses such as virtual-corporation [BER02] and E-market [YAN02]. Through the use of protocols such as XML, SOAP, WSDL and UDDI, Web services can contact each other dynamically over the Internet and enable users to build flexible business solutions.

Web services provide options for both suppliers and users. Web service suppliers can dynamically cooperate with other suppliers to compose and provide larger Web services, while users can select their desired Web services scattered across the network and organize them together to accomplish their goals. For instance, a travel agency service may need to contact both an airline service and a hotel service in order to book a client's holiday. Understandably, the combination of Web services can be very complex in many cases. Thus, a simple but structured means is required to describe such combinations of services. At the moment, there are some specific languages, e.g., WSFL [LEY01] and XLANG [THA01], proposed for describing workflows (Web service combinations). We use the term *workflow* to describe a business process that is automatically executed and managed by computer systems. There exists also some work on Web services flow control (e.g., [KIM00, YAN02]). However it is the behavior of multiple parties participating in a workflow that introduces new security challenges.

1.2 The Problem

Essentially, a session is a lasting interaction between system entities, often involving a user, typified by the maintenance of some state of the interaction for the duration of the interaction [HOD02]. In the traditional Client/Server model, a session generally only consists of two partners (client and sever). In the peer-to-peer model of Web services, the conditions are quite different.

A Web service is a static, long-lived entity with a unique identifier all over the world (e.g., its URL address). When a Web service receives an initial request to participate in a business flow (called a session), it will invoke a new instance to handle the requests that are pertinent to this particular business flow [HAD02]. However, a Web service instance is a dynamic, transient entity usually without a unique identifier. This is because a business flow may involve multiple instances that belong to different systems and organizations, while a Web service may have multiple instances that take part in different workflows simultaneously. So an authentication mechanism is needed to make sure that an instance in one Web service business flow is communicating with other instances that take part in the same business flow. The authors of [HAD02] present an initial session authentication protocol for establishing a trust relationship between instances of a business flow. But as the authors have stated in their work, there are several related issues in the area of Web service session control that are as yet unresolved, such as when and how to end a session. At the moment, this field has not been fully explored.

1.3 Main Results

This research is concentrated on the analysis of the issues regarding session control of Web services, as well as exploring different ways to design and develop a multiparty session authentication system for Web services. Models for different solutions are generated respectively to evaluate their performances. In the final solution, a trusted third party session authority is leveraged to manage the session, and each instance in a session is associated with a unique identifier. Furthermore, a session management protocol is designed and a CA (Coordinated Atomic) action mechanism is employed to enhance its ability of attack confinement.

In order to examine our idea in practice, a proto-type of the Instance ID authority system is implemented in Java. Various asymmetric algorithms are employed in the experiment system in order to compare their performances.

1.4 Organization of the Thesis

Chapter 1 introduces an overview of the research.

Chapter 2 gives an overview of Web service architecture and related security issues. First of all, the Web service model and related specifications are introduced. After the introduction, a discussion of the new security challenges that Web services must face is provided. Among these challenges, the security requirements for session authentication are particularly important. This new issue is related to strong flow ability of Web services. After discussing the limitation of existing solutions in session management and related fields, we conclude that a new session security management scheme for Web services is needed.

Chapter 3 introduces the structure of our new session authentication system. The chapter describes the solution in [HAD02] and its drawbacks followed with the introduction of the session authority and the CAA (CA action) manager.

Chapter 4 contains the results and analysis of the experiment performed as well as the analysis of the model of the system. The results of the experiment will then be compared with those generated by the model. Also, we present another decentralised plan of multi-party authentication system for Web services, which was considered when we designed our protocol. This plan does not rely on any trusted third party. Using formal analysis, we compare this solution with the centralised system described in Chapter 3.

Chapter 5 gives the conclusion and discusses future work.

Chapter 2 Security Issues with Web Services

Since the introduction of Web services several years ago, the IT industry has realized that Web services enable dynamic and flexible enterprise applications. A corporation can employ numerous Web services from different providers, which are implemented with different languages, and executed on different platforms to generate a single unified application. Though few doubt the potential of Web services, the security issue is currently a major obstacle preventing the adoption of this new technology. In this chapter, we will introduce the situation surrounding Web service security.

2.1 Web Services

2.1.1 Service-Oriented Architecture (SOA)

In the traditional e-business way, a corporation that wants to accomplish a business goal has to buy the related software, normally built as a standard product, from the supplier. Also, corporations have to constantly maintain and update the software by themselves. So the way that different corporations implement their e-business systems is ad hoc. Furthermore, it is notoriously complex, costly, and time-consuming to integrate these heterogeneous business information systems. Under the pressures of competition, corporations need a more effective and flexible model to reduce their costs and improve their competitiveness. The concept of a Service-Oriented Architecture is put forward under this background. A Service-Oriented Architecture is essentially a collection of services that communicate with each other through a standard means. The communication can involve either simple data passing or multiple service coordination. In this model, an application can be wrapped within a well-defined interface. In other words, this application is wrapped into a service. This interface hides all the details of the application (e.g., language, operation system, database, etc.) The user can access this application through the interface without knowing any details about its implementation.

Service-oriented architectures are not a new thing. Early attempts at SOA such as the Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA) led to the realization that the future business logic of an application is not necessarily coupled with the present logic [BEQ02]. Corporations can select and change their business partners dynamically as well as reduce software maintenance costs.

2.1.2 Web Services

According to IBM, Web services are a form of Web applications that are self-contained, self-describing and modular. These Web applications can be published, located, and invoked across the Web. Web services perform functions ranging from simple requests to complicated business processes. Once a Web service is deployed, other applications (and other Web services) can discover and access it dynamically [TID00]. A Web service is described using a standard, formal XML notion, called the service description. It includes all the details necessary for interacting with the service, including message format (that details the operations), transport protocols, and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware and software platform it is executed in and the programming language it is implemented in. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services can fulfil a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction [KRE01]. Currently, multiple major software companies have proposed their own Web services plans: Microsoft's .Net, Sun's Sun ONE (Sun Open Net Environment) and so on. These plans are based on several common technologies: UDDI [GIB01], WSDL [CHR01], SOAP [BOX00] and XML [COW02].

Web service technology is independent of any transporting protocols. Although HTTP is the de facto standard network protocol for Internet-available Web Services, other Internet protocols such as SMTP and FTP can also be supported. In Intranet domains, different reliable messaging and call infrastructures (e.g., MQSeries, CORBA) can also be employed. The network technology can be chosen based on various requirements including security, availability, performance, reliability and so on.

2.1.3 Web Service Architecture

According to [KRE01], there are three main roles in the Web Services architecture: service provider, service requestor and service registry. The service provider is the owner of the service as well as the platform that hosts access to the service. The service requestor is the entity that discovers and invokes required Web services. A service requestor can be a person or Web service. Lastly, the Web registry is a searchable registry of service descriptions where service providers publish their service descriptions.

The interaction between these three roles involves the operations *publish*, *find* and *bind* as follows:

- *Publish*: To be accessible, a service description needs to be published so that a service requestor can find it. Where it is published can vary depending on the requirements of the application.
- *Find*: In the find operation, the service requestor retrieves a service description directly or queries the service registry for the type of the service required.
- *Bind*: Eventually, every service needs to be invoked. In the bind operation the service requestor can invoke a Web service statically (This means the service is contacted through the pre-defined code) or initiate an interaction with the service dynamically at runtime, that is, uses the binding details in the service description to locate, contact, and invoke the service(s).

Figure 1 shows the overview of the Web Service architecture.

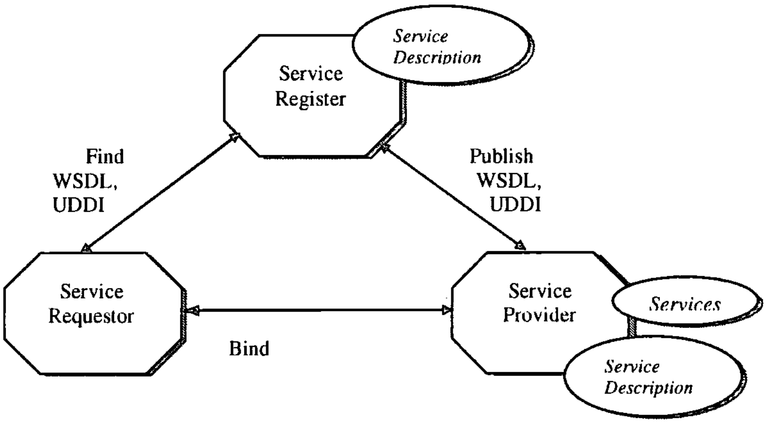


Figure 1 Web services roles, operations and artefacts [KRE01]

In the Web service architecture, there are two artefacts: *Service* and *Service description* (presented as ellipses in Figure 1) [KRE01].

- *Service*: A service is a software module deployed on network accessible platforms provided by the service provider. It exists to be invoked by or to interact with a service requestor. It can also function as a requestor, using other Web Services in its implementation.

□ *Service Description*: The service description is a machine-processable specification of the Web service's interface [BOO03]. It contains the useful information such as data types, operations, binding information, network location, categorization, and other meta-data which are necessary for service requestors to locate and utilize their favourite services.

2.1.4 Web Services Protocol Stack

2.1.4.1 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a lightweight protocol for the exchange of information in a decentralized, distributed environment [BOX00]. It is an XML based protocol and provides a mechanism for exchanging structured and typed information between peers. As with the illustration of Figure 2, a SOAP message consists of three parts:

□ *Envelope*, which is the root element of every SOAP message and marks the beginning and the end of a SOAP message.

□ *Header*, which is an optional element and can be used to describe attributes of the message or the operations that should be done to the message (e.g., the way that a recipient of a SOAP message should process it).

□ *Body*, which is required in every SOAP message and provides a simple mechanism for exchanging mandatory information intended for the ultimate recipient of the message [BOX00].

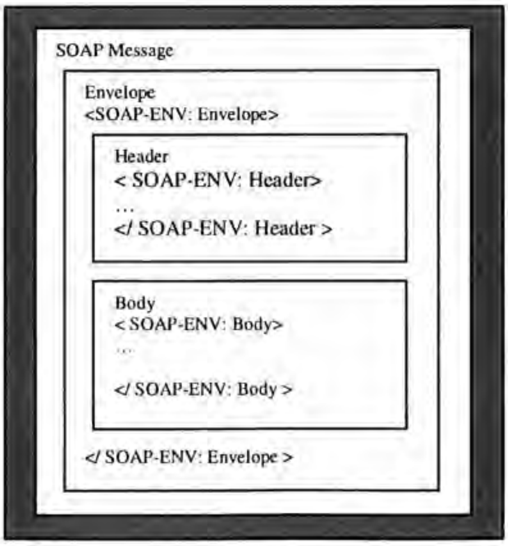


Figure 2 Anatomy of SOAP messages [BEQ02]

Figure 3 and Figure 4 present a simple conversation between a requestor and a Web service **HelloWorld**. Firstly, the requestor sends a message to **HelloWorld**. The SOAP message of this request is described in Figure 3. This message implements a RPC call to the sayHello( ) method of the Web service, and the parameter of this method is a string "Dacheng".

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
 <soapenv:Body>
  <ns1:sayHello soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:helloworld">
   <arg0 xsi:type="xsd:string">Dacheng</arg0>
  </ns1:sayHello>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 3 Example of SOAP messages (1)

The message presented in Figure 4 is the response from the Web service to the requestor. The `<sayHelloReturn>` element includes the replied string, "Hello Dacheng!"

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
 <soapenv:Body>
  <ns1:sayHelloResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="urn:helloworld">
   <sayHelloReturn xsi:type="xsd:string">Hello Dacheng!</sayHelloReturn>
  </ns1:sayHelloResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 4 Example of SOAP messages (2)

2.1.4.2 Web Services Description Language (WSDL)

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [CHR01]. The latest version of the WSDL specification was developed by Web Services Description Working Group and was submitted to W3C as a suggestion in March 2001[CER02]. As Figure 5 illustrates, WSDL specifies a series of elements [CHR01]:

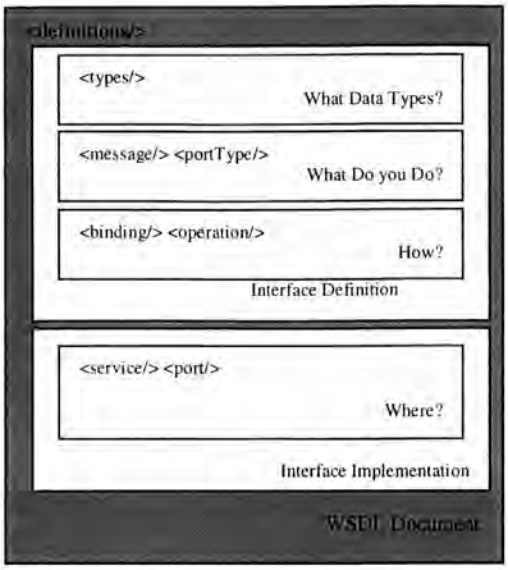


Figure 5 Structure of WSDL [BEQ02]

- □ `Port`, which specifies the Web service's endpoint by combining a binding and a network address.

- □ `PortType`, which is an abstract set of operations supported by one or more endpoints.

- □ `Operation`, which is an abstract description of an action supported by the service. This element defines the name of the function and input/output types.

- □ `Message`, which is an abstract, typed definition of the data being communicated.

- □ `Types`, which is a container for data type definitions using some type system such as XSD.

- □ `Binding`, which is used to attach a specific protocol, data format or structure to an abstract message, operation, or endpoint. It describes how to transmit the operations defined in a `<PortType>` element over the network.

- □ `Service`, which is a collection of related endpoints and specifies where to find the Web service.

2.1.4.3 Universal Description Discovery and Integration (UDDI)

Universal Description Discovery and Integration (UDDI) is a technical specification for describing, discovering, and integrating Web services [CER02]. UDDI enables business organizations to quickly and dynamically publish and discover Web services. It is a critical part of the Web service protocol stack.

Microsoft, IBM, and Ariba announced the first version of UDDI: UDDI 1.0 in September 2000. Since the initial announcement, there are more than 280 companies taking part in the UDDI initiative.

Generally, the UDDI technical architecture consists of three parts [CER02]:

- □ *UDDI data model*, which is an XML schema for describing Web services.
- □ *UDDI API*, which is a SOAP-based API for searching and publishing UDDI date.
- □ *UDDI cloud services*, which are the operation sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

So the UDDI project not only provides a technical specification for building a distributed directory of business and Web services, but also serves as a fully operational implementation of this specification.

As described above, UDDI provides a schema to describe Web services and businesses. We can divide the information captured within UDDI into three main categories [CER02]:

- □ *White pages*, which describe business information (e.g., business name, business description, contact information).
- □ *Yellow pages*, which describe the general services information of the business. Yellow pages may include information on industry, product, etc. based on standard taxonomies.

- ❑ *Green pages*, which describe the technical information of the services. It is notable that UDDI is not only restricted to describing Web services based on SOAP, but can also describe other kinds of service (e.g., CORBA, Java RMI services).

2.1.5 Web Services versus Previous SOA Attempts

Before the occurrence of Web services, there have been three major SOA attempts (J2EE, CORBA and DCOM). Compared with these attempts, Web service has the following advantages [TOO03]:

- ❑ Web service provides a universal standard, WSDL, to describe the service. No mater how the services were implemented, the user can access them with WSDL. Before Web services, there were no universal standards used for SOA. Different attempts came with their own standards. DCOM uses Microsoft-IDL, CORBA uses CORBA-IDL, and J2EE uses Java. Moreover, these standards are incompatible. This increases the difficulty of user integration.
- ❑ Web service separates interfacing from programming. Before a developer programs with J2EE, DCOM, or CORBA, he has to learn their programming model and API sets respectively. On the contrary, a Web service-oriented architecture does not define the way in which the interface is implemented. Thus, there is no need to learn any particular API when Web services integration is supported.

2.2 Web Service Security Challenges

Practically, Web services are loosely coupled, language-neutral, platform independent. And it is well known that Web services have many advantages such as easy enterprise application integration, distributed development, and Business-to-Business e-business implementation. However, Web services also bring many great security risks, which have become the most critical issues that must be addressed before Web services are widely employed in e-business applications. In this section, we discuss some typical risks in this field.

2.2.1 Malicious Attacks

In order to generate a secure environment for Web services, developers have to counter many potential attacks. Generally, there are two main kinds of attacks. One kind of attack utilizes the loopholes and bugs of software to circumvent the access control of the system or force the system to do something improper. Buffer overrun attack [GHO01] and timing attack [RAY00] are representative of this kind of attack.

The other kind of attack compromises the security of the systems by leveraging the algorithms' drawbacks or the security protocols' logic flaws. This kind of attack can be further divided into two categories: passive attacks and active attacks. Passive attacks, also known as eavesdropping attacks, are the simplest way to attack the security system. In this type of attack, adversaries extract useful information from the private conversations of honest entities by eavesdropping on their communication and then snatching the messages that are transported through the network. Eavesdropping attacks are normally used for gathering information for future active attacks. Since the eavesdroppers only monitor the conversations of other entities, it is difficult for the partners of the communications to detect the attack and act accordingly. In comparison with previous Web applications, Web services seem to be more vulnerable to eavesdropping attacks. Since Web services transmit XML documents in which data are well structured and easily understood, it is much easier to retrieve information from these documents than from binary data.

In contrast to passive attacks, active attacks compromise the security of systems by additionally subverting the communications in many ways (i.e., injecting messages, intercepting messages, replaying messages, altering messages) [BLA97]. We review four common active attacks: dictionary attack, replay attack, man-in-the-middle attack, and number theoretic attacks. In addition, we will discuss the potential threat of these attacks to Web services.

1. Dictionary attack is also called guessing attack. In some security systems, users choose the secrets (i.e. password) for authentication. Such secrets are sometimes selected from a relatively small domain of secrets. We call this type of secrets *poorly chosen secrets* or *weakly shared secrets*. With the use of a 'dictionary', an attacker can perform the attack through iterative guessing and verification. Some experiments have proved that this type of attack is surprisingly effective in compromising particular security systems [GON93, WU99]. These attacks can be subdivided into two classes:

- ❑ *Off-line guessing attacks*: An adversary eavesdrops on protocol messages and stores them locally for verification. There is no need for a server to participate in verification, so it is nearly impossible to notice these attacks [KWO97].

- ❑ *On-line guessing attacks*: An adversary attempts to use a guessed password iteratively on-line. The attack is performed by replaying eavesdropped messages or impersonating other clients. Unless a protocol provides the security server with sufficient information to detect authentication failures, the server cannot notice the attack [KWO97].

In Web service environment, Web services generally combine together to form a single application. These Web services may be under the management of different security systems. Thus, a Web service may have to cooperate with partners whose host security systems use password that are vulnerable to dictionary attacks. So, the security of other Web services will be threatened even if there is only one Web service whose host security system is vulnerable to dictionary attack.

