Multi-party authentication in Web Services

Abstract:
Web Services are rapidly evolving, emerging technologies. Since they are based on message exchanges over the net, security is a major concern. A Web Service may be composed of multiple service instances taking part in several business workflows simultaneously. It is essential to establish trust relationships amongst the different service instances. Standard Web Service technologies such as SOAP, XML, SOAP-DSIG, address the security concerns to some extent. However it is the dynamic behavior of multiple parties participating in a workflow that introduces new security challenges. It is necessary to authenticate session participants, without prior knowledge of all parties, participating in the session.
This report discusses a session-oriented multi-party authentication protocol[], which establishes trust relationships among such session participants. It also discusses another design for multi-party authentication[] where each service instance of a given session is provided with a unique identifier.

Keywords: Web services, SOAP, WSDL, UDDI, Authentication,

1. Introduction
Web services are Internet-based, self-contained, modular applications that can be published, located, and invoked across the Web. They can communicate with other components automatically without human intervention. They perform functions, which can be simple requests to complicated business processes. Examples of typical Web services are, airline ticket reservation process, real time travel advisory, or sometimes even a restaurant review article. Once a Web Service is deployed, other applications (and other Web Services) can discover and invoke the deployed service. This interoperability allows businesses to dynamically publish, discover and aggregate a range of Web services through the Internet to more easily create innovative products, business processes and value chains. Use of Web Services on world-wide-web is expanding rapidly with growing need for interoperability and application-to-application communication. The rest of the report is organized as follows:

1.1 Web Service Architecture
At the heart of the Web Service Architecture is the need for program-to-program communications. In order for that communication to take place, it is essential to first describe the Web Service itself in detail, so that other programs can understand it and connect to it. That is exactly what XML does. Essential components of a Web service are: XML, SOAP, WSDL and UDDI.

- Extended Markup Language (XML) is the universal platform used for deploying and delivering these applications on a global scale. It provides a universal data exchange standard that allows access to data irrespective of its format or location. The provider hosts the XML service description for the Web service, which includes details about the service, including its location, transport protocols and message formats it uses.

- Web Service Description Language (WSDL) is the language used to create service descriptions. It is able to create descriptions not only about the location of the service and
how to run it, but also higher-level information, such as what business is hosting the service, the kind of service it is, keywords associated with the service and similar information.

- Simple Object Access Protocol (SOAP) is the protocol through which the service provider, service registry and service requestor communicate. It's an XML-based technology, used to exchange structured data between network applications. SOAP is used to publish the service description to a service registry. It describes what is on the message and who should deal with it. Similarly, all other interactions between service registry, service requestor and service provider are done via SOAP i.e it enables systems to talk to one another and make requests.

- Universal Description Discovery and Integration (UDDI) is the directory technology used by service registries that contain the description of Web services and that allows the directory to be searched for a particular Web service. UDDI is in essence a Yellow Pages that can be used to locate Web services. There can be both private and public UDDI directories.

Figure 1: Execution of Web Services

1.2 Typical Web Service Scenario

The following scenario demonstrates a business-level transaction involving a set of web services. Consider an industrial company that purchases Iodine from a chemical manufacturer on the Web. In order for the buyer to purchase the Iodine, she requires additional value-added services provided by third parties, such as shipping with specific delivery terms, payment financing,
casualty insurance, and government compliance for safe transport. So for the business transaction to be completed all these services have to be available. The software providing the top-level business transaction needs to coordinate with each of the participating web services. These include (1) the chemical provider's inventory system; (2) an insurance policy service to insure the product being shipped; (3) a financing service to ensure payment according to vendor terms; (4) a transportation service to guarantee timely shipment and delivery of product; and (5) a regulatory service to ensure compliance with government safety requirements.

![Diagram of typical web scenario](image)

**Figure 2: Typical Web Scenario**

### 2. Web Service Security

Since Web services are based on message exchanges on the net with the possibility of dynamic short-term relationships, security is a major concern. Most of the application internals are exposed to the outside world. As the application is closer to the data it opens room for security threats.