2. In most cases, attackers cannot get valuable information directly through eavesdropping on the conversations protected by a security protocol over the network. Sometimes, however, they can replay these messages later to get useful responds for other attacks or impersonate a logical entity accessing security resources. We call this kind of attack as replay attack.

According to different employments of the antique messages, there are two types of replay attacks:

- ❑ *Run external attacks*: Attackers get messages from one protocol run and replay them in another. The Denning-Sacco attack on the original Needham-Schroeder key distribution protocol is a classic example of this type of attack [SYV94, DEN81].

- ❑ *Run internal attacks*: Attackers get messages and replay them within the current protocol run. An example of this type of attack is described in [DEN81].

As we stated, Web services transmit messages in XML documents, and the data in XML document are in well-defined structure. So it is much easier for an adversary to pick up useful information from XML documents intercepted from Internet and re-organize them to implement replay attacks.

3. Man-in-the-middle attack compromises the security of the systems by placing a computer or another device in the middle of the communication chain to steal passwords, keys, or eavesdrop on communications [BRO02]. In this attack, the adversary intercepts some messages of a conversation between two entities and replaces them with his carefully designed messages [ABA97]. After these processes, the adversary generates the communication channel with two entities, and these entities communicate with the adversary respectively. Hence, the adversary can monitor and modify the messages transmitted between the victims.

   Web services work in the peer-to-peer environment. In many cases, the messages sent from a Web service have to traverse through multiple Web services and may be modified as required before it reaches its ultimate destination. Consequently, it is more difficult to verify whether the message obtained is the original one.

4. Number theoretic attack compromises the security of the systems which make use of the improper parameters [PAT97]. Unlike the attacks we mentioned above, which attempt to find and make use of the logic drawbacks of the security protocol, number theoretic attacks employ the knowledge of number theory to narrow the domain of the secret key of the security system and make it easier to guess the secret key. In order to prevent this kind of attack, the designer of the security protocol must be familiar with the features of the security algorithms so that they can select the proper parameters.

   When we discussed the potential threat of dictionary attack above, we mentioned that the security of the integrated Web service application would be compromised even if there were only one participant that is vulnerable to the dictionary attack. For the same reason, the Web services may sometimes suffer from the number theoretic attacks when their participants use improper parameters for their security systems.

Essentially, due to the structure of Web services, Web services have to expose their details to the open network so that both potential users and malicious attackers can get useful information easily. In addition, Web services are generally developed to achieve complex functions. The interface of a Web service may be composed of a number of methods and can be much more complex than previous Web applications. So it is much easier for adversaries to find drawbacks from the interfaces and employ them to compromise the security of the application.

In short, the threat of malicious attacks to Web services is much more serious than those to previous Web applications. Much research is under way in this field. Due to their efforts, some new security protocols have been proposed. We will introduce some of them in later sections.

2.2.2 Firewalls

Web services break the boundary of organizations. In order to simplify the communication among Web services provided by different vendors, Web services normally use port 80 and port 443 to transport messages. The standard perimeter firewall will regard these messages as standard Web traffic and let them pass unexamined - potentially passing malicious threads. Moreover, Web services send their requests and data in standard XML format. Firewalls can do a good job of port monitoring and recognizing brute force malicious attack but are not able to view the content of these messages in order to detect and then prevent more sophisticated security compromises [YAN02 (1)].

2.2.3 Message-Level Security

Web service technology provides an effective way to realize B2B model. Different from traditional B2C model, B2B business processes are more complex, and sometimes it may be necessary for SOAP messages to traverse multiple hops before they get to the intended destination. Traditionally, people employ transport layer security protocols (e.g., Secure HTTP (HTTPS), Secure Sockets Layer (SSL) and Transport Layer Security (TLS)) to protect the integrity and confidentiality of the information [GAR97]. These transport layer protocols can only provide encryption and authentication between a pair of endpoints. Therefore, when people try to use these protocols to transport SOAP messages between Web services, message security may be compromised at the intermediate points. Thus, it is necessary to develop a set of message-level security protocols for Web services.

2.2.4 Exception Handling

Web services can cooperate together to achieve a business goal. The combination of Web services may involve some form of control flows that reflects causal relationships between the invoked services [NAK02 (1)]. Sometimes this kind of service flows consists of large amount of Web services and the relationship among the partners can be very complex. Since the Web services involved within a service flow is normally provided by different vendors and administrated by different organizations, how to handle the exceptions or errors occurred during the process of the service flow is a big issue. For instance, some services within a service flow may work incorrectly due to internal reasons, and some services may not be able to continue their work due to network errors or malicious attacks. Moreover, before the exceptions or attacks are noticed by other Web services within the flow, more Web services may have already been invoked. Consequently, how to synchronize the Web services and roll back the flow to a trusted state is an interesting topic.

Besides the issues we have described above, there are still many interesting subjects in this field, such as how to generate communication between two organizations using different security systems, how to work with an outside vendor that is insecure, how to Web-enable a legacy application that was never designed to be exposed to the public Internet, and so on. Since there are so many security issues associated with Web services, developing new security protocols for Web services is necessary and urgent.

To generate a secure environment for Web services, several basic requirements must be first addressed. We describe them as follows [NAK02]:

- □ *Confidentiality* guarantees that the exchanged information is protected against eavesdroppers.
- □ *Authentication* guarantees that the access to e-business applications and data is restricted to only those who can provide the appropriate proof of identity.
- □ *Integrity* refers to assurance that the message was not modified accidentally or deliberately in transit.
- □ *Non-repudiation* guarantees that the sender of the message cannot deny having sent it.
- □ *Authorization* is the process to decide whether or not the entity can access the particular resource.

In the following section, several current Web service security protocols are discussed in detail, highlighting the implementation relationship between the main requirements mentioned above and the protocols listed below.

2.3 XML Security Specifications

In this section we will introduce several security specifications to meet different XML security aspects. These include:

- □ *XML Digital Signature*, which describes a means to generate and represent digital signatures in XML.
- □ *XML Encryption*, which specifies a means of encrypting data and presenting it in XML format.
- □ *Web Services Security*, which expends SOAP specification to enable message integrity, message confidentiality, and single message authentication.
- □ *XML Key Management Specification*, which integrates PKI with Web service technology and specifies protocols for distributing and registering public keys.

2.3.1 Canonical XML

Before we delve into the details of XML signature and XML encryption, we will first introduce Canonical XML. If people leverage some kinds of security algorithms (e.g., Encryption algorithms, MAC algorithms, digital signature algorithms) to provide the confidence of the integrity of a message, the slightest change to that message will result in totally different values. It is fine to apply this feature of the security algorithms used to protect the integrity of messages for normal use. But in the case of XML, the condition is a little more complex.

In the context of XML documents, two documents in different textual representations may still have the same content. Obviously, the syntax variants that do not cause any logical change do not imply that the integrity of the XML document or the authentication of its sender is suspicious.

Canonical XML specification describes a method for generating a physical representation, the canonical form, of an XML document that accounts for the permissible changes [BOY01]. In both XML encryption and signature specifications, security functions work on the canonical form directly. Therefore, the results are the same on two logically identical documents, even though their physical structures may be different.

2.3.2 XML Signature

XML signature specifies XML syntax and processing rules for creating and presenting digital signatures [BAR01]. The digital signature of a message can help the receiver ascertain the identity of the originator. Furthermore, Digital signatures can guarantee the non-repudiation and freshness of the messages.

XML signature can be applied to both binary data and octet data. It also provides various ways to represent the signatures. The signature can be a part of the XML document or detached from the data that is signed.

The XML signature specification makes use of the `<Signature>` element, which has the following structure, to represent signatures in XML format [BEQ02].

```
<Signature ID>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    <Reference URI >
      <Transforms/>
      <DigestMethod/>
      <DigestValue/>
    </Reference>
  </SignedInfo>
  <SignatureValue>
 <KeyInfo>
 <Object ID>
</Signature>
```

Among the elements within the `<Signature>` element, `<CanonicalizationMethod>` element specifies the canonicalization algorithm used to canonicalize the XML document. `<SignatureMethod>` element identifies the cryptographic functions used to generate the signature. `<DigestMethod>` element defines the digest algorithm (e.g., HMAC, SHA-1, etc.)

to be applied to the signed object. This kind of algorithms is also called message authentication codes (MACs), seals, integrity check values, or message integrity codes (MICs). It can ensure of the integrity of the message received.

The code below present an instance of XML signature [BAR01].

```
<Signature Id="MyFirstSignature"
xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
   <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
   <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"/>
   <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
     <Transforms>
       <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
     </Transforms>
     <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
   </Reference>
 </SignedInfo>
 <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
 <KeyInfo>
  <KeyValue>
    <DSAKeyValue>
      <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
    </DSAKeyValue>
  </KeyValue>
 </KeyInfo> </Signature>
```

2.3.3 XML Encryption

XML encryption is proposed by the W3C XML Encryption Working Group [IMA02]. This specification defines the process for encrypting data and representing the result in XML. Encryption functions normally serve to improve confidentiality, guaranteeing that the exchanged information is protected against eavesdroppers. The encryption of the message may also indicate the message's originator [GOL96]. There are two main types of encryption algorithms: symmetric encryption algorithms (conventional encryption algorithms) and asymmetric algorithms (public-key encryption algorithms). Conventional encryption algorithms encipher and decipher the information with the same secret key while public-key encryption algorithms use a pair of keys (one for enciphering and one for deciphering).

The encrypted data may be XML documents, XML elements, XML element contents, or any arbitrary data. The encrypted information is enclosed within the `<EncryptedData>` element.

This element will be inserted to the XML document and take place the encrypted content [BEQ02].

The basic structure of the <EncryptedData> element is presented as follows [IMA02, BEQ02]:

```
<EncryptedData Id Type MimeType Encoding?>
  <EncryptionMethod/>
  <ds:KeyInfo>
    <EncryptedKey>
    <AgreementMethod>
    <ds:KeyName>
    <ds:RetrievalMethod>
    <ds:*>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>
    <CipherReference URI?>
  </CipherData>
  <EncryptionProperties>
</EncryptedData>
```

The <EncryptionMethod> element indicates the security algorithm applied to encrypt the data. The <CipherData> element indicates the encrypted data. And the <EncryptedKey> element is used to transport encryption keys from the originator to a recipient. The key value is always encrypted to the recipient [IMA02].

Following is an example [IMA02] of encrypting an element of a XML document.

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

The XML document above illustrates the details of a credit card, which includes card number, issuer, expiration data, etc. After encrypting the <CreditCard> element, the message should be as follows:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
```

```
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <CipherData>
      <CipherValue>A23B45C56</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

It is notable that `<CreditCard>` element is taken place by `<EncryptedData>` element.

2.3.4 Web Services Security (WS-Security)

To meet the various aspects of Web service security requirement, IBM, Microsoft, and Verisigin proposed a set of security specifications [BEQ02]. WS-Security specification is one of them (see Figure 6). WS-Security describes enhancements to SOAP in providing quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies [ATK02].

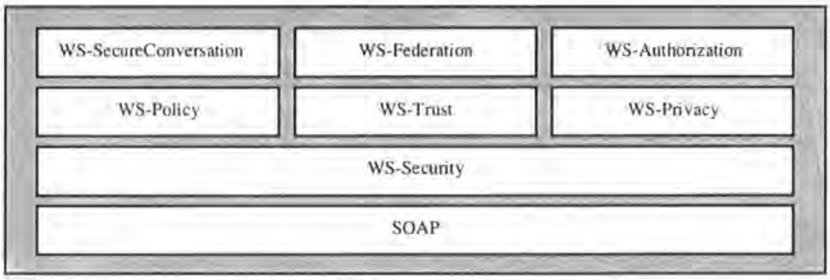


Figure 6 Web services security specifications [BEQ02]

These specifications include [BEQ02]:

- □ WS-Policy: describes the capabilities and constraints of the security (and other business) policies on intermediaries and end points.
- □ WS-Trust: describes a framework for trust models that enables Web services to interoperate security.
- □ WS-Privacy: will describe a model for how Web services and requesters state privacy preferences and organizational privacy practice statements.
- □ WS-SecureConversation: describes how to manage and authenticate message exchanges between parties including exchanging security context

and establishing and deriving session keys. The session mentioned here only consists of two partners.

- □ WS-Federation: describes how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.
- □ WS-Authorization: will describe how to manage authorization data and authorization policies.

WS-Security is the foundation to implement the specifications described above. It provides an extensible mechanism to associate security tokens such as certificate of X.509 and ticket of Kerberos with SOAP messages. Kerberos and X.509 [MEN95] are the most common authentication systems and have been employed in many fields for a long time. Also, WS-Security leverages XML signature and XML encryption to confirm the integrity and confidentiality of SOAP messages.

Overall, WS-Security is a message-level security protocol, and it provides the basic support to integrate Web service technology with traditional authentication systems and generate various security solutions for Web services.

2.3.5 XML Key Management Specification (XKMS)

The WS-Security protocol provides facilities to transport encoded binary security tokens and makes the initial effort to integrate traditional authentication systems with Web service technology. Furthermore, the XML Key Management specification provides an interface between PKI and Web services in order to simplify PKI deployment and achieve seamless-integration between PKI and Web services.

XKMS consists of two parts: the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS). X-KISS allows a client to delegate part or all of the tasks required to process XML Signature `<ds:KeyInfo>` elements to an XKMS service, while X-KRSS is a protocol to support the registration of a key pair by a key pair holder, with the intent that the key pair subsequently be usable in conjunction with the XML Key Information Service Specification or a Public Key Infrastructure [HAL03].

One of the advantages of XKMS is that key management functions are achieved with Web services. XKMS provides a PKI trust service and removes the need for client support of PKI features, so the complex certificate processing logic is abstracted from the applications and becomes a server side component [NAK02].

The Web service security protocols that we described above are important in integrating traditional cryptographic technologies and authentication systems with Web services. But according to our research, we found that in some aspects, these traditional solutions are insufficient to fulfil the security requirements of Web services. In this thesis, we discuss a new security issue which is related to the strong workflow ability of Web services. In the next section, we will provide an overview of this security issue and discuss the limitations of traditional solutions in this case.

2.4 Multi-Party Authentication for Web Services

One desirable feature of Web services is their strong workflow ability. However, more work is needed to enhance the security of Web service flows. In this section, we will discuss this topic. Before we discuss the security issues of Web service flows, we will first take a glance at Web services flow language (WSFL), as it is a specification used to describe the complex combinations of Web services in XML.

2.4.1 Web Services Flow Language

WSFL language, proposed by IBM recently, specifies a set of XML grammars to describe the software workflow processes within the framework of Web service architecture. According to different aspects of Web services compositions, WSFL provides two types of composition models: Flow Models and Global Models [LEY01]. In the Flow Models, a composition, also known as a flow composition, describes how to use the functionality provided by the collection of composed Web services. In the Global Models, the composition describes how the Web services interact with each other rather than specifying the execution sequence. WSFL supports recursive composition, that is, every Web service composition can be regarded as a new Web service in a larger composition.

The schema syntax for the <flowModel> element is provided in the following code sample [LEY01]:

```
<complexType name="flowModelType">
```

```
  <sequence>
      <element name="flowSource"
               type="wsfl:flowSourceType"
               minOccurs="0"/>
      <element name="flowSink"
               type="wsfl:flowSinkType"
               minOccurs="0"/>
      <element name="serviceProvider"
               type="wsfl:serviceProviderType"
               minOccurs="0″ maxOccurs="unbounded"/>
      <group ref="wsfl:activityFlowGroup"/>
  </sequence>
  <attribute name="name" type="NCName" use="required"/>
  <attribute name="serviceProviderType" type="Qname″/>
</complexType>
<group name="activityFlowGroup">
  <sequence>
      <element name="export" type="wsfl:exportType"
               minOccurs="0" maxOccurs="unbounded"/>
      <element name="activity" type="wsfl:activityType"
               minOccurs="0″ maxOccurs="unbounded"/>
      <element name="controlLink" type="wsfl:controlLinkType"
               minOccurs="0" maxOccurs="unbounded"/>
      <element name="dataLink" type="wsfl:dataLinkType"
               minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</group>
```

As with the presentation of the code above, the flow model is defined using six different elements [LEY01]:

- The `<flowSource>` and `<flowSink>` elements define the input and output of the flow model.

- The `<serviceProvider>` elements represent the services participating in the composition.

- The `<activity>` elements represent the usage of individual operations of a service provider inside the flow model.

- The `<controlLink>` and `<dataLink>` elements represent control and data connections among activities in the model.

Therefore, the flow model is sufficient in describing the relationship among Web service partners within a workflow. In a Web services composition such as the one presented in Figure 7, multiple participants collaborate following some policy. These participants normally do not contact with each other until it is necessary. Furthermore, these participants provided by different vendors, developed in different languages, executed on different platforms, are combined dynamically at run time. The ability of flexible and dynamic binding is one of the great advantages of Web services. Nevertheless, it also puts forth new challenges in the form of flow security management. Multi-party authentication issue for Web services is the major one among them.

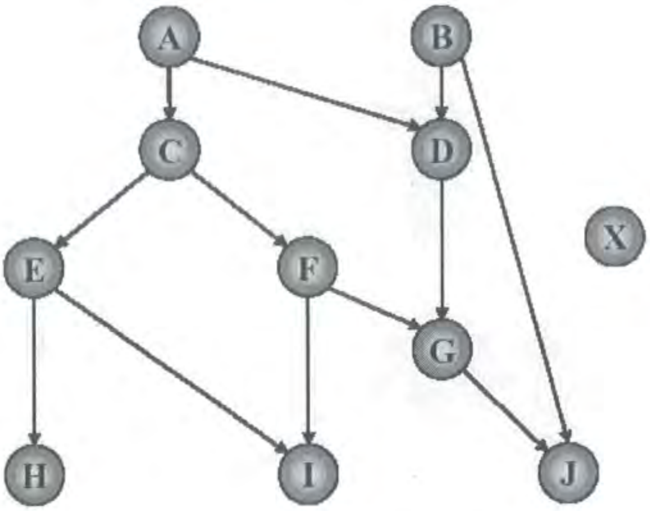


Figure 7 Interaction between multiple Web service instances within a flow [LEY01]

2.4.2 Multi-party Authentication for Web Services

Since neither XML-RPC nor the SOAP specification provides a mechanism to keep the session state, how to maintain the Web service session state based on a stateless protocol such as HTTP is a problem that Web service developers have to face. In fact, this stateless issue is a problem that has intrigued Web application developers for a long time. There have been a number of clever means used to get around it (e.g., using HTTP cookies to preserve the state of a series of requests, reposting application data with each request). Recently, work has been done to use these measures in Web services and some progresses have been achieved in this area. For instance, ASP.NET provides a `System.Web.SessionState.HttpSessionState` class; the Apache implementation of SOAP (i.e., ApacheSOAP) in combination with Java and a suitable application service (e.g., Apache Tomcat Server) can use cookies to help every user find corresponding service instance. However, these measures are not designed generally for managing session state in Web service applications, but rather specifically for managing the session state in the applications following their specification (e.g., ASP.NET applications). The worst part is that using them ties you to the HTTP protocol. The SOAP protocol is designed to work independently of the transport protocols. So tying your application to HTTP obviously limits your flexibility and may create additional work if you want to provide Web services over any other transport protocol other than HTTP (e.g., SMTP). Another drawback of these solutions is that they don not provide any encryption protect to the session IDs. These IDs are transported over Internet in plaintext or inserted into the URL address directly. Therefore, the system needs additional measures to validate the identity of the service requestor.

Furthermore, in the traditional client-server model, there is usually more than one client contacting the server at the same time. Therefore, the server needs to distinguish one client from another. In this case, a "session" is a single client's conversation with a logical server. But as e-business becomes more and more complex, especially with the advent of Web service, it is normal that several organizations cooperate together to achieve a business goal. In this case the traditionally session seems insufficient in expressing the complex relationships among the partners within a business flow. Hada and Maruyama [HAD02] describe the idea of *multi-party* session in their paper. They defined a session in the Web service world as a series of operations executed by Web service instances that need to share a common state, and listed some new issues in this field. As with their definition, a session is in fact a Web service flow instance.

Based upon the above discussion, it is easy to see that a multi-party authentication mechanism is needed to protect the security of Web service flows. With it, a participant within a session can prove its legal identity as a session partner to other participants within the same session in a secure way.

In the following sections, we will introduce the conventional session management within the Kerberos authentication system and group key management algorithm for peer-to-peer group, which is a potential solution for multi-party session authentication issue we discussed above.

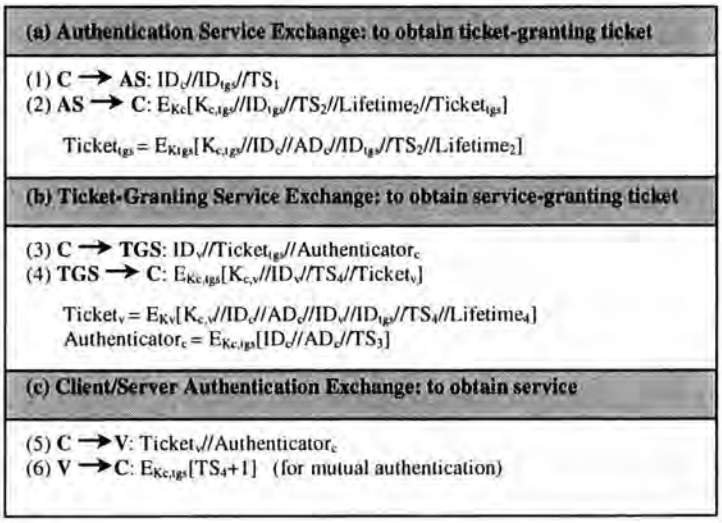
2.4.2.1 Session Management in Kerberos

To facilitate the illustration of Kerberos, we introduce following notations firstly.

C	A client principal
V	A service principal
TGS	A Ticket-Granting Server
AS	An Authentication Server
K_x	Secure Key held by party X
$K_{a,b}$ ($K_{A,B}$)	Session Key for party A and party B
E_K[X//Y]	Encrypted message of information A and information B with the key K
TS	Time Stamp

Kerberos [KAU02] is a famous network authentication protocol developed by MIT. The purpose of this protocol is to provide authentication for users and services. The Kerberos system is based on trusted third-party security servers, the authentication server (AS) and the ticket-granting server (TGS) [BEL91]. All information of the entities under their management is stored in the database. In Kerberos, every principle (user or service) holds a key, which has been registered with the AS. The users' keys are derived from their passwords, while the services' keys are selected randomly. Furthermore, a kind of short time secret key - session key is employ in Kerberos to secure the communications within the session.

As Figure 8 details, Kerberos assigns and distributes a session key for a particular session with a six-step conversation among the client, security servers and the service.



(a) Authentication Service Exchange: to obtain ticket-granting ticket

(1) **C** → **AS**: $ID_c//ID_{tgs}//TS_1$
(2) **AS** → **C**: $E_{Kc}[K_{c,tgs}//ID_{tgs}//TS_2//Lifetime_2//Ticket_{tgs}]$

$Ticket_{tgs} = E_{Ktgs}[K_{c,tgs}//ID_c//AD_c//ID_{tgs}//TS_2//Lifetime_2]$

(b) Ticket-Granting Service Exchange: to obtain service-granting ticket

(3) **C** → **TGS**: $ID_v//Ticket_{tgs}//Authenticator_c$
(4) **TGS** → **C**: $E_{Kc,tgs}[K_{c,v}//ID_v//TS_4//Ticket_v]$

$Ticket_v = E_{Kv}[K_{c,v}//ID_c//AD_c//ID_v//ID_{tgs}//TS_4//Lifetime_4]$
$Authenticator_c = E_{Kc,tgs}[ID_c//AD_c//TS_3]$

(c) Client/Server Authentication Exchange: to obtain service

(5) **C** →**V**: $Ticket_v//Authenticator_c$
(6) **V** →**C**: $E_{Kc,tgs}[TS_4+1]$ (for mutual authentication)

Figure 8 Summary of Kerberos version 4 message exchange [STA98]

Assume **C** intends to access **V,** which is under the management by Kerberos. Firstly, **C** sends a message to **AS** (see Step 1 in Figure 8). This message includes the identity of **C**, the identity of **TGS** which **C** should contact in the future and a time stamp TS_1, which allows **AS** verifying that the clock of **C** is synchronized with **AS** [STA98]. **AS** responds to **C** with a message, which is encrypted with K_c (see Step 2). This message includes a key $K_{c,\ tgs}$, the authentication information $Ticket_{tgs}$ for **TGS,** the time-stamp TS_2 and some other additional information (e.g., ID_{tgs}, $Lifetime_2$). Such information proves that the $Ticket_{tgs}$ is for **TGS** and prevents replay attack. Since K_c is shared by **AS** and **C** secretly, this message cannot be decrypted by any clients other than **C**.

After verifying the message received from **AS**, **C** contacts **TGS** asking for the key to access **K** (Step 3). **TGS** decrypts $Ticket_{tgs}$ with the key that it shares with **AS** and extracts $K_{c,\ tgs}$ from $Ticket_{tgs}$. Additionally, with other information in $Ticket_{tgs}$ (i.e., IDc, TS_2) **TGS** can examine the validity of this ticket. Afterwards, **TGS** uses $K_{c,\,tgs}$ to decrypt Authenticator$_c$ and compare the information regarding user **C** in Authenticator$_c$ with the related information in the Ticket$_{tgs}$. If all the examination is successful, **TGS** sends the reply back to **C** (see Step 4) and this reply is encrypted with the session key $K_{c,\,tgs}$ to protect the confidentiality of the message. At this stage, **C** shares the session key $K_{c,\,v}$ with **V**. Steps 5 and 6 achieve a mutual authentication for **C** and **V**. After this process, **C** can ensure that **V** has gotten $K_{c,\,v}$, and vice versa.

In brief, Kerberos is an authentication system designed for the Client/Server model, and the session in Kerberos only consists of two session partners [HAD02]. So Kerberos cannot be employed directly to provide the authentication for the session partners within a multi-party session.

2.4.2.2 Group Key Generation Algorithms

Traditional authentication systems (e.g. Kerberos, SSL) are commonly based on the Client/Server paradigm. They can help the two entities at both ends of the conversation to generate some kind of trust relationship. On the other hand, many emerging network applications (e.g., online games, distributed simulations, conferencing) require multiple entities dynamically composing and working together. We call this kind of applications as dynamic peer groups.

Group key generation algorithms can generate and distribute security keys for dynamic peer groups. In a dynamic peer group, the group membership is not known in advance, that is, the parties may join and leave the multicast group at any given time [BRE01]. In this case, dynamic peer groups share identical feature with Web service flows. Therefore, it is reasonable to take dynamic peer group authentication protocols into consideration as potential candidates to improve the security of Web service flows. Normally, there are two main ways to generate and manage a group secret:

- Centralized: the group key is entirely generated by a single party who then distributes it to all other group members [MAI00]. To implement a centralised key management system, we can assign a Trusted Third Party (TTP), which is something like the authority server in the Koreros system, or fix a certain partner of the group to generate and distribute the group secret. The session authentication system described in [HAD02] is a centralised key management system, which relies on the TTP [MAI00], and we will discuss its drawbacks in the next chapter.

- Contributory: each group member makes an independent contribution to the group key [STE98]. Furthermore, there are two kinds of Contributory key management protocols, which have slightly different flavours [STE98].
  - *Partially Contributory*: some operations are contributed to by each member and others are centralized.
  - *Fully Contributory*: all key agreement operations are contributed to by each group member.

Let $M=\{M_1, \ldots, M_n\}$ be a set of users wishing to share a key S_n. The GDH.2 protocol executes in n rounds. In the first stage (n-1 rounds) contributions are collected from individual group members and then, in the second stage (n-th round) the group keying material is broadcast. The actual protocol is as follows:

Initialisation:

Let p be a prime and q a prime divisor of p-1. Let $\boldsymbol{G}$ be the unique cycle subgroup of $\boldsymbol{Z}^*_p$ of order q, and let α be a generator of $\boldsymbol{G}$.

Round i ($0 < i < n$):

1. M_i selects $r_i \in \boldsymbol{Z}^*_q$ randomly.
2. $M_i \twoheadrightarrow M_{i+1}$: $\{ \alpha^{r1 \ldots rn/ri} \mid i \in [1, i]\}, \alpha^{r1 \ldots rn}$

Round n:

1. M_n selects $r_n \in \boldsymbol{Z}^*_q$ randomly.
2. $M_n \twoheadrightarrow$ All M_i: $\{ \alpha^{r1 \ldots rn/ri} \mid i \in [1, n[\}$

Figure 9 Group Diffie-Hellman (GDH.2) [ATE00]

Figure 9 illustrates a classic group key generation algorithm - Group Diffie-Hellman, which can be used to implement a fully contributory group key management protocol. The Group Diffie-Hellman algorithm extends the Diffie-Hellman algorithm so that multiple entities can use it to generate and distribute a secret key in a decentralised way.

The drawbacks of this algorithm are obvious (in fact, these drawbacks are common for this kind of decentralised group key generation algorithms). It is easy to see that M_n plays an important role in the key generation in GDH.2. It broadcasts the information to all the other group partners and they use the information to generate the group key. That means it must be a reliable and honest party. Otherwise, it can perform attacks without detection by other group members, for example, partitioning the group into two [ATE00]. However, in some case, especially in peer groups, it is difficult to assign a fixed party to act as M_n, unless a TTP is assigned for the group.

Furthermore, huge recourse consumption is a fatal drawback of the Group Diffie-Hellman algorithm. In order to generate a group key within a group with n group partners, all the partners must process for n rounds and exchange $2(n-1)$ messages on the whole. Also, when an entity takes party in or leaves a group, all the group partners must be informed and the group key must be refreshed.

Therefore, this kind of decentralised group generation key algorithms is not suitable for a group whose partners are scattered in a large area, and it is not suitable for the environment in which the communication channel is not secure. Furthermore, the high overhead of message transport makes this kind algorithm only suitable for a group only with limited partners.

2.5 Summary

Web services are new developments based on the idea of SOA. Compared to previous technologies, Web services have many unique characteristics. These characteristics give new challenges to the security systems. Before a new technology is accepted by the industry, the security issues of it must be understood and resolved. In this chapter, we introduced the Web service mechanism and the new security challenges associated with it. In addition, we discussed the new security challenge in session authentication management for Web services, which this paper is concerned with.

It is notable that WSFL only specifies the representation of the Web services which are involved within the flow. WSFL does not describe how to represent the Web service instances within a Web service flow instance since it is impossible to assign related Web services instances before a Web service flow is executed. Nevertheless, when a Web service flow is executed, the entities working within it are Web service instances rather Web services. Therefore, when a Web service flow is in process, an additional mechanism is needed to manage the Web service instances within the flow instance, as WSFL specification does not provide this function.

Based upon the above discussions, we arrive at the conclusion that it is necessary to develop a novel message-level multi-party authentication system for Web services that is independent of transporting protocol. This system should provide unified identifiers to the Web service instances within a session (i.e., session partners) and should enable them to ascertain that it only communicates with the partners within the same session.

In the next chapter, we will first discuss a solution proposed by Hada and Maruyama. We will then introduce our solution, which assigns every partner within the session a unique identifier, and we will describe our efforts to employ CA action to enhance their solution with threat confinement ability.

Chapter 3 Multi-Party Authentication Protocols for Web Services

As stated in the previous chapters, a new session authentication protocol is needed to improve the security of Web service flow. In this chapter, we will first introduce Hada and Maruyama's solution, the first in this field [HAD02]. Next, we will discuss the drawbacks and disadvantages of their solution. Finally, we will introduce our work on improving their solution.

3.1 Hada and Maruyama's Session Authentication Protocol

Hada and Maruyama [HAD02] proposed an approach in order to authenticate session participants without prior knowledge of all parties participating in the session. In their solution there is a TTP, Session Authority (SA), which takes charge of distributing session authentication messages. An instance of SA is associated with a session. The SA instance assigns a session key (i.e. a session secret) to a particular session and distributed it secretly to the service instances of that session. If two instances hold the same session secret, they will be regarded as the participants within the same session. Then, the instances participating in that session can use this session secret to authenticate each other and distinguish them from the service instances involved in other sessions.

The protocol consists of two parts: a message authentication protocol and a session management protocol. The message authentication protocol transports authentication information between session participants, while the session management protocol is in charge of starting, running, and ending a particular session. These two protocols work together to provide basic management of the session.

Figure 10 describes a case [HAD02] of the session management protocol: Online Session Management.

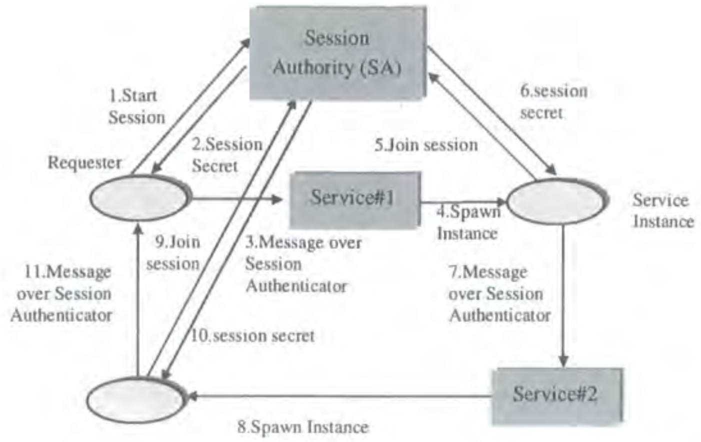


Figure 10 Online Session Management [HAD02]

First, a requester instance initiates a session by sending a `<StartSession>` message to the SA (Step 1). The requester will receive a session secret from the SA (Step 2) and then send an application message to a Web service, **Service#1**, over the session authenticator (Step 3). On receiving the message, the service is given the session handle and asked to join the session. Next, the service creates a new service instance and transfers control to the instance (Step 4). The instance will send a `<JoinSession>` message to the SA (Step 5) and receive later the session secret from the SA (Step 6). It will then have the session secret so as to authenticate any received message and obtain the payload. If the service instance needs to delegate its operation to some other service, it can send a message to another Web service, **Service#2**, using the session authenticator (Step 7). The same process happens in the second service provider (Steps 8-10) and the service instance of the second service provider will then have the session secret. This service instance can send a message to the original requester using the session authenticator (with the same session secret) so that the requester knows that the second service instance is a legitimate session participant.

The case described above needs extra communication for service instances to acquire the session secret. There is an alternative that eliminates this extra communication: Offline Session Management.

As presented in Figure 11, the Requester gets the session secret from the Session Authority, and this secret is transmitted to the new session partners over the network directly. These service instances then use this secret to prove their legal identities as session partners. Compared to Online Session Management, Offline Session Management mechanism is more effective. The drawback of this mechanism is the disability of providing session partners the exact conditions about all the session.

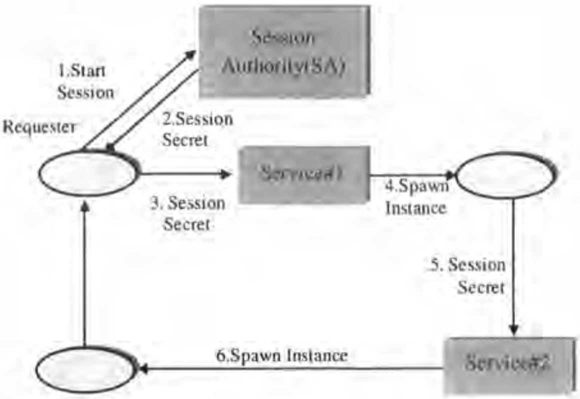


Figure 11 Offline Session Management [HAD02]

From our point of view, this session authentication protocol is not a complete solution. There are still unsolved problems:

- As mentioned in [HAD02], Hada and Maruyama's solution only provides a common session secret to all the participants within a given session rather than a unique identifier for each of them. An attacker, who has compromised a Web service instance and obtained the session secret, can use the secret to communicate with other session participants to gain their trusts. The attacker will then have opportunities to impersonate other session partners. It becomes difficult to confine the damage to a small area, rather than the entire session, even after it was detected that a session participant has been compromised.

- The SA in their protocol does not have a measure to validate the identity of the Web service instance that applies to enter the session. Any Web service instance, as long as it holds the session ID, can contact the SA and apply to enter the session. An adversary may try to attack a session following this way.

3.2 Essential Knowledge of the New Session Authentication System

3.2.1 Key Exchanges of the Diffie-Hellman Algorithm

The Diffie-Hellman algorithm was firstly proposed by Diffie and Hellman in 1976 [STA98, MEN96]. Their protocol enables two users to communicate over public communication channels and safely exchange a key.

The Security of the Diffie-Hellman algorithm relies on the difficulty of computing discrete logarithms.

Consider the equation $y = g^x \bmod p$.

Given g, x and p, we can easily calculate y. On the contrary, given y, g and p, it's difficult to calculate x in general, especially when p is a large prime. As we know, the asymptotically fastest known algorithm [STA98] for taking discrete logarithms modulo a prime number is on the order of

$$e^{((\ln p)^{1/3} \ln(\ln p))^{2/3}}$$

Obviously, it is especially difficult to get x from a given y when p is big number.

We now describe this algorithm:

Assume that there are two principles called Alice and Bob. They agree on a large prime p and an element g ($2 \leq g \leq p-2$), which is a primitive root of p. That is, the numbers $g \bmod p$, $g^2 \bmod p$, $g^3 \bmod p$, …, $g^{p-1} \bmod p$ are some permutation of the distinct integers from 1 to $p-1$. Suppose:

1. Alice chooses a random number x and Bob chooses a random number y from the set $\{1,\dots, p-2\}$.
2. Alice keeps x private and sends $g^x \bmod p$ to Bob.
3. Bob keeps y private and sends $g^y \bmod p$ to Alice.
4. Alice gets the Key = $(g^y)^x \bmod p = g^{xy} \bmod p$
5. Bob gets the Key = $(g^x)^y \bmod p = g^{xy} \bmod p$

6. Alice and Bob then share the same Key = g^{xy} mod p.

If Alice and Bob keep x and y secretly, the only information that an adversary can get are g, p, g^x mod p, and g^y mod p. The adversary has to utilize discrete logarithm to calculate the key. This is computationally difficult, particularly for large primes.