Hacking and traffic snooping problems are solved easily, using SSL at the protocol layer, and encrypting SOAP messages. Authentication and identity management are two of the most significant Web Service security problems as the transactions are conducted between two computers. For e.g. many companies, like Deutsche Bank, are building portals that call on Web services to gather data from back-end applications. The problem is that those applications don't know where the request is coming from.

Broadly Web Services security requires:

- **Authentication**: Establishing identity of user by providing a set of credentials. In return user receives a security token that can be used to access the server.
- **Authorization**: Establishing what a user is allowed to do.
- **Confidentiality**: Ensuring that only the intended recipient can read the message, accompanied by encryption.
• **Integrity**: Ensuring that the message has not been tampered with, generally accomplished with digital signatures.
• **Non-Repudiation**: Requiring that neither the sender nor the receiver can refute having sent the message.

### 2.1 Threats and Countermeasures

Malicious web service threats typically fall into one of three categories:

- Identity threats, such as authentication attacks, eavesdropping etc.
- Content-borne threats, which are attacks with elements in the actual XML payload, such as XML viruses.
- XML Denial of Service (XdoS), which are new application-level versions of network level DoS attacks.

#### 2.1.1 Identity Threats

The top Identity threats directed at Web services are:

- Unauthorized access
- Parameter manipulation
- Network eavesdropping
- Disclosure of configuration data
- Message replay

![Figure 3: Top threats and attacks directed at Web services](image)

**Unauthorized Access**

Weak authentication and authorization can be exploited to gain unauthorized access to sensitive information and operations. These could be taken care of by either using password digests, or Kerberos tickets or X.509 certificates in SOAP headers for authentication. Role based authentication could also be used to restrict access to Web services.
Parameter Manipulation
Parameter manipulation refers to the unauthorized modification of data sent between the Web service consumer and the Web service. For example, an attacker can intercept a Web service message, perhaps as it passes through an intermediate node en route to its destination; and can then modify it before sending it on to its intended endpoint. To prevent this the messages could be digitally signed and encrypted.

Network Eavesdropping
With network eavesdropping, an attacker is able to view Web service messages as they flow across the network. For example, an attacker can use network monitoring software.

Transport level encryption such as SSL or IPSec could be used if one can control both endpoints. The message payload could be encrypted to provide privacy. This approach would work in scenarios where the message travels through intermediary nodes route to final destination.

Message Replay
Web service messages can potentially travel through multiple intermediate servers. With a message replay attack, an attacker captures and copies a message and replays it to the Web service impersonating the client. The message may or may not be modified. The threat of message replay could be countered by use encrypted communication channel like SSL, encryption of message payload, and a unique message ID or nonce with each request to detect duplicates.

Disclosure of Configuration Data
There are two main ways in which a Web service can disclose configuration data. First, the Web service may support the dynamic generation of Web Service Description Language (WSDL) or it may provide WSDL information in downloadable files that are available on the Web server. This may not be desirable. Second, with inadequate exception handling the Web service may disclose sensitive internal implementation details useful to an attacker. This could be tackled by authorizing access to WSDL files using NTFS permissions or by removing WSDL files from Web Server.

2.1.2 XML Virus Attacks
 Content borne attacks[] are generally intended to affect actual applications that run Web Services after tunneling unnoticed through the security infrastructure. Two examples are SQL injection attacks and buffer over-flow attacks. SQL injection is the practice of inserting malicious SQL statements into XML to disrupt back-end systems. If a Web Service connected to a database doesn’t validate SQL, an incoming XML message containing rogue XML statements could break out of the expected database query and be used to obtain unauthorized information or data. A buffer overflow attack is aimed at the service end-point and capitalizes on vulnerabilities there, such as not setting aside enough memory to deal with a large variety of inputs. Protection against content attacks requires robust parsing and XML Schema validation capabilities.