3.2.2 Coordinated Atomic (CA) Actions

A coordinated atomic (CA) action [XU95] is a mechanism for coordinating multi-thread interactions and ensuring a consistent access to objects in the presence of competitive concurrency and potential faults.

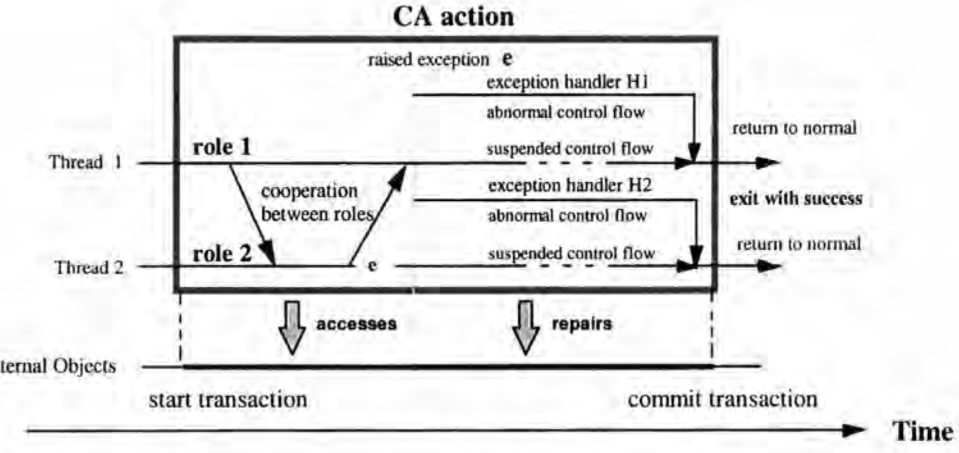


Figure 12 Example of a CA action [RAN99, XU99]

As shown in Figure 12, concurrent execution threads (e.g., Web service instance) participating in a given CA action enter and leave the action synchronously. Within the CA action, operations on objects can be performed cooperatively by *roles* executing in parallel. To cooperate in a CA action a group of concurrent threads must come together and agree to perform each role of the action, with each thread undertaking a different role. Inside a CA action, some or all of its roles can be involved in further (nested) CA actions. If an error is detected inside a CA action, appropriate forward and/or backward recovery measures must be invoked cooperatively by all the roles in order to reach a mutually consistent conclusion. An *acceptance test* can and should be provided in order to determine whether the outcome of the CA action is successful. The external objects which are being competed for must behave atomically with respect to other CA actions and threads so that they cannot be used as an implicit mean of "smuggling" information into or out of a CA action.

3.3 Instance ID Authenticator Protocol

The following notations will be used in the rest of the paper.

n	A big prime number suitable for the Diffile-Hellman protocol
G	A primitive root of n
g^R	The abbreviation of g^R mod n
$K_{x,\,y}$	The security key shared by service instances X and Y
$MAC_{X,Y}(M)$	Message Authentication Code for message M under key K_{xy}
U (R, g^R)	A service instance U whose private message is R and identifier is g^R

Similar to Hada and Maruyama's solution, we present a protocol based on some standard Web services technologies such as SOAP, XML-Signature/Encryption, and WS-security. These protocols can meet certain security requirements in terms of confidentiality, Integrity, and Non-repudiation [NAK02], and we will thus not consider replay attacks and man-in-the-middle attacks in our design.

3.3.1 Operations of Instance ID Authenticator Protocol

In this Instance ID Authenticator protocol, a session manager is a Web service, and it manages a given Web service session. The manager is composed of two parts: a Session Authority (SA) and a CAA manager that manages atomic actions for sessions. The SA is responsible for 1) storing and managing the identifiers of the session participants and 2) providing trusted information for each session participant. With such information, a message sender can specify the intended recipient, and so the message receiver can be sure that the received message comes from a proper sender.

We assume in our protocol that

- According to the Diffie-Hellman algorithm, each service instance of a given session selects a private secret number R randomly, and uses the public key g^R as its identifier. The manager instance of the session guarantees that every identifier is unique within that session.

- The manager instance's identifier is equal to the identifier of the session, and it must be different from those of other manager instances that belong to other sessions.

We present here essential operations for the instance ID Authenticator protocol. Session participants can use them to verify each other's identities. The messages of all the operations in our protocol should be appended with the authenticators that validate the originator and the integrity of the messages unless mentioned otherwise. Details of the authenticator will be described in section 3.3.2.

3.3.1.1 Introduction-of-New-Instance

A session participant can send the "**Introduction-of-New-Instance**" message to its manager instance to recommend a new service instance:

```
<Introduction-of-New-Instance>
 <New-Instance  type="uri"  value="URI  of  the  new  instance"/>
 <NIID>ID of the new instance</NIID>
 .........
</Introduction-of-New-Instance>
```

The `<NIID>` element within the message contains the identifier of the recommended service instance.

3.3.1.2 Identifier-Query

A service instance sends the "**Identifier-Query**" message to the manager instance to check the identifier of another session partner:

```
<Identifier-Query>
 <Instance-under-query type="uri" value="URI of the instance whose
 identifier the initiator instance want to know"/>
 .........
</Identifier-Query>
```

The reply from the manager instance is:

```
<Identifier-Query-Result>
 <Instance-under-query-ID>
      ID of the checked instance
 </Instance-under-query-ID>
 <Instance-under-query type="uri" value="URI of the checked instance
 "/>
 .........
</Identifier-Query-Result>
```

3.3.1.3 Start-Communication

If a service instance wants to start a communication with another instance of the session which it never contacted before, it should send the "**Start-Communication**" message to the intended instance:

```
<Start-Communication>
 <SessionHandle>
  <SA type="uri" value="URI of the SA"/>
            <ID>
                  Session ID
            </ID>
       </SessionHandle>
       <SendingID>
                  ID of the sending service instance
       </SendingID>
 <Sending-Instance type="uri" value="URI of the sending instance"/>
 .........
</Start-Communication>
```

Within the message, the initiator should provide the details of self so that the recipient instance can examine its identity when it is necessary. After the recipient receives and verifies this message, it will send the result back:

```
<Start-Communication-Result>
 <Accept>true</Accept>
  .........
</Start-Communication-Result>
```

If the value of <Accept> element is true, that means the initiator has received the permission to start the communication.

3.3.1.4 Validation

When the recipient instance obtains the "**Start-Communication**" message from the initiating instance, the recipient instance should send the "**Validation**" message to the manager instance in order to verify the identifier of the initiating instance:

```
<Validation>
 <Initiator type="uri" value="URI of the initiator instance"/>
 <Initiator-ID>ID of the initiator service instance</Initiator-ID>
  .........
</Validation>
```

After receiving this message, the SA instance will check the relative information and sends the result back:

```
<Validation-Result>
 <Result>true</Result>
  .........
</Validation-Result>
```

3.3.2 Generating and Appending the Authenticator to Messages

We now describe the details of how to append the identifiers of session participants and other security information to SOAP messages passed between Web service instances with SOAP:

Step 1: A Web service instance prepares a SOAP envelope in order to send a message to participants of a given session.

Step 2: The instance inserts the authentication information (e.g., session handle, the identifier of the sending instance, etc.) into the header of the envelope. An example of the code is as follows:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<SOAP-ENV:Header>
<Session-Authentication:Authentication
    xmlns:Session-Authentication="http://www.durham.com/">
  <Session-Authentication:SessionHandle>
   <Session-Authentication: Manager type="uri" value="URI of the SM"/>
   <Session-Authentication:SessionID>
      Session ID
  </Session-Authentication:SessionID>
  </Session-Authentication:SessionHandle>
  <Session-Authentication:SendingID> The ID of the sending instance
  </Session-Authentication:SendingID>
  <Session-Authentication:ReceivingID>
      The ID of the receiving instance
  </Session-Authentication:ReceivingID>
  </Session-Authentication:Authentication>
</SOAP-ENV:Header>
   <SOAP-ENV:Body>
       ............
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Step 3: The sending instance makes use of the Diffie-Hellman algorithm to calculate a secret key with its private message and the identifier of the intended receiver. It will then utilise XML Signature to apply the Message Authentication Code (MAC), generated with the secret key, to the SOAP message. A code example is given as follows:

```
<?xml version=“1.0” encoding=“utf-8”?>
 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <S:Header>
   ............ Other information about this SOAP message
   <wsse:Security
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    ......
    <ds:Signature>
     <ds:SignedInfo>
```

```
      <ds:CanonicalizationMethod Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
      <ds:Reference URI="#MsgBody">
       <ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>LyLsDDPi4wPU...</ds:DigestValue>
      </ds:Reference>
     </ds:SignedInfo>
<ds:SignatureValue>DJbwhm5gK...</ds:SignatureValue>
     <ds:KeyInfo>
      <wsse:SecurityTokenReference>
      <wsse:Reference URI="#MyID"/>
      </wsse:SecurityTokenReference>
     </ds:KeyInfo>
    </ds:Signature>
   </wsse:Security>
   … same as the message in step 2
  </S:Header>
  <S:Body Id="MsgBody">
         ... ... ... ...
  </S:Body>
 </S:Envelope>
```

Step 4: The sending instance sends the SOAP message to the receiving instance.

Step 5: After the receiver receives the message, it uses its own private key and the identifier of the sender to re-generate the secret key. The receiving instance will then generate the MAC using the secret key. By comparing the newly generated MAC with the MAC appended to the message, the receiver can verify the identity of the sender.

3.3.3 A Scenario

We use in this section a scenario to illustrate the operations of the instance ID Authenticator protocol. Consider a user service instance (UI(X, g^X)) and a session manager service.

First, **UI** contacts the session manager to initiate a session and the session manager invokes a manager instance (MI(Y, g^Y)) to handle **UI**'s request. After **MI** and **UI** exchange their identifiers, g^X and g^Y, a session ($\mathrm{Se}_{(\mathrm{MI})}$) is initiated (see Figure 13).

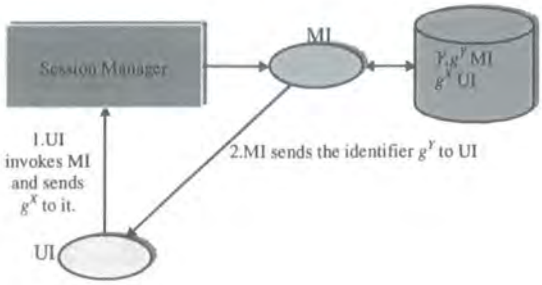

Session Manager
MI
Y, g^Y MI
g^X UI
1.UI invokes MI and sends g^X to it.
2.MI sends the identifier g^Y to UI
UI

Figure 13 Example of the authentication process

Suppose that **UI** wants to contact **Service 1**. And a new service instance (NI(Z,g^Z)) is invoked by **Service 1** to manage **UI**'s request. **UI** and **NI** exchange their identifiers, g^X and g^Z, and some related information (e.g., URL of **MI**, g^Y). **UI** then needs to send an "**Introduction-of-New-Instance**" message to **MI** before **MI** accepts **NI** as a participant of the session (see Figure 14).

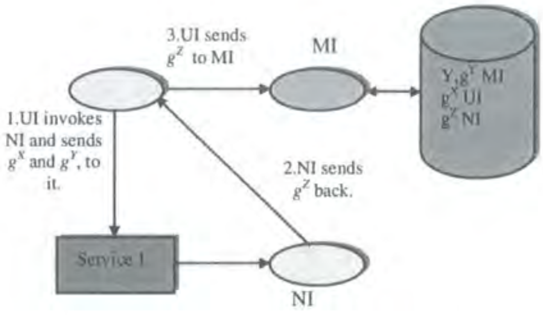


Figure 14 Interactions between UI and NI

Assume that another service instance (NI2(N, g^N)) is invoked and recommended to **MI** (see Steps 1, 2 and 3 of Figure 15). Since **NI2** does not know **UI**'s identifier, it sends **MI** an "**Identifier-Query**" message to check **UI**'s identity in order to communicate with **UI**. After getting **UI**'s identifier from **MI**, **NI2** will send a "**Start-Communication**" message to **UI**. After receiving a message from **NI2**, **UI** needs to contact **MI** to verify the identity of **NI2** since it has not communicated with **NI2** before. If the reply from **MI** is True, **UI** will then verify the MAC appended with the message and send a reply back to **NI2** (see Step 4).

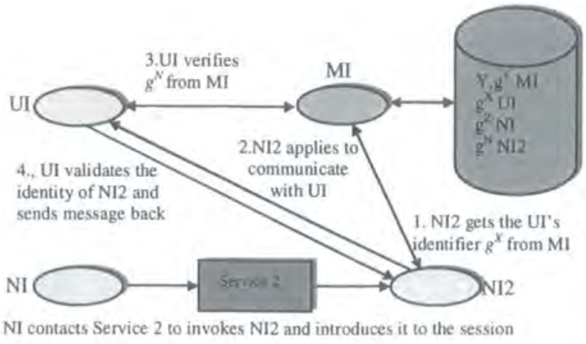

3.UI verifies g^N from MI
MI
UI
Y, g^Y MI
g^X UI
g^Z NI
g^N NI2
2.NI2 applies to communicate with UI
4., UI validates the identity of NI2 and sends message back
1. NI2 gets the UI's identifier g^X from MI
NI
Service 2
NI2
NI contacts Service 2 to invokes NI2 and introduces it to the session

Figure 15 Example of interactions between UI and NI2

3.3.4 Security Analysis

In our protocol, each service instance of a session has a private secret and a public identifier. The public identifier will be transported over the Internet while the private secrets are kept by the instances securely. The attacker cannot get the key shared by the instances participating in the communications from the plaintext and the MAC code [SCH96, STA00]. If the participants can keep their private information secretly, it will be computationally difficult for an attacker to derive the key from the public information and thus impersonate one or more participants of the session. Moreover, a service instance that attempts to join a session must be "recommended" by a session participant to the session authority first. If a malicious instance outside of the session tries to communicate with any instance inside the session, it will be detected by the manager instance of that session. This further improves the security level of the system.

3.4 Session Management Protocol

The instance ID Authenticator protocol provides dependable authentication operations for participants of a given session and helps them authenticate each other. It works well in the case of simple business flows. We will now consider the following, more complicated, cases:

- □ A participating service instance of a given session may have local operations, which are not supposed to be known by other participants. For instance, the participant may attempt to initiate a sub-workflow and does not want other participants to know any details of it. The protocol should provide a mechanism for enclosing local details and secrets of session participants.

- □ When an attacker has compromised a session participant, the instance ID Authenticator protocol has difficulty in confining the threat to a small area of the session. The attacker may recommend more malicious service instances into the session.

In order to resolve these problems we introduce a session management protocol which uses the CA action mechanism to manage sessions so as to achieve a better level of attack confinement.

3.4.1 Overview

In the session management protocol, we use nested CA actions to structure a workflow. Nested actions could be pre-defined and invoked dynamically as required. A service instance or role in a CA action is only permitted to communicate with other roles in the same action. What happens in a nested CA action is completely transparent to the enclosing action. This provides a protection mechanism for the roles within the CA action. Any service instance invoked by a role of a CA action must be terminated at the point in time when that CA action ends. In addition, the CA action support mechanism is responsible for monitoring the state of roles. CAA manager must ensure that there are no errors or exceptions unresolved (including the possibility of signalling an exception to the enclosing action) when the CA action ends. If an attacker or a malicious service instance is found in a CA action, the thread and damage must be limited to that CA action. While a transaction model may offer some similar features, we select CA action scheme for the reason that CA action scheme has a powerful ability to handle concurrent exceptions [RAN99, XU99]. In a given session, multiple instances may be running concurrently, and an error may involve more than one instance. In order to resolve this type of errors, the instances involved should take actions collectively. CA actions provide an appropriate mechanism for handling such concurrent exceptions.

3.4.2 Management Operations of CA Actions

We define five basic protocol operations that provide functions for manage a set of nested CA actions.

3.4.2.1 Start-an-Original-CA-Action

A user can send the "**Start-an-Original-CA-Action**" to its session manager to initiate a new session. Since the user has not exchanged the identifiers with the session manager instance, this message is not appended with the authenticator.

```
 <Start-an-Original-CA-Action >
  <SendingID> The ID of the sending instance
  </SendingID>
  ............ some other information (e.g. the details of the original CA
        action)
 </Start-an-Original-CA-Action >
```

The session manager invokes a new manager instance, including an SA instance and a CAA manager instance, and the manager instance sends the following message back:

```
<Start-an-Original-CA-Action-Result>
 <SessionHandle>
  <Manager type="uri" value="URI of Manager"/>
       <ID>
            Session ID
       </ID>
 </SessionHandle>
</Start-an-Original-CA-Action-Result>
```

From now on, the session is under the management of the original CA action.

3.4.2.2 Start-a-Nested-CA-Action

If a service instance tries to invoke a nested CA action, it should send this message to its present CAA manager, which is a part of the session manager instance:

```
 <Start-a-Nested-CA-Action>
  <CAHandle>
         <CA type="uri" value="URI of the CA"/>
         <ID>CAA manager ID</ID>
        </CAHandle>
  <NCAHandle>
         <NCA type="uri" value="URI of the NCA"/>
         <Policy>
    Policy about the nested CA action and other relative information
   about the roles
   </Policy>
  </NCAHandle>
  <SendingID> The ID of the sending instance
  </SendingID>
   ............some other information
 </Start-a-Nested-CA-Action>
```

If CAA manager accepts this application, it invokes a new nested CAA manager instance and sends the nested manager instance's related information back:

```
<Start-a-Nested-CA-Action-Result>
 <accept>true</accept>
 <NCAHandle>
  <NCA type="uri" value="URI of the NCA"/>
  <NCAID>Nested CAA manager instance's ID  </NCAID>
 </NCAHandle>
</Start-a-Nested-CA-Action-Result>
```

3.4.2.3 Inform-Enter-a-Nested-CA-Action

After a new nested CAA manager instance is invoked, the nesting CAA manager instance sends the "**Inform-Enter-a-Nested-CA-Action**" message to all the service instances which are expected to act as roles of that nested CA action and inform them to enter that nested CA action:

```
<Inform-Enter-a-Nested-CA-Action>
 <CAHandle>
        <CA type="uri" value="URI of the CA"/>
        <ID>CAA manager's ID</ID>
       </CAHandle>
 <NCAHandle>
  <NCA type="uri" value="URI of the NCA"/>
  <NCAID>Nested CAA manager instance's ID   </NCAID>
  <Role>.........</Role>
 </NCAHandle>
 <SendingID> The ID of the sending instance
 </SendingID>
 ............some other information
</Inform-Enter-a-Nested-CA-Action>-
```

3.4.2.4 Enter-a-Nested-CA-Action

With the permission of entering a nested action, a service instance may decide not to enter that action immediately. After finishing its present job, the instance sends the "**Enter-a-Nested-CA-Action**" message to the nested CAA manager, informing the manager that this instance is ready to enter the nested CA action and is under the control of the manager from now on:

```
<Enter-a-Nested-CA-Action>
 <NCAHandle>
  <NCA type="uri" value="URI of the NCA"/>
  <NCAID>Nested CAA manager instance's ID
  </NCAID>
  <Role>........</Role>
 </NCAHandle>
 <SendingID> The ID of the sending instance
 </SendingID>
```

```
</Enter-a-Nested-CA-Action>
```

After the nested CAA manager obtains this message and verifies the MAC appended with the message, it sends a reply back to the sending instance:

```
<Enter-a-Nested-CA-Action-Result>
  <accept>true</accept>
  <NCAHandle>
   <NCA type="uri" value="URI of the NCA"/>
   <NCAID> nested CAA manager instance's ID
   </NCAID>
   <Role>……..</Role>
  </NCAHandle>
</Enter-a-Nested-CA-Action-Result>
```

3.4.2.5 CA-Action-End

When the original (outermost) CA action ends, all the session ends. When a CA action finishes, the CAA manager sends the "**CA-Action-End**" message to all the roles of that CA action. This message includes information about the state of the CA action at the point of termination:

```
<CA-Action-End>
  <Result>Success</Result>
  <CAHandle>
        <CA type="uri" value="URI of the CA"/>
        <CAID>ID of CAA manager instance</CAID>
  </CAHandle>
</CA-Action-End>
```

From the "**CA-Action-End**" message, the roles are informed that the CA action has ended successfully. After having received the replies from all the roles, the CAA manager terminates the CA action.

3.4.3 A Scenario

In this section, we use the previous scenario described in Section 3.3.3 again to illustrate how the session management protocol works with the instance ID Authenticator protocol.

As shown in Figure 16, suppose that **UI** first sends a "**Start-an-Original-CA-Action**" message to the session manager in order to start a new session. An instance of session manager **MI** (including a SA instance and an original CAA manager instance) is created and invoked.

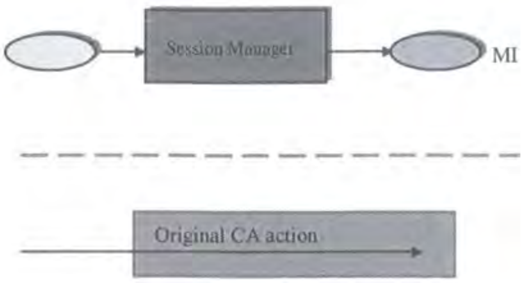


Figure 16 Invoking the Original CA action

UI then invokes the service instance **NI** and **NI** invokes the service instance **NI2**. Once these service instances have been accepted as participants of the session, the CAA manager instance will regard them as instances invoked in the Original CA action (see Figure 17).

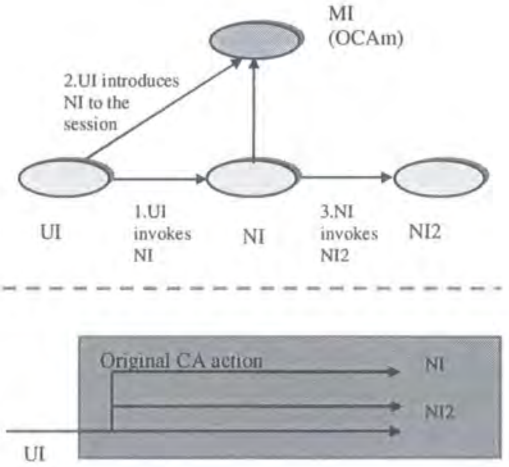


Figure 17 Invoking new service instances within the Original CA action

In order to enclose the operations between **NI2** and **UI** into a nested CA action, **NI2** needs to send a "**Start-a-Nested-CA-Action**" message to **MI** and applies for the creation of a new nested CA action. After **MI** invokes a new nested session manager instance (CI(C, g^{C})), it sends an "**Inform-Enter-a-Nested-CA-Action**" message to **UI** and **NI2**, permitting them to enter that nested CA action. When **UI** and **NI2** are ready, they send an "**Enter-a-Nested-CA-Action**" message to the nested CAA manager instance separately. **UI** also has to use $K_{UI,CI} = g^{XC}$ to authenticate itself to the nested CAA manager and then enter the nested action. Similarly, **NI2** uses g^{NC} to identify itself. After **UI** and **NI2** have entered the nested CA action, they are all under the control of the nested CAA manager (see Figure 18).

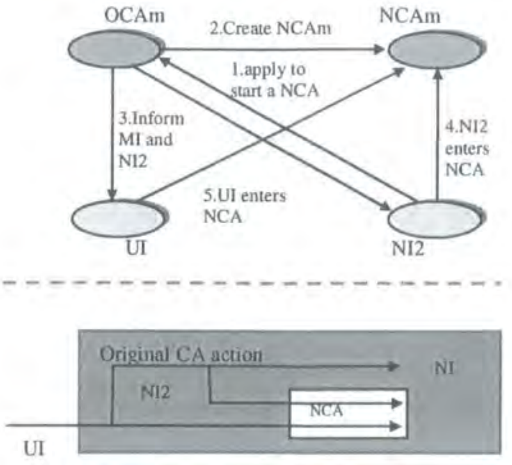


Figure 18 Invoking a nested CA action

3.5 Summary

In this chapter we have presented the design of our multi-party authentication protocols for Web Services and used the XML notation to specify the corresponding operations. Our protocol is designed for complex Web services applications running over the Internet. Since Web services may belong to different organizations and under the control of different security systems, our protocol is intentionally designed to be independent of any particular security systems. We have used the Diffie-Hellman scheme to exchange the authentication information and CA actions to structure nested business flow. Compared to Hada and Maruyama's scheme [HAD02], our protocol is improved in authenticating service instances for complex sessions and defending against a variety of attacks. The following chapter introduces the model analysis of our system and the results obtained from the experiment.

Chapter 4 System Evaluation and Formal Analysis

We have developed an experiment to evaluate the performance of our protocols. We also design an analytic model to further support the conclusions of the experiment, as the experiment can only partly simulate real life environments.

4.1 Description of the Experiment

4.1.1 Introduction of Programming Language and Tools

In the experiment, we employ Java to implement the experiment system. Java is an object-oriented language developed by Sun Microsystems to provide a programming language that can be used on a variety of platforms. Because of its platform-independence, Java has been widely used in e-business application development and has been accepted by the industry. Furthermore, it provides a very good exception mechanism.

In addition, we make use of GLUE toolkits to generate experimental Web services, which communicate based on the SOAP protocol. GLUE is a complete Web service platform provided by The Mind Electric. The platform itself provides extensive support for SOAP, WSDL, and UDDI. It is easy to use and fast at creating and deploying applications with Web services.

4.1.2 Structure of the Experiment

Our session multi-party authentication system is designed for large scale, with a large mumber of users, and should work in a distributed environment. Normally, operations in a distributed system may be executed simultaneously and the performance of the system is sometimes difficult to evaluate. Therefore, in the experiment, we try to simulate the worst case, where all the operations of the system are executed consequentially. Thus, we implement the experiment in the following way (see Figure 19).

A Web service is developed as the session authenticator and three Web services (**Web service 1, Web service 2, and Web service 3** in Figure 19) are developed to spawn the Web service instances that act as session partners in the experiment. **Web services 1, 2** and **3** invoke each other in the sequence shown in Figure 19 repeatedly until they have spawned a particular amount of services instances and introduced them into the session. The amount of the session partners within a particular session is managed by the first Web service instance generated by **Web service 1**. The function of this service instance is to contact the session manager to initiate a session as well as end it at the proper time. In this service instance, a pre-set counter is used to specify the amount of session partners.

For an example, in order to establish a session with six partners and record the time consumed to accomplish this session, the counter should be set as two and then **Web service 1** generates a new instance to start a new session. When the service instance spawned by **Web service 3** in the first turn contacts **Web service 1**, **Web service 1** will check the counter in the first generated instance. If the counter is bigger than one, **Web service 1** will then reduce the counter by one and contact **Web service 2** to start another turn. Else, **Web service 1** will inform the session authority to end the session and calculate the time consumption of this session.

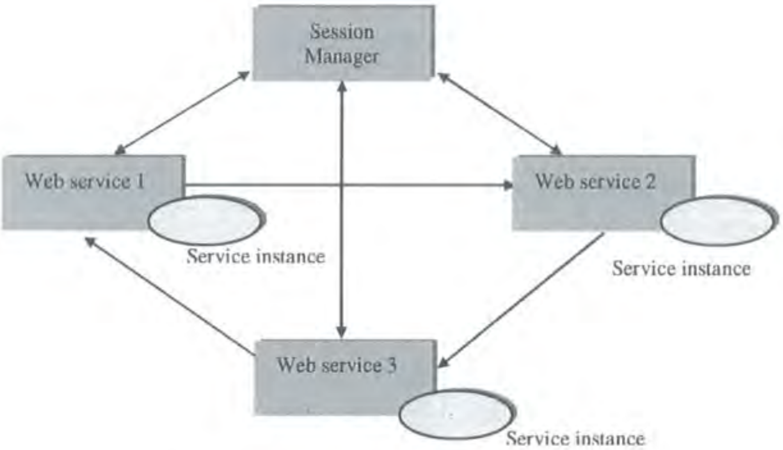


Figure 19 Structure of the experiment

After the instances invoked by these three Web services have finished their jobs, they will be disposed of to release the resources. Therefore, in one session there are only two or three session partners occupying the memory simultaneously even when this system is simulating a session with large number of partners. This design avoids the case that the system consumes so many resources that the precision of the experiment result is affected.

4.2 Experimental Evaluation

Since all the operations in this experiment should be executed sequentially, it would be reasonable to execute the experiment on a computer with one CPU. However, in order to make the experiment more convincing, we have done this experiment both on a single computer and in a LAN environment. The results getting from the two cases have the same tendencies.

4.2.1 Standard Deviation of Experiment Results

To our certain knowledge, there are several factors which may influence the performance of the experiment:

- Java shields the garbage collection from developers. The Java virtual machine (JVM) can free the resources no longer needed by the program automatically. This mechanism benefits the software development. However, the unpredictable overhead of the garbage collection mechanism also affects the performance of the Java program and makes it relatively less stable.

- The operation system used in the experiment is Windows 2000. And the API `System.currentTimeMillis()` is used to get the system time at the begin and the end of the experiment. The subtraction of the two times is regarded as the time consumption of the experiment. However, the operation of the experiment process may be interrupted by the operation system backend management mechanism unpredictably. Therefore, the result from the experiment may be longer than the time that the experiment has really consumed.

- The private key of each Web service instance in the experiment is randomly selected. Therefore, the time consumption of generating public keys and secret keys based on these random numbers varies in different runs of the experiment, although in theory the average result should approach to a limit when large numbers of the private keys are used.

It is therefore necessary to investigate how much variation these factors will cause. In order to evaluate the impact of these factors, we repeated the data collection process of each different operation fifteen times (see Figures 20, 21 and 22).

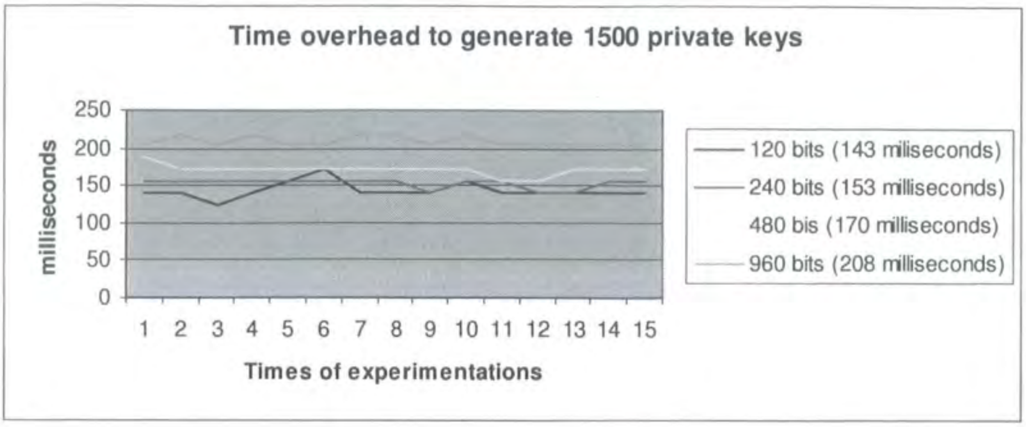


Figure 20 Time consumed to generate private keys

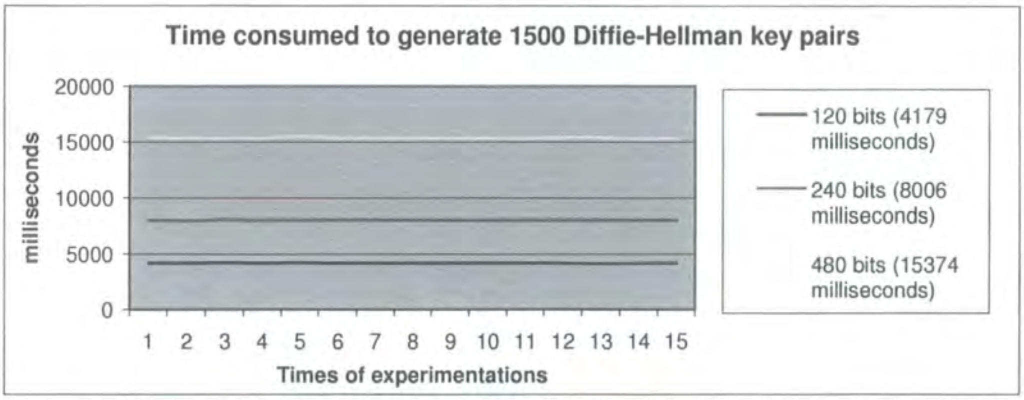


Figure 21 Time consumed to generate key pairs

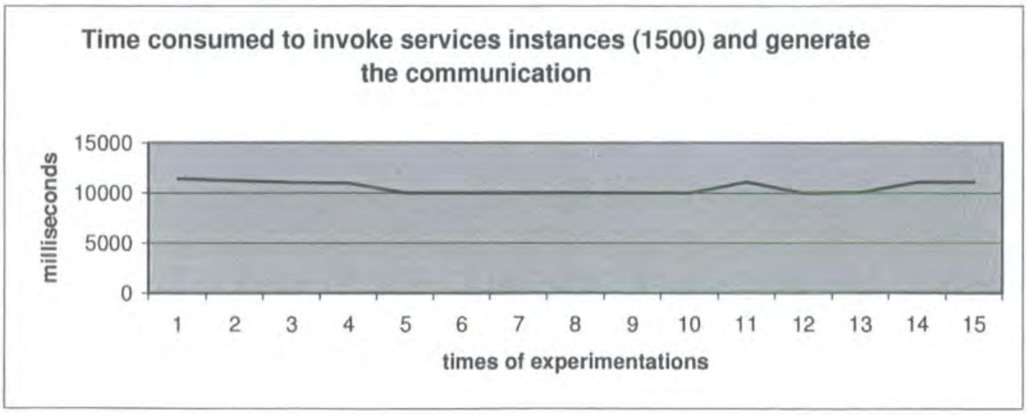


Figure 22 Time consumed to invoke services instances (1500) and generate the communication

As what Figures 20, 21 and 22 illustrate, the standard deviation of the results from different runs of the experimentation is relatively small, and so the influence of the factors described above is generally negligible.

4.2.2 System Scalability

In the experiment shown in Figure 23, different sessions with the number of partners ranging from 300 to 1500 were generated to evaluate the scalability of the system. The curves from the experiment indicate that the time consumed in accepting new service instances into a session is proportional to the number of session partners, which is an acceptable result.

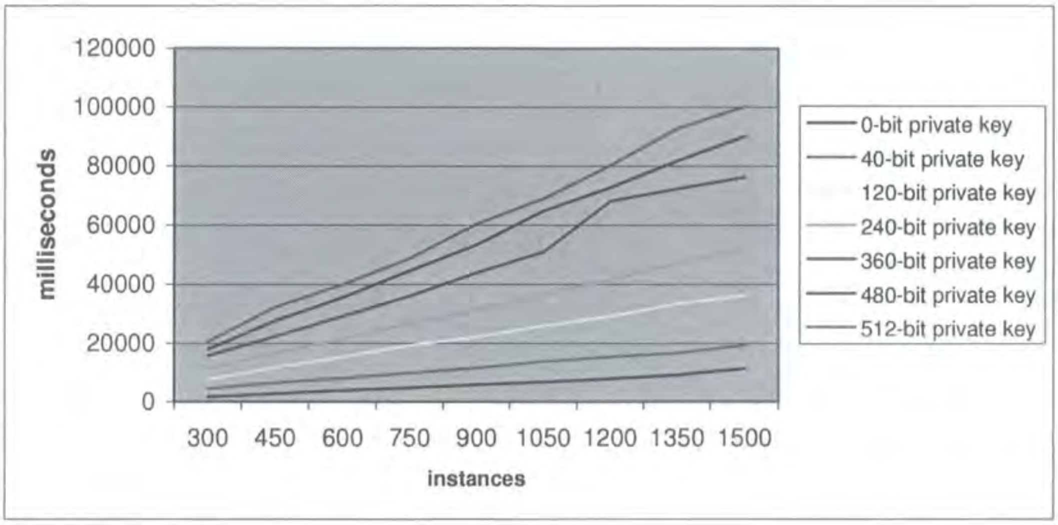


Figure 23 Scalability of the session authentication system

4.2.3 Concurrency

In order to evaluate the actual performance of the session authentication system when concurrency is allowed, two experiments have been conducted. In the first experiment (see Figure 24), we established seven kinds of sessions whose number of partners varied from three to ten and collected the data of the performance load of a maximum of ten simultaneous sessions. It is noted that the curves of the sessions' time consumptions are approximately linear.

However, it may be possible that these curves in Figure 24 do not reflect the real feature of the system concurrent performance because the number of simultaneous sessions is not large enough.