2.1.3 Denial of Service Attacks:

The first widespread XDoS attack was the ‘entity attack’ where unprivileged users could use completely correct entity declarations in an XML message to wreak havoc on an unprotected
XML 1.0 standard-compliant parser. Recursive entity declarations can cause the parser to shut-down with an out-of-memory error. Inadvertent DoS attacks can occur as a result of simple human error, such as a programmer mistakenly sending 100 requests per second instead of 10.

***********Does this really fit in here????????***********

2.2 Web Service Scenario: Challenges and Issues

SOAP messages are sent from an initial SOAP sender to an ultimate SOAP receiver along a SOAP message path consisting of zero or more SOAP intermediaries that process and transform the SOAP message. A challenge is to preserve security properties of the SOAP message from the initial SOAP sender to the ultimate SOAP receiver. Transport layer security mechanisms such as HTTP over TLS may be used to secure messages between two adjacent SOAP nodes, whereas message layer security mechanisms defined in the Web Services Security standard must be used in the presence of intermediaries or when data origin authentication is required. Security headers may contain multiple Security Tokens, Security Token References, Nonces, Signatures, Encrypted Keys, Encrypted Data, and at most one Timestamp. Each security header is targeted to a specific SOAP actor. A SOAP message may contain multiple security headers, however each must be targeted to a different SOAP actor. Each security header may contain multiple Security Tokens.

SOAP messages are composed of XML elements. Elements may be signed and/or encrypted by being referenced from a Signature or a Reference List within an Encrypted Key. Individual elements within a message may be referenced from multiple Signatures and/or Reference Lists and messages may be composed of signed and/or encrypted elements from other messages. As intermediaries process messages, they potentially sign and encrypt new and pre-existing data, as well as consume signed and encrypted data targeted at a SOAP actor that they portray. It is important to preserve the security context of the message as it undergoes these transformations.

Traditionally SSL, TLS, VPNs and IPSEC are some of the common ways of securing content. However these are point-to-point technologies. They create a tunnel through which data can pass. IPSEC can authenticate hosts on either side of the communication. With SMIME (Secure Multi-purpose Internet Mail Exchange) protocol, data could be sent digitally signed and encrypted over the Internet. Both these protocols require that the communicating parties have persistent identities. However these protocols cannot be applied to authenticate participants of dynamically generated sessions. Web Services require more granularity. They have to maintain secure content and control according to their security policies. Following is a set of challenges:

- Inter-enterprise Web services are dealing with un-trusted clients
- End-to-end isn’t just point-to-point. The creator of the message wrote the payload but intermediaries may touch or rewrite the message afterwards.
- Clients and services do not have a way to negotiate their mutual constraints and capabilities before interacting.
2.4 SSL is Not Adequate for Securing Web Services

SSL along with TLS is used to provide transport level security for Web Service authentication. Between the web site and the web service there is a need for persistent message security for SOAP documents. SSL is inadequate for this type of security. While SSL encrypts the data stream, it doesn't support end-to-end confidentiality; it leaves the data exposed between the web site and the web service provider. The limitations can be summarized as follows:

- SSL provides point-to-point security or operates between end-points (and not applications), but for web services we need end-to-end security in which multiple intermediate nodes could exist between the two end-points. In a web services environment, there could be multiple XML-based business documents going through multiple intermediary nodes and it will be difficult for such nodes to participate in security operations in an integrated fashion.

- SSL operates at the transport level and not at the message level. In other words, messages are protected only while in transit. That is, you cannot save the message for later to prove that it hasn't been modified.

- SSL provides confidentiality and sender/receiver authentication but doesn't support non-repudiation. Using SSL, a communicating partner cannot prove that the other party has performed a particular transaction. That is, SSL doesn't support an end-to-end audit trail from service request to service response.

- SSL doesn't support element-wise signing and encryption. Given a large XML order document, one may want to only sign or encrypt the credit card info...and that is difficult in SSL. This is because SSL is a transport-level security scheme as opposed to a message-level scheme.