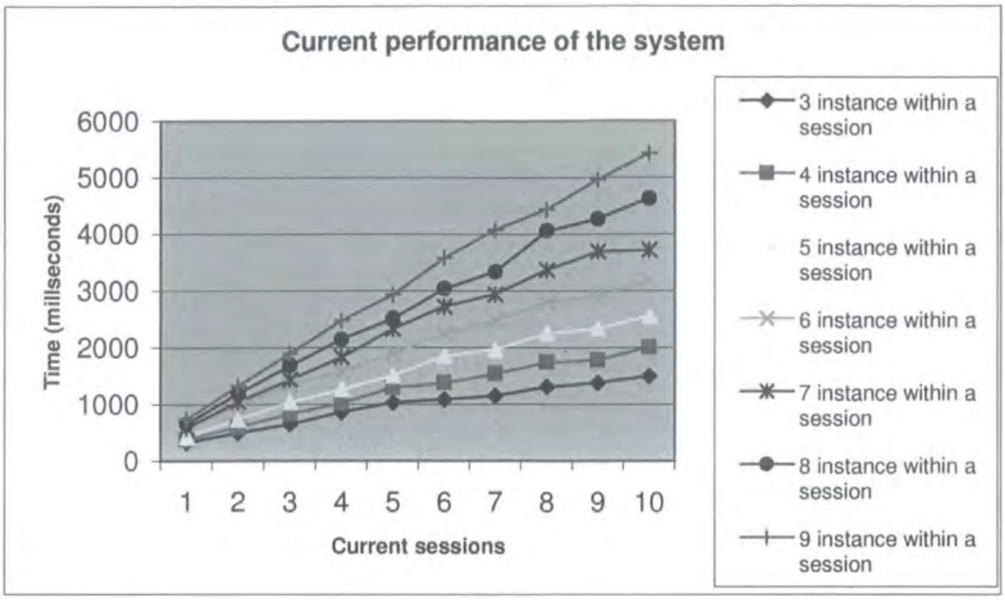


Figure 24 Current performance of the system (experiment 1)

So, more sessions are executed concurrently in the second experiment. It is obvious that the increasing speed of the time consumed for achieving a session reduces gradually as more sessions are involved into the experiment (see Figure 25).

However when more than 50 sessions are executed concurrently, some messages transported between service instances are denied due to the limits on the amount of simultaneous messages that a Web server can handle. The system will become unstable in that condition. Thus, the curve in Figure 25 is only reliable up to fifty concurrent sessions. However, from that part of the curve, we can obtain the same result that we mentioned above.

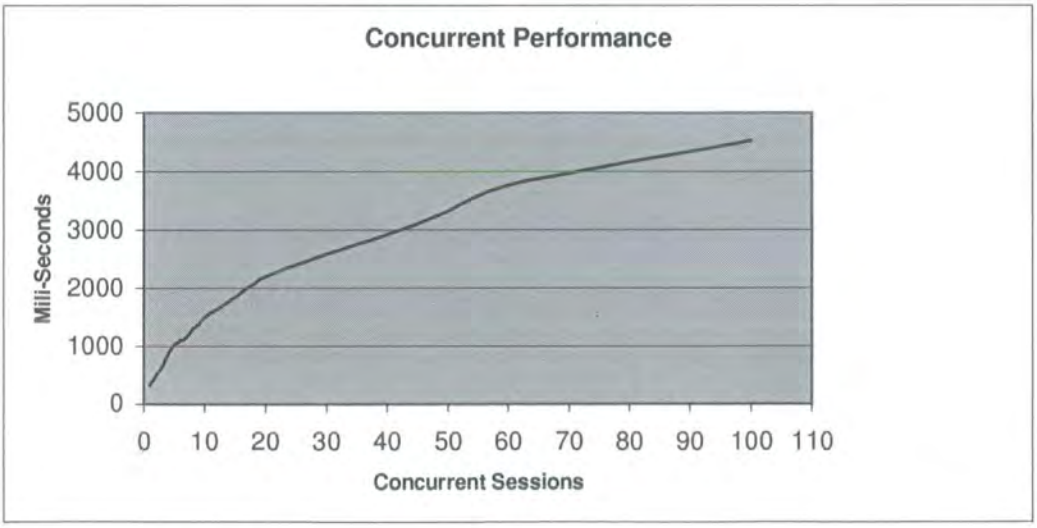


Figure 25 Current performance of the system (experiment 2)

4.2.4 Calculation Speed of Web Services

Normally, when we execute a Java program on the same computer more than once, the performance of different experimentations is similar. However, the performance of Web services seems to be different. Figure 26 compares the different performance of a Web service and a normal Java program to generate 1500 120-bit private keys on the same computer.

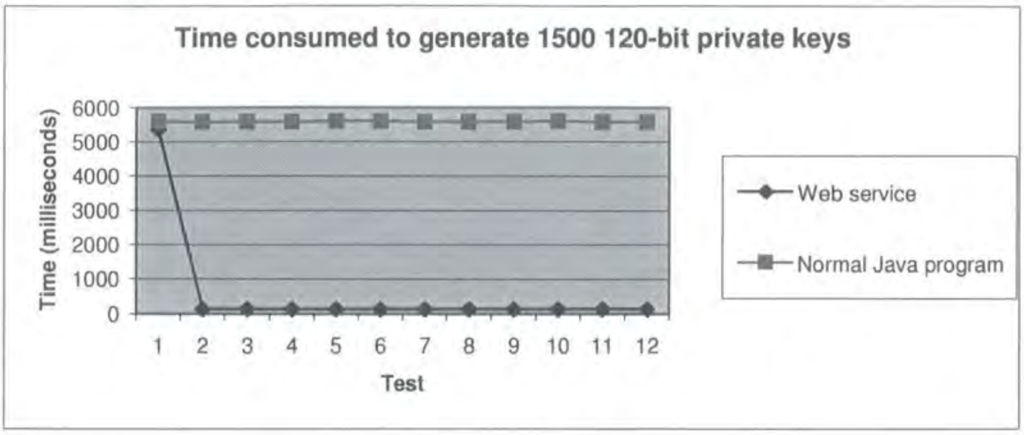


Figure 26 Performance comparison of a Web service and normal a Java program

The time consumption of the normal Java program is similar each time while that of the Web service becomes much faster after the first time. From the second time onwards, the Web service's performance becomes stable. Thus in the experiment, we always collect the data after the Web service's performance stabilizes and use these data to evaluate the Web service's time consumption. We explain this phenomenon as follows. When the user invokes a Web service for the first time, the object of the service will be generated and kept in memory. If the user contacts it later, the Web service will invoke the object from the memory directly rather than instantiate it again.

4.3 Model Analysis of the Session Authentication System

Since the experiments can only partially simulate the Web service sessions with a limited number of session partners, we also create an analytical model to further analyse these experimental results.

4.3.1 Notation

We firstly introduce the following notations to facilitate subsequent discussions.

P_I Service instance which is the *i*-th instance introduced into the session

$T_{inst, I}$ The time consumed of spawning a new service instance P_I

$T_{key\text{-}pair(j, i)}$ The time consumed to generate the *j*-th private key and public key in the process of introducing P_I into the session

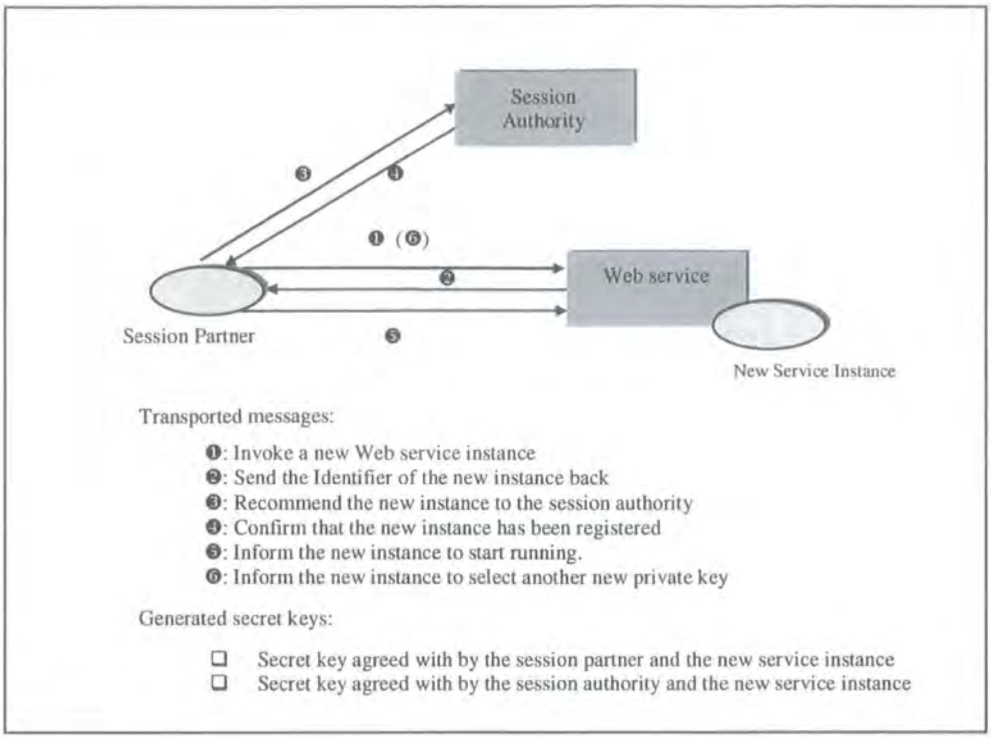
$T_{sec(j, i)}$ The time consumed to generate the *j*-th secret key in the process of introducing P_I into the session

$T_{m(j, i)}$ The time consumed to generate and transport the *j*-th message between two Web service instances in the process of introducing P_I into the session

4.3.2 Time Consumption of Introducing a New Session Partner to the Session Authority

In our model, a user instance first contacts a session authority to initiate a session and the session authority then assigns a SA instance to manage this session. The user is regarded as the first session partner of the session. From then on, a new service instance must be recommended to the session authority before it is accepted as a session partner. Normally, this task is performed by the session partner, which invoked this new service instance (see the Session partner in Figure 27).

As with the presentation of Figure 27, when a partner of a session attempts to recommend a newly generated Web service instance to the session authority of the session, at least five messages should be sent over the network. Firstly, the session partner sends a request to access a Web service (see Step 1). Besides the request, other useful information such as the identity of the session partner is also sent. After the Web service receives the message, it spawns a new service instance to manage the request. The new service instance selects its private key randomly from the key space, calculates its public key and sends the public key as its identifier back to the session partner (see Step 2). After receiving the reply from the new instance, the session partner forwards the new instance's identifier to the session authority instance, which holds all the information of the session partners within that session. If the session authority realizes that the identifier of the new instance has already been adopted by another session partner, the session authority will inform the recommender that the register has been refused. In this case, the recommender should inform the new service instance to select a new private key (see Step 6). Steps 2, 3 and 4 are repeated until the session partner gets confirmation from the session authority that the new instance has been accepted. After receiving the confirmation, the session partner sends a message (message 5) to inform the new service instance to start working.



Transported messages:

❶: Invoke a new Web service instance
❷: Send the Identifier of the new instance back
❸: Recommend the new instance to the session authority
❹: Confirm that the new instance has been registered
❺: Inform the new instance to start running.
❻: Inform the new instance to select another new private key

Generated secret keys:

□ Secret key agreed with by the session partner and the new service instance
□ Secret key agreed with by the session authority and the new service instance

Figure 27 The process of registering a new instance to the session authority

Let us now consider a session partner that attempts to introduce a newly spawned Web service instance P_I to the session authority, and this process achieves after x_i identity selections. The total time consumption of this process $T_{total, I}$ is as follows:

$$T_{total, I} \leq \sum_{j=1}^{4x_i+1} (T_{m(j,i)}) + \sum_{j=1}^{x_i} (T_{key\text{-}pair(j,i)}) + \sum_{j=1}^{2(x_i+1)} (T_{sec(j,i)}) + T_{inst, I} \qquad (1)$$

with equality when all the operations of the system are processed sequentially.

Therefore, in one session where n service instances are accepted into the session, the total time consumption of this process $T_{n\text{-total}}$ could be expressed in the following way:

$$T_{n\text{-total}} = \sum_{i=1}^{n} T_{total, I} \qquad (2)$$

$$\leq \sum_{i=1}^{n} \Big(\sum_{j=1}^{4x_i+1} (T_{m(j,i)}) + \sum_{j=1}^{x_j} (T_{key\text{-}pair(j,i)}) + \sum_{j=1}^{2(x_j+1)} (T_{sec(j,i)}) + T_{inst, I} \Big)$$

$$= [4(x_1 + x_2 + .. + x_n) + n](\overline{T}_{message}) + (x_1 + x_2 + .. + x_n)(\overline{T}_{key\text{-}pair}) +$$

$$2[(x_1 + x_2 + .. + x_n) + n](\overline{T}_{secret}) + n\overline{T}_{instant}$$

where $\bar{T}_{message} = \dfrac{\sum_{i=1}^{n}(\sum_{j=1}^{4x_i+1}(T_{m(j,i)}))}{\sum_{i=1}^{n}(4x_i+1)}$, $\bar{T}_{key\text{-}pair} = \dfrac{\sum_{i=1}^{n}(\sum_{j=1}^{x_i}(T_{key\text{-}pair(j,i)}))}{\sum_{i=1}^{n}x_i}$, $\bar{T}_{secret} = \dfrac{\sum_{i=1}^{n}(\sum_{j=1}^{2x_i+1}(T_{sec(j,i)}))}{\sum_{i=1}^{n}(2x_i+1)}$ and $\bar{T}_{instant} = \dfrac{\sum_{i=1}^{n}(T_{inst,I})}{n}$.

Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of random variable, the probability, **P**(n, k), that there is at least one duplicate is [STA98]:

$$\mathbf{P}(n,\ k) = 1 - \frac{n!}{(n-k)!n^k}$$

If the Key space of the security system is large enough, the probability that two partners within the same session will select the same private key is extremely small. Thus, for every i in $\{1,2,\ldots,n\}$, $x_i \approx 1$. Thus the value of ($x_1 + x_2 + .. + x_n$) in equation (2) is approximately n. Therefore, when all the operations execute sequentially:

$$\begin{aligned} T_{n\text{-}total} &\approx 5n(\bar{T}_{message}) + n(\bar{T}_{key\text{-}pair}) + 4n(\bar{T}_{secret}) + n\bar{T}_{instance} \qquad (3) \\ &= n[5(\bar{T}_{message}) + \bar{T}_{key\text{-}pair} + 4\bar{T}_{secret} + \bar{T}_{instance}] \end{aligned}$$

Equation (3) implies that, in order to accept n service instances into a session, there will be about $5n$ messages transported, n key pairs, $4n$ secret keys, and n new service instances generated. That is, in the worst case the time consumption of introducing service instances into one session increases linearly with the amount of the session partners.

4.3.3 Time Consumption of Initiating the Communication between Session Partners

We also generated an analytic model of our security system for generating the communication between session partners.

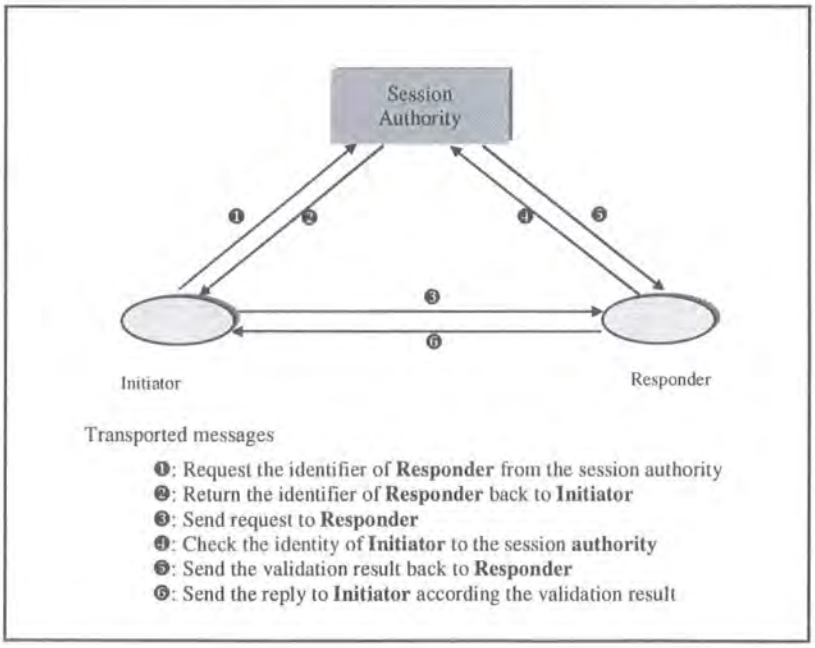
Figure 28 illustrates the conversation between two session partners, **Initiator** and **Responder**, which tried to initiate a communication between each other. Since these two session partners never communicated before, each of them has to ask the session authority instance to examine the other's identity. Generally, there are six messages that have to be sent through the network. **Initiator** will agree on a secret key with **Responder**.

Let $T_{gc(i,j)}$ be the time consumption of the process for two partners, P_I and P_J, to agree on the secret key. And

$$T_{gc(i,j)} \le \sum_{u=1}^{2} T_{secret(i,j),u} + \sum_{u=1}^{6} T_{message(i,j),u} \qquad (4)$$

In the above inequality we refer to the time consumption of generating the u-th secret key in this process as $T_{secret(i,j),u}$ and the time consumption of generating and transporting the u-th message in this process as $T_{message(i,j),u}$. Equality is achieved when all the operations in the process are executed sequentially.

From inequality (4), we can say that the resources consumed to authenticate a session partner do not vary as the change of the membership of the session or the relationship among the session partners. If **Initiator** and **Responder** have contacted before, then there is no additional operation needed. They should have generated the secret key and can use it directly to prove their identities. Of course, the session partners can contact the session authority instance to verify each other again for security purposes.



Transported messages

❶: Request the identifier of **Responder** from the session authority
❷: Return the identifier of **Responder** back to **Initiator**
❸: Send request to **Responder**
❹: Check the identity of **Initiator** to the session **authority**
❺: Send the validation result back to **Responder**
❻: Send the reply to **Initiator** according the validation result

Figure 28 The process of contacting other session partners

Assuming there is a session, in which there are n partners never communicate with each other before. The time it takes every partner to form original connections with all the other session partners T_{gac} should be:

$$T_{gac} = \sum_{j=2}^{n} \sum_{i=1}^{j-1} (T_{gc(i,\ j)}) \tag{5}$$

$$\leq \sum_{j=2}^{n} \sum_{i=1}^{j-1} \left(\sum_{u=1}^{2} T_{secret(i,\ j),u} + \sum_{u=1}^{6} T_{message(i,\ j),u} \right)$$

$$= \frac{n(n-1)}{2} (2\overline{T}_{secret} + 6\overline{T}_{message})$$

Where $\overline{T}_{secret} = \dfrac{\sum_{j=3}^{n} \sum_{i=1}^{j-2} (\sum_{u=1}^{2} T_{secret(i,\ j),u})}{\sum_{j=3}^{n} \sum_{i=1}^{j-2} 2}$ and $\overline{T}_{secret} = \dfrac{\sum_{j=3}^{n} \sum_{i=1}^{j-2} (\sum_{u=1}^{6} T_{message(i,\ j),u})}{\sum_{j=3}^{n} \sum_{i=1}^{j-2} 6}$.

The time complexity of above inequality equation is $O[n^2]$.

In the above discussion, we have described the process of our security system for two session partners initiating communication. In Section 4.4, we will compare this model with the model of another decentralised solution.

4.3.4 Comparison of the Results of the System and the Model

In the experiment, the transporting channel is stable. Therefore, the time consumption of transporting a message between two instances and spawning a service instance is stable. Every session partner spawned by the same service in the experiment has the same calculating ability since all these instances are executed on the same computer.

In addition, it is impossible to anticipate the time consumption of generating a public key or a secret key precisely in the experiment. However, we can predicate that:

As we described in Chapter 3, in the Diffie-Hellman algorithm, time consumption of generating a public key or a secret key is determined by the length of the private key. And the private key used in this system is a random number picked from the set $\{0, 1, \ldots, 2^n - 1\}$ where n is selected by the system according to the requirement of the security strength. When the system employs Diffie-Hellman to agree with the secret keys, n is normally twice as the length of the session keys that are derived from the security keys.

If i integers are selected randomly from the set $\{0,1,\ldots,\ 2^n-1\ \}$, the average length of these random numbers is a limit when i is large enough.

Therefore, when the number of session partners is large, as it is in our experiment, the value of the arguments in the right of the equation (3) should not have a large variation in different experimentations. Hence, we can record the time consumption of different operations such as transporting messages, generating private keys, public keys, and spawning service instances separately. Then, we can add them together to evaluate the performance of the system.

In Figure 29, we include the time consumptions of achieving various sessions. These sessions consist of different numbers of session partners. The lengths of the private keys employed within the sessions are different from one column to another. Specially, the column "0 bit" in Figure 29 indicates the session that does not do the operations of generating the key pairs and validating secret keys, that is, it only records the time consumed to generate the service instance and the time consumed to generate and transport the messages.

Length of the private keys Number of the Instances	0 bit	40 bits	120 bits	240 bits	360 bits	480 bits	512 bits
300	1906	4768	7775	11516	15969	18016	20703
450	2938	6562	11812	16688	22422	27640	32297
600	3921	8079	15532	21579	29266	35609	39985
750	4875	10042	19435	26891	36047	48609	48765
900	5937	11750	22328	31532	43950	53453	60438
1050	6953	13956	26016	36813	51078	67844	69047
1200	8005	15344	29235	41453	68040	72812	80141
1350	9328	16704	33500	47016	72219	81703	92781
1500	11422	19375	36187	51812	76312	90172	100203

Figure 29 Private key's length and the time consumption (millisecond) of the experiment system

In addition, from the observation of the Diffie-Hellman protocol we can tell that the average time consumed to generate a public key is computationally close to a secret key. Let $\overline{T}_{pri}$ be the average time consumed in generating a private key and $\overline{T}_{pub}$ be that of a public key, equation (3) becomes

$$\begin{aligned} T_{n\text{-}total} &\approx n[5(\overline{T}_{message}) + \overline{T}_{key\text{-}pair} + 4\overline{T}_{secret} + \overline{T}_{instance}] \qquad (6) \\ &\approx n[5(\overline{T}_{message}) + \overline{T}_{pri} + 5\overline{T}_{pub} + \overline{T}_{instance}] \end{aligned}$$

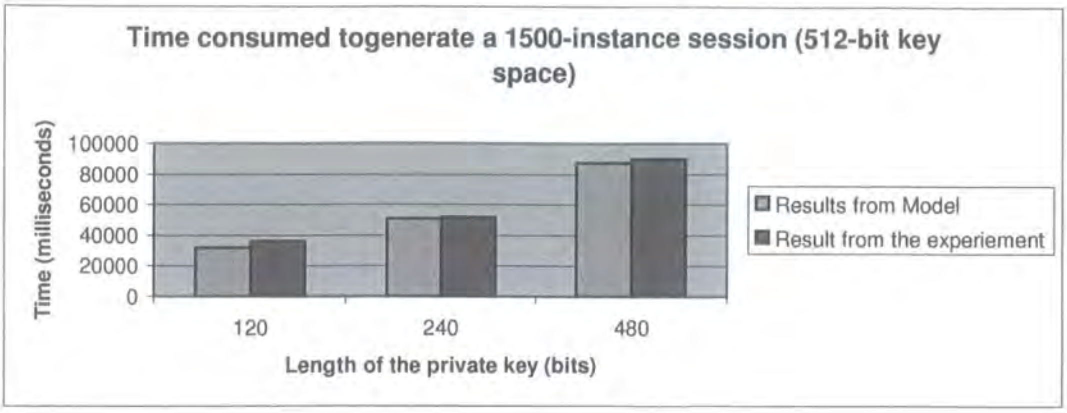


Figure 30 Comparison between analytical and experimental results

Based on the data collected from experiments (as what shown in Figures 20, 21 and 22), we use equation (6) to generate analytical results. Figure 30 shows that the analytical sum of time spent by different parts illustrated in Figures 20, 21 and 22 matches the experimental results presented in Figure 29.

It is notable that the experimental results are always a little larger than the analytical results derived from equation (6). This phenomenon may be explained as follows:

The data shown in Figures 20 and 21, used for the analytical model, is based on an implementation in which the keys are generated by a single Web service instance, one at a time. However, the experimental results, illustrated by Figure 29, are based on the fact that the keys generated by multiple service instances in total. The operations in the experimental system are therefore more possibly disturbed by the OS background management mechanism. Consequently, time consumption of the experimental system should be longer than that of the analytical model, especially when the system execution time is long.

4.3.5 Proportion of Different Parts of Time Consumption

Figure 31 shows the proportion of time consumed by different operations within the security system. The time consumption of generating private keys is less than 1%, negligible compared with other operations. The time consumed in generating public keys and secret keys takes the most part of the system time consumption. However, we anticipate that the proportion of time consumed by transporting messages will increase when the system executes in a large-scale distributed system. In extreme conditions, the time consumption of transporting messages will determine the performance of the system.

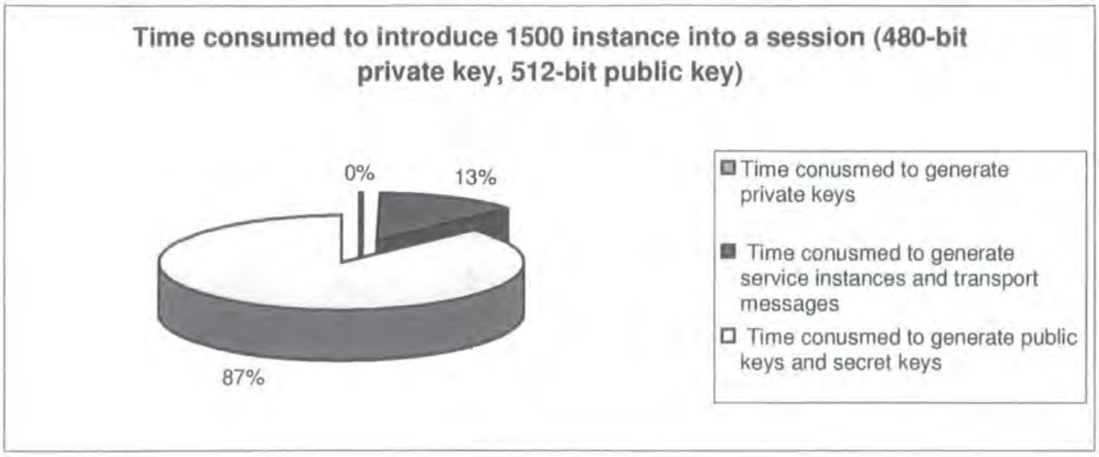


Figure 31 Time consumption of different operations in the system

4.4 Decentralised Solution for Session Authentication Protocol

Before choosing the solution described in Chapter 3 as our solution of the session authentication protocol, we had considered another decentralised mechanism, which does not rely on any third party to manage the identities of the session partners (see Figure 32).

4.4.1 Description of the Decentralised Solution

Like the centralised solution we described previously, this decentralised solution leverages the Diffie-Hellman algorithm to distribute secret keys among session partners. Session partners will use these secret key to authenticate each other in the conversations. In the decentralised solution, in order to invoke a new service instance, there are three messages necessary to be sent. Figure 32 illustrates a scenario of this process.

Suppose a session partner **Instant0** is a service instance involved within a session, and it attempts to assess a Web service, **Web service 1**. Firstly, **Instant0** selects a pair of keys for itself and then sends its request to **Web service 1** (see Step 1 in Figure 32). Apart from the request, **Instance0** also sends its public key as its identifier to **Web service 1**. After receiving the message from **Instance0**, **Web service 1** invokes a new instance, **Instance1** to manage this request. **Instance1** selects a private key from the key space and sends the associated public key as its identifier back to Session Partner (see Step 2 in Figure 32). After Steps 1 and 2, **Instance0** and **Instance1** have achieved the public key exchange so that they can agree on a secret key using the Diffie-Hellman algorithm. In the third step of Figure 32, **Instant0** informs **Instance1** to start working after receiving the reply from **Instance1**. Same as our multi-party authentication system discussed in Chapter 3, the messages in Steps 2 and 3 are attached with MAC information calculated with the secret key that is shared by the **Instance0** and **Instance1** so that these two service instances can verify the originators of the messages. After these three Steps conversation, each of them can make sure that the other instance has agreed on the secret key. **Instance1** can now be regarded as a session partner.

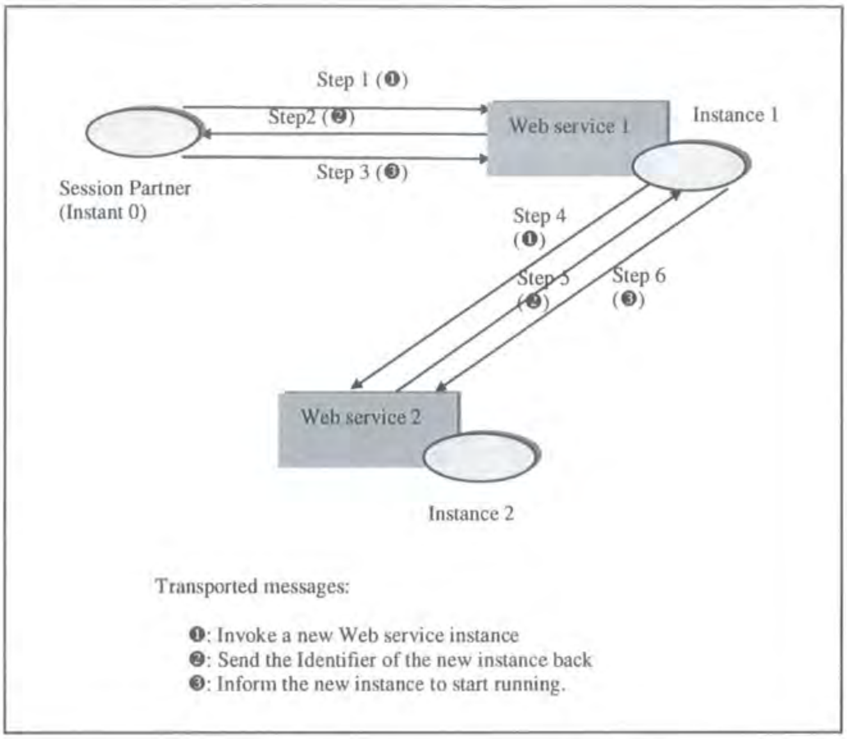
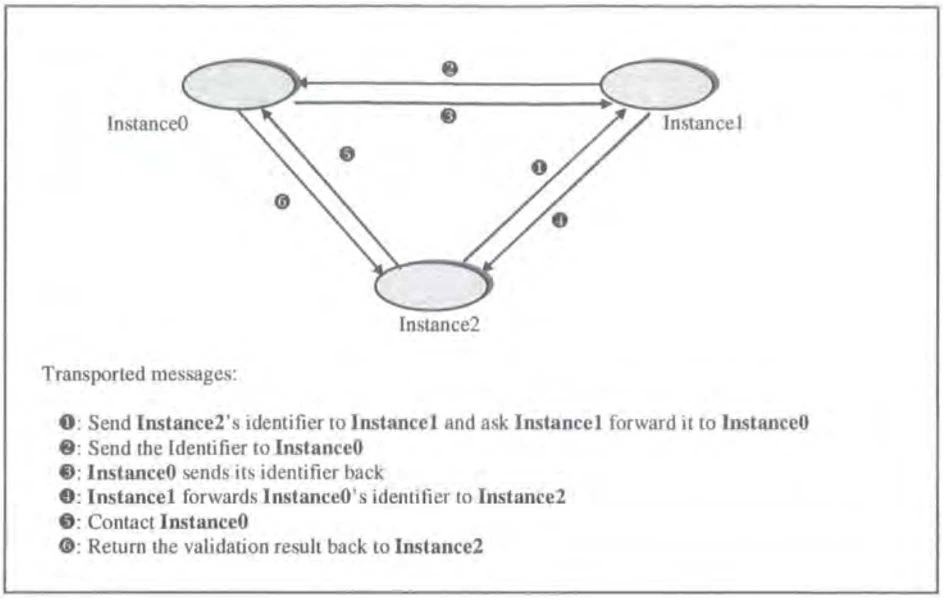


Figure 32 The process of invoking new session partner in the decentralised solution

Assume **Instance1** tries to access another Web service, **Web service 2**. Following the same way, **Instance1** contacts **Web service 2** and generates the communication with **Instance2** which is invoked by **Web service 2** to manage the request of **Instance1** (Steps 4, 5 and 6 in Figure 32).

After **Instance2** has finished its job, it is required to send the result back to **Instance0** directly (see Figure 33). Since at this stage **Instance0** and **Instance2** have both generated trust relationships with **Instance1**, **Instance2** can send its identifier to **Instance0** through **Instant1** (Steps 1 and 2 in Figure 33), and **Instance0** can send its identifier back to **Instance2** through **Instance1** as well (Steps 3 and 4).



Transported messages:

❶: Send **Instance2**'s identifier to **Instance1** and ask **Instance1** forward it to **Instance0**
❷: Send the Identifier to **Instance0**
❸: **Instance0** sends its identifier back
❹: **Instance1** forwards **Instance0**'s identifier to **Instance2**
❺: Contact **Instance0**
❻: Return the validation result back to **Instance2**

Figure 33 The process of initiating communication with other session partners in the decentralised solution

After Steps 3 and 4, **Instance2** agrees on a secret key with **Instance0** using the Diffie-Hellman algorithm. Thus, **Instance2** can contact **Instance0** directly and use the newly generated secret key to prove its identity.

4.4.2 Model Analysis of the Decentralised Solution

Let us now consider partner within a session that invokes a service instance P_I and accept it to the session. A triple-message conversation is needed to invoke and generate the trust relationship with P_I over the network. Moreover, P_I should select a private key and the relevant public key for itself. We refer to the time consumption of this process as $T_{key\text{-}pair,\,I}$. In addition, the two parties of the conversation should generate the secret key respectively. So, the time consumption of accepting P_I into the session $T_{total,\,I}$ is:

$$T_{total,\,I} = \sum_{j=1}^{3} T_{m(j,\,i)} + T_{key\text{-}pair,\,I} + \sum_{j=1}^{2} T_{sec(j,\,i)} + T_{inst,\,I}\,,$$

In the discussion, we neglect the condition that two parties select the same private key accidentally, since the possibility is extremely small. Therefore, when accepting new session partners, the time consumption of this decentralised solution is a little less than that of the centralised solution discussed in Chapter 3.

It is a little complex to assess the time consumption of initiating communication between two session partners that have never contact before. Since two session partners that never communicate before may have to exchange their identifiers via other session partners, this time consumption depends on how many session partners are involved within the process of exchanging the identifiers. Figure 34 details a structure of a session, within which a session partner **Partner1** invokes all the other session partners (**Partner2** to **Partnern).** Thus, **Partner1** has the trust from all other session partners. Assume two session partners P_I and P_J, where $I, J \in \{2,3,..,n\}$, attempt to generate a secret key and start communication. So, in the worst case, the time consumption $T_{gc(i,j)}$ should be:

$$T_{gc(i,j)} \leq \sum_{u=1}^{2} T_{secret(i,j),u} + \sum_{u=1}^{6} T_{message(i,j),u}$$

The time it takes for every partner in this session to form original connections with all the other session partners T_{gac} should be:

$$T_{gac} = \sum_{j=3}^{n} \sum_{i=2}^{j-1} \left(T_{gc(i,j)}\right)$$

$$\leq \sum_{j=3}^{n} \sum_{i=2}^{j-1} \left(\sum_{u=1}^{2} T_{secret(i,j),u} + \sum_{u=1}^{6} T_{message(i,j),u}\right)$$

$$= \frac{(n-1)(n-2)}{2}\left(2\overline{T}_{secret} + 6\overline{T}_{message}\right)$$

Where $\overline{T}_{secret} = \dfrac{\sum_{j=3}^{n} \sum_{i=2}^{j-1} \left(\sum_{u=1}^{2} T_{secret(i,j),u}\right)}{\sum_{j=3}^{n} \sum_{i=2}^{j-1} 2}$ and $\overline{T}_{secret} = \dfrac{\sum_{j=3}^{n} \sum_{i=2}^{j-1} \left(\sum_{u=1}^{6} T_{message(i,j),u}\right)}{\sum_{j=3}^{n} \sum_{i=2}^{j-1} 6}$.

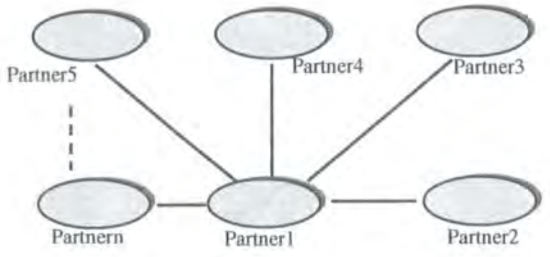


Figure 34 Session structure (1)

It is notable that the order of growth of above equality is $O[n^2]$, the same as that of the inequality (5) in Section 4.3.

Consider a session, which is presented within Figure 35. In this session, each P_I is invoked by $P_{I\text{-}1}$, where $i \in \{2,3,..,n\}$. So to initiate a communication for two strange session partners P_I and P_J, where $1 \le i \le j \le n$, the time consumption $T_{gc(i,j)}$ should be:

$$T_{gc(i,j)} \le \sum_{u=1}^{2} T_{secret(i,j),u} + \sum_{u=1}^{2(j-i)+2} T_{message(i,j),u}$$

For every partner in the session to generate trust relationship with each other, the time consumption T_{gac} should be:

$$T_{gac} = \sum_{j=3}^{n}\sum_{i=1}^{j-2} T_{gc(i,j)}$$

$$\le \sum_{j=3}^{n}\sum_{i=1}^{j-2}(\sum_{u=1}^{2} T_{secret(i,j),u} + \sum_{u=1}^{2(j-i)+2} T_{message(i,j),u})$$

From the above equality, we can see that in the worst case the order of growth of initiating the communication among session partners is $O[n^3]$, which is higher than that of the centralised solution ($O[n^2]$).