************Can I move this to appendix************

2.5 Security Standards for Securing Web Services:

- **XML Signature**: A standard specification developed jointly by the W3C and IETF (Internet Engineering Task Force). An XML signature is equivalent to a digital signature; it can be used to digitally sign portions of an XML document. It is used with SOAP messages.

- **XML Encryption**: A standard specification developed by the W3C proposes to encrypt portions of XML documents. This specification can be used to assure confidentiality in case of a security context ranging over several SOAP intermediaries. To do that, portions of the SOAP message are kept confidential from SOAP intermediaries while the message is in transit.

- **XML Key Management Specification (XKMS)**: Developed by the W3C to allow clients to obtain cryptographic key information (such as keys and certificates). It also describes protocols for key management such as registration and revocation, suitable to be used together with XML Signature and XML Encryption.

- **Security Assertions Markup Language (SAML)**: Defined by OASIS, it outlines a framework for exchanging authentication and authorization information. It is used for
user identity assertions and for asserting actions performed by various elements of an enterprise infrastructure.

- **XML Access Control Markup Language (XACML):** The primary goal of this specification is to standardize access control language in XML syntax. Such a language can be used to express access control policies like who can do what and when. The Web Services Policy Language (WSPL), which is based on XACML, is a generic language for expressing policy information.

- **WS-Security:** The OASIS Web Services Security specification provides a way to ensure that messages remain confidential, haven’t been tampered with, and are actually from senders asserting to have sent them. WS-Security specifies the use of XML Digital Signature and XML Encryption with SOAP, enabling the application developer to insert a security token that identifies the original sender and optionally captures information about intermediate destinations of the XML message. Security tokens can be as simple as a name, IP address, and password or as complex as a PKI certificate or a SAML assertion. The specification supports common encryption algorithms and techniques. The standard provides ways to encrypt all or just parts of the XML in the message. Selective encryption and signing also lets senders add different signatures and keys to parts of a single document that are designated for different recipients. However, the process of identifying which data is supposed to be confidential, based on information listed in a header manifest, adds overhead to a system.

2.6 Firewalls and Appliances (Where does this fit in??)

A principal Web services security problem is that XML transfers everything over HTTP, allowing traffic to pass through firewalls via TCP port 80. This enables easy communication between networks whose firewalls block all ports except the ones that Web protocols use. However, firewall penetration also creates security concerns. In light of this threat, some firewalls have been extended to recognize, examine, and filter XML and SOAP traffic.

Web services security technology is still being developed and has not stabilized enough to inspire confidence. “Web services security hasn’t progressed beyond securing communications between trusted parties”[].

3.1 Authorization using trusted third party

In an open e-Business world, many unfamiliar business entities may need to communicate and access the Web services. An approach suggested in [] is to set up a security assertion authority as a trusted third party. Then two parties could interact with each other via this authority. For e.g a buyer could send a request to the authority with a user id and password. The authority could authenticate the buyer and issue a document containing authentication and attribute assertions. The buyer could then send a purchase order to the supplier, attaching the security assertion as a SOAP header. The supplier could perform authorization relying on the received assertion. A security assertion document is a kind of a ticket. The identity of the carrier is not authenticated by the receiver. This implies the possibility of a single sign-on for distributed multiparty Internet services. By providing trusted third parties to play the role of assertion authorities, an open-
ended, flexible e-business infrastructure is proposed. SAML provides a technology basis for such a business structure.

3.2 Need for Session Authentication for Web Services

A web service is a static long-lived entity with a unique identifier globally known. When it receives an initial request to participate in a business flow, a new instance is invoked to handle the requests pertinent to this flow see Fig <4>. We consider here a typical business flow scenario, which consists of three processes viz: a buyer process, a supplier process and a shipping process and multiple instances of each process. The buyer process initiates the session and invokes an instance, which sends a P.O to the supplier. The supplier starts an instance to handle this request and dynamically chooses a shipping company to fulfill this P.O, and sends shipment request to the shipper. The shipper creates its own instance to handle this request.