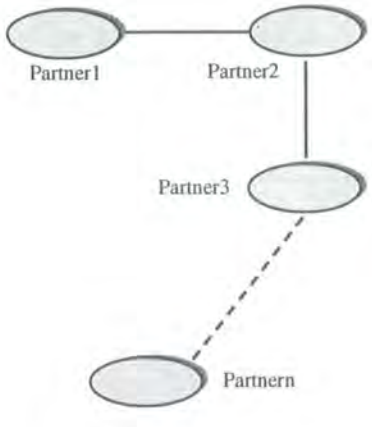


Figure 35 Session structure (2)

4.4.3 Comparison between the Centralised Solution and the Decentralised Solution

Both the centralised solution and the decentralised solution have their own advantages. The decentralised solution does not rely on any third party to manage the key distribution. Therefore, the decentralised solution avoids the inherent drawbacks of protocols that rely on a third party. This is a great advantage. Nevertheless, it also has some inevitable drawbacks. Here, we list some of them as follows:

- ❑ In the decentralised solution, session partners lack effective ways to get the real-time information of the session (e.g., the change in the number of session partners, the information of newly accepted session partners).
- ❑ In the decentralised solution, it is difficult to examine whether the identity of a session partner is unique in the session.
- ❑ In the decentralised solution, the session partners also have the responsibility to help other related session partners to initiate communication. Consequently, the Web services involved in the session have to keep the session information until the end of the session, even when they have finished their job.
- ❑ In the decentralised solution, the security of the session is dependent upon the security of every session partner. The strength of a chain is determined by the weakest link. Therefore, when a session partner crashes or becomes compromised, it maybe be fatal to the entire session.

Overall, the decentralised solution traverses the identifiers through session partners and its security is generated on the assumption that all the session partners are secure and trustable. If a session partner is compromised, the security of all the session may be easily compromised in many cases. Compared with the centralised solution, its fatal drawback is its inability of providing session partners real-time session information. After evaluating these two solutions, we finally select the centralised one.

4.6 Summary

In this chapter, we introduced the implementation of our experiment. The main part of the experiment is concentrated on proving the linear increase of time consumption when introducing Web service instances into the session. Since it is very difficult to generate a large number of Web services executed on different computers respectively, we develop an experiment to simulate the working environment of our protocols. In order to make our solution more convincing, we use an analytic model to further prove the experimental results. Furthermore, we describe a decentralised solution and compare it with the solution described in Chapter 3. In the next chapter, we will discuss the conclusion from the research and mention future work.

Chapter 5 Conclusions and Future Work

5.1 Conclusions

Web service is a new technology that emerged several years ago. People believe that this technology is able to help corporations generate e-business solutions which are more effective and dynamic than ever. Unfortunately, the lack of proper solutions to secure Web services is a major obstacle to large-scale commercial usage of the approach. The main goal of this research is to explore multi-party authentication issues for Web services and to develop a multi-party authentication system for Web services capable of providing reliable session security service and management to a Web service business.

Up until now, the prototype system has been generated and the results obtained from the experiment show that the session authentication system could provide trustable session key management:

- ❑ A Web service instance within one particular session can select some unique identifier among the session partners, and with this identifier, it will not be confused with others.
- ❑ The public key algorithm is involved so that the identifiers cannot be forged. Supplementary to this, a measure is provided for the session partners to validate the identity of the object who it is communicating with.
- ❑ The mistake of sending the messages to the service instance within different session is avoided. In the experiment, some errors were deliberately inserted into the experiment system, for instance, changing the identifiers of the session partners and changing the private keys of the session partners. These errors were detected successfully.

The experiment system employs Session Authority, which is a fixed, highly secure trusted third party, to manage all the secure information of the session partners and so it is a centralised management mechanism. Therefore, our system inevitably suffers from the single point failure and single point attack.

On the contrary, the distributed management mechanism requires all the group numbers take part in the management to overcome the shortcomings mentioned above. However, the distributed management is notoriously complex and computationally heavy [AMI00].

Currently, some algorithms (decentralised group key algorithms) are mentioned for the distributed key management in the peer-to-peer groups in some paper [AMI00, KIM00 (1)], and we have introduced them briefly in the previous chapter. Essentially, these algorithms are only suitable for relatively small groups [KIM00 (1)]. Furthermore, they only can generate a general key for all the group partners. Therefore, they are not suitable to resolve the multi-party session authentication issue we discussed above.

The centralised management mechanism has its irreplaceable advantages, simplicity and effectiveness. Nowadays, nearly all the most popular security systems (e.g., Kerberos, PKI [GER98, HOU02], Kryptoknight, etc.) make use of some kind of trusted third party to manage the secure information. Thus after analysing the models of the centralised solution and the distributed solution separately, the centralised system proves to be the choice that is more sensible.

5.2 Future Work

The session management of Web services is still a new field in the area of Web service security. The system developed has great potential to be improved, and it will act as the foundation for the future search.

5.2.1 Searching for Potential Candidates of the Public-Key Algorithm

In the experiment, the Diffie-Hellman algorithm is used to generate the secret key among the session partners. The Diffie-Hellman algorithm is a very famous key agreement algorithm and is wildly employed by existing security systems. However, it still makes sense to attempt to employ other key agreement algorithms to our security system and compare their performances, so that the optimum one may be selected.

Up until the present, the performance of the ECC (Elliptic Curve Cryptography) algorithm [STI02] is checked in the experiment and its performance has been compared with the Diffie-Hellman algorithm (see Figure 36).

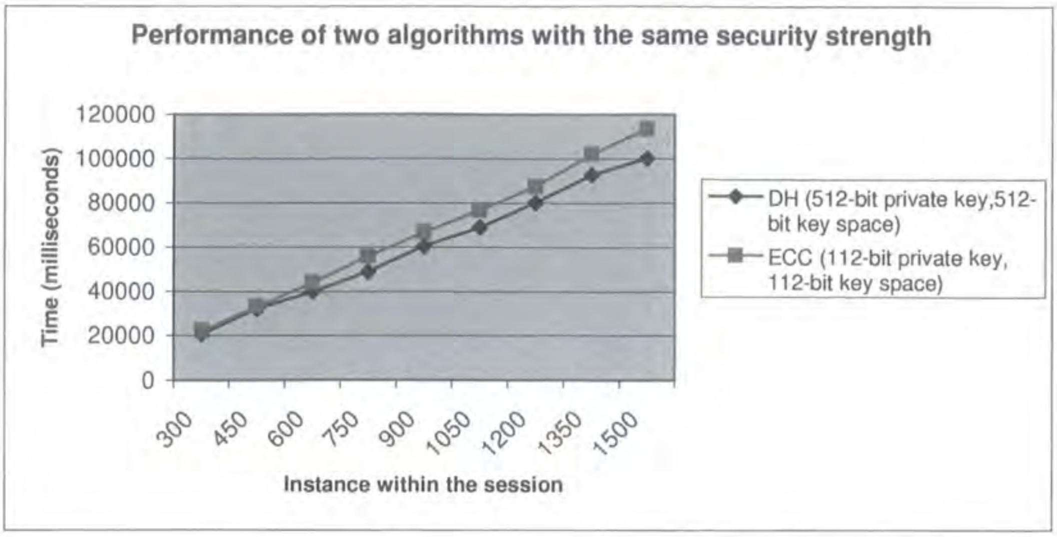


Figure 36 Comparison between the Diffie-Hellman algorithm and the ECC algorithm

In the experiment, the ECC algorithm does not show any speed advantage in the competition with the Diffie-Hellman algorithm. Nevertheless, this does not mean that the Diffie-Hellman algorithm is better than the ECC algorithm. In practice, several optimisations for the ECC algorithm can be employed to reduce the computational consumption. In the future, the optimised measures for the ECC algorithm will be implemented and other asymmetric security algorithms will continue to be explored in order to improve the performance of the system.

5.2.2 Semantic Issues of Session Management

In the session management system, the CA action model is leveraged to manage Web service sessions. CA actions have a good mechanism to handle the exceptions that occur in a distributed system. However, before it is integrated into a Web service environment, many semantic issues must be explored. For instance, the system comprised of Web services is loss-coupled. Different Web services may be developed and maintained by different organizations. It is normal that they select different ways to express exceptions and handle them. Therefore, it is necessary to find an efficient way to help the CAA manager understand the meanings of the exceptions thrown by different roles so that it can effectively organize the roles to handle these exceptions.

Beside the work cited above, there are still many things left to do. For instance, new security protocols for Web service are continually emerging. Some of these protocols can be integrated into our solutions. When the authentication system was designed, there was no security protocol for Web service specifying how to generate the secure conversation between two Web service instances. Therefore, we define the conversation messages in our solution by ourselves. Moreover, in our design, the MAC (message authentication code) tehcnique is necessary to prove the origin of the message and it should be attached to the SOAP message with WS-Security protocol. Until the experiment is finished, Sun Corporation has not provided the APIs for this protocol. So the present experiment is aimed at evaluating the scalability of the instance ID Authenticator protocol, and the secret keys were transported in plain text. In the future, the WS-Security and other new emerged security protocols (e.g., WS-SecureConversation) will be integrated into the experiment system, and performance of the system will be further evaluated.

In brief, there is great potential for future work on our solution. In our opinion, at moment the most pressing need is for an effective and optimized mechanism to distribute and manage the identifiers for session partners.

In the field of web service session management, there is still much to explore.

5.3 Acknowledgements

My thanks to my supervisor Jie Xu, and my father and mother for all their support.

References

[ABA97] Martín Abadi, "Explicit Communication Revisited: Two New Attacks on Authentication Protocols," *IEEE Transactions on Software Engineering Vol.23, No. 3,* pp.185-186, Mar. 1997.

[AMI00] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik, "Secure Group Communication in Asynchronous Networks with Failures: Integration and experiments," *Proc. the 20th IEEE International Conference on Distributed Computing Systems,* pp. 330–343, Apr. 2000.

[ATE00] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik, "New Multiparty Authentication Services and Key Agreement Protocols," *IEEE Journal on Selected Areas in Communication,* 2000.

[ATK02] Bob Atkinson, Giovanni Della-Libera, Satoshi Hada, Maryann Hondo, Phillip Hallam-Baker, Johannes Klein, Brian LaMacchia, Paul Leach, John Manferdelli, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, John Shewchuk, and Dan Simon, "Web Service Security (WS-Security)," http://www-106.ibm.com/developerworks/webservices/library/ws-secure/, Apr. 2002.

[BAR01] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon, "XML-Signature Syntax and Processing," http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/, Aug. 2001.

[BEL91] Steven Bellovin, "Limitations of the Kerberos Authentication System," *Proc. Winter 1991 Usenix Conference,* pp. 253-267, Jan. 1991.

[BEQ02] Henry Bequet, Meeraj Moidoo Kunnmpurath, Sean Rhody, and Andre Tost, "Beginning Java Web Services," Wrox press ltd, 2002.

[BER02] A. Berfield, P.K. Chrysanthis, I. Tsamardinos, S. Banerjee, and M.E. Pollack, "A Scheme for Integrating E-Services in Establishing Virtual Enterprises," *Proc. 12th International Workshop on Research Issues in Data Engineering: Engineering e-Commerce/ e-Business System* , pp. 134-142, Feb. 2002.

[BLA97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes, "Key Agreement Protocols and their Security Analysis," *Proc. 6th IMA International Conference on Cryptography and Coding, vol. 1355 of LNCS,* pp. 30-45, Sep. 1997.

[BOO03] David Booth, Michael Champion, Chris Ferris, Francis McCabe, Eric Newcomer, and David Orchard, "Web Services Architecture," http://www.w3.org/TR/2003/WD-ws-arch-20030514/, 2003.

[BOX00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsn, Satish Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1," http://www.w3.org/TR/SOAP/, May 2000.

[BOY01] John Boyer, "Canonical XML Version 1.0," http://www.w3.org/TR/2001/REC-xml-c14n-20010315, Mar. 2001.

[BRE01] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater, "Provably Authenticated Group Diffie-Hellman Key Exchange," *Eighth ACM Conference on Computer and Communication Security*, pp. 255–264, Nov. 2001.

[BRO02] David K. Broberg, "Man-In-The-Middle or Middleman," http://www.digitaltelevision.com/2002/may/expert_broberg.shtml, May 2002.

[CER02] Ethan Cerami, "Web Services Essentials," O'Reilly, Feb. 2002.

[CHR01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," http://www.w3.org/TR/wsdl, Mar. 2001.

[COW02] John Cowan and Reuters, "Extensible Markup Language (XML) 1.1," http://www.w3.org/XML/, Oct. 2002.

[DEN81] Dorothy E. Denning and Giovanni Maria Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM, 24(8),* pp. 533-536, Aug. 1981.

[GAR97] Simson Garfinkel and Gene Spafford, "Web Security & Commerce," O'Reilly, Jun. 1997.

[GER98] Ed Gerck, "Overview of Certification Systems: X.509, CA, PGP and SKIP," http://www.iks-jena.de/mitarb/lutz/certification/mc/cert.htm, Apr. 1997.

[GIB01] Andy Gibbs, "What is UDDI," http://www.nwfusion.com/newsletters/techexec/2001/00932451.html, Jun. 2001.

[GOL96] D. Gollmann, "What Do We Mean by Entity Authentication," *Proc. 1996 IEEE Symposium on Security and Privacy,* pp. 46-54, May 1996.

[GON93] Li Gong, T. Mark A. Lomas, Roger M. Needham, and Jerome H. Saltzer, "Protecting Poorly Chosen Secrets from Guessing Attacks," *IEEE Journal on Selected Areas in Communications, Vol.11, Issue 5,* pp. 648-656, Jun. 1993.

[HAD02] S. Hada and H. Maruyama, "Session Authentication Protocol for Web Services," *Proc. 2002 Symposium on Application and the Internet Workshops*, pp. 158-165, Jan. 2002.

[HAL03] Phillip Hallam-Baker, "XML Key Management Specification (XKMS) Version 2.0," http://www.w3.org/TR/xkms2/, Apr. 2003.

[HOD02] Jeff Hodges and Eve Maler, "Glossary for the OASIS Security Assertion Markup Language (SAML)," http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-01.pdf, May 2002.

[HOU02] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key

Infrastructure Certificate and Certificate Revocation List (CRL) Profile," http://www.ipa.go.jp/security/rfc/RFC3280-00EN.html, Apr. 2002.

[IMA02] Takeshi Imamura, Blair Dillaway, and Ed Simon, "XML Encryption Syntax and Processing," http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, Dec. 2002.

[KAU02] Charlie Kaufman, Radia Perlman, and Mike Speciner, "Network Security Private Communication in a Public World," Prentice Hall, Apr. 2002.

[KIM00] Y. Kim, S. Kang, D. Kim, J. Bae, and K. ju, "WW-FLOW: Web-Based Workflow Management with Runtime Encapsulation," *IEEE Internet Computing*, vol. 4, no. 3, pp. 55-64, 2000.

[KIM00 (1)] Y. Kim, A. Perrig, and G. Tsudik, "Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups," *Proc. ACM CCS-7*, pp. 235–244, 2000.

[KRE01] Heather Kreger, "Web Services Conceptual Architecture (WSCA 1.0)," http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, May 2001.

[KWO97] Taekyoung Kwon, Myeong Kang, and Jooseok Song, "An Adaptable and Reliable Authentication Protocol for Communication Networks," *Proc. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution,* pp. 737-744, Apr. 1997.

[LEY01] F. Leymann, "Web Services Flow Language (WSFL 1.0)," http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.

[MEN95] S. Mendes and C. Huitema, "A New Approach to the X.509 Framework: Allowing a Global Authentication Infrastructure without a Global Trust Model," *Proc. 1995 Internet Society Symposium on Network and Distributed System Security*, pp. 172-190, Feb. 1995.

[MEN96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "Handbook of

Applied Cryptography," CRC Press, Oct. 1996.

[NAK02] Y. Nakamur, S. Hada, and R. Neyama, "Towards the Integration of Web Services Security on Enterprise Environments," *Proc. 2002 Symposium on Applications and the Internet Workshops*, pp. 166-177, Jan. 2002.

[NAK02 (1)] S. Nakajima, "On Verifying Web Service Flows," *Proc. 2002 Symposium on Applications and the Internet Workshops*, pp. 223-224, Jan. 2002.

[PAT97] Sarvar Patel, "Number Theoretic Attacks on Secure Password Schemes," *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 236-247, 1997.

[RAN99] Brian Rrandell, "Fault Tolerance in Decentralized Systems," *Proc. 4th International Symposium on Autonomous Decentralized Systems*, pp. 174-181, Mar. 1999.

[RAY00] Jean-François Raymond and Anton Stiglic, "Security Issues in the Diffie-Hellman Key Agreement Protocol," http://citeseer.nj.nec.com/cache/papers/cs/22803/http:zSzzSzwww.geocities.comzSzj_f_raymondzSzmesarticleszSzdhshort.pdf/security-issues-in-the.pdf, Dec. 2000.

[SCH96] Bruce Schneier, "Applied Cryptography (Second Edition)," John Wiley & Sons, 1996.

[STA98] W. Stallings, "Cryptography and Network Security, Principles and Practice (second Edition)," Prentice Hall, 1998.

[STA00] William Stallings, "Network Security Essentials: Applications and Standards," Prentice Hall, 2000.

[STE98] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: A New Approach to Group Key Agreement," *The 18th International Conference on Distributed Computing Systems*, pp. 380-387, May 1998.

[STI02] Douglas R. Stinson, "Cryptography Theory and Practice," Chapman & Hall / CRC, 2002.

[SYV94] Paul Syverson, "A Taxonomy of Replay Attacks," *Proc. 7th IEEE Computer Security Foundations Workshop*, pp. 187–191, 1994.

[THA01] S. Thatte, "XLANG-Web Services for Business Process Design," http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, May 2001.

[TID00] Doug Tidwell, "Web Services -- the Web's Next Revolution," http://www-106.ibm.com/developerworks/webservices/edu/ws-dw-wsbasics-i.html, Nov. 2000.

[TOO03] Annraí O'Toole, "Web service-Oriented Architecture: The Best Solution to Business Integration," http://www.ctoupdate.com/ctoupdate-65-20030815WebServiceOrientedArchitectureTheBestSolutiontoBusinessIntegration.html, 2003.

[VAS01] V. Vasudevan, "A Web Services Primer," http://www.xml.com/pub/a/2001/04/04/Webservices/, Apr. 2001.

[WU99] Thomas Wu, "A Real-World Analysis of Kerberos Password Security," *Proc. 1999 Internet Society Network and Distributed System Security Symposium,* Feb. 1999.

[XU95] J. Xu, B. Randell, A. Romanvosky, C. Rubira, R.J. Stroud, and Z. Wu, "Fault Tolerance in Concurrent Object-Oriented Software through Coordinated Error Recovery," *Proc. 25th Int'l Symp. Fault-Tolerant Computing,* pp. 499-508, Jun. 1995.

[XU99] J. Xu, B. Randell, A. Romanvosky, R.J. Stroud, A.F. Zorzo, E. Canver, and F.V. Henke, "Rigorous Development of a Safety-Critical System Based on Coordinated Atomic Actions," *Proc. 29th Annual International Symposium on Fault-Tolerant Computing,* pp. 68-75, Jun. 1999.

[YAN02] J. Yang, M.P. Papazoglou, and W.V.D. Heuvel, "Tackling the Challenges of Service

Composition in E-marketplaces," *Proc. 12th International Workshop on Research Issues in Data Engineering: Engineering e-Commerce/ e-Business System*, pp.125-133, Feb. 2002.

[YAN02 (1)] Andrew Yang, "XML Web Services Security Issues," http://www.xwss.org/articlesThread.jsp?forum=34&thread=648, Apr. 2002.