![Typical Business Flow Scenario](image)

Figure 4: Typical Business Flow Scenario

B.I Buyer Instance
S.I Supplier Instance
Sh.I Shipper Instance

The Buyer receives a notification from the shipper that goods are being shipped. The buyer may not know the Shipper, so in that case the question of trust looms up as a major issue. The notification may be sent by shipper over a SSL connection, signed by some C.A. This only tells the buyer that the shipping service is some authentic service but does not provide the guarantee that this is an instance of the session started by the buyer. The shipping service may have hundreds of service instances simultaneously participating in different transactions, serving different companies. If someone compromises/confuses this shipping instance a false notification may be generated. Thus the need to authenticate dynamically generated participants belonging to a particular session.
3.3 Hada - Maruyama’s Protocol

The protocol is split into two parts. A message authentication protocol transports authentication information between session participants and session management protocol is responsible for starting, running and ending a particular session. There is a session authenticator (SA) component, which is responsible for distributing session keys and authenticating messages.

3.3.1 The Message Authentication Protocol

The protocol is based on the assumptions that a service is a persistent long-lived entity with a globally known identifier such as a URL while a service instance is a transient, short-lived entity. There are two levels of message authentication protocols used in the proposed session management protocol.

1. **Session Authenticator**

   Allows service instances to mutually verify their transient membership. It is a protocol for the sending service instance to send a MACed SOAP envelope to the receiving instance, which may not yet have the session key. The following are the steps involved:
   
   i. Sending instance prepares SOAP envelope.
   ii. Can optionally use XML encryption to encrypt the envelope
   iii. Adds authentication information to SOAP header. This consists of session handle, message identifier, sending service’s certificate and receiving service’s certificate. This identifier is essential to avoid replay attacks. Session handle may also include expiration date.
   iv. Sending instance uses SOAP-DSIG to apply a MAC to the entire SOAP envelope under the session key. This MACed envelope is then sent to receiving service.
   v. On receiving MACed envelope, receiving service checks whether it already has a valid session secret for the session. If not, it obtains the same from session management protocol.
   vi. Having session key, receiving service instance validates MAC and accepts SOAP envelope after successful validation. If the envelope was encrypted then decryption is done.
   vii. Thus receiving instance receives authenticated message and session handle.

2. **Service Authenticator**

   This is the protocol for the sending web service to send a SOAP envelope to a receiving web service.
   
   i. Sending service prepares SOAP envelope to send.
   ii. The Authentication header is added
   iii. SOAP-DSIG is used to digitally sign the message. Optionally XML encryption could be used to encrypt the digitally signed envelope.
   iv. Signed (encrypted) envelope is sent to receiving service.
   v. Receiving service decrypts it, validates the signature, verifies the certificates specified as the receiver’s certificate as its own and accepts this as a message.
3.3.2 Session Management Protocol

The underlying assumption here is that the messages will be sent using service authenticator and SSL/TLS. Further assume the presence of entity session authenticator that manages the sessions. SA could be initiator of the session or any other instance. SA is responsible for

- Assigning session Ids.
- Creating session secrets.
- Maintaining status information for each session.
- Keeping participants informed of the status.
- Shutting down sessions.

The protocol has the following steps:

A session is created by sending a `<start-session>` request to the SA. Reply to `<start-session>` command contains session handle and session secret that were created for that session.

A would-be participant can send a `<join-session>` message to the SA and get back the session secret in return.

A participant can request shutting down a session by `<end-session>` request.

SA keeps track of information about a session by `<query-session>` request.

SA can send a `<notify-session>` message to its participants to notify shutting down of a session.

One participant can invite another to participate in the session by sending a `<forward-session>` message. This message needs to be protected from eavesdroppers as it contains the session secret. The SA must digitally sign the Admission policy admitting a participant to a session. Messages are sent using the message authentication protocol such as service authenticator and SSL/TLS as required.

3.4 Online Session Management

There are two ways for a participant to join a session, offline and online.

In the online mode, first the requester initiates a session by sending a start-session request to SA. It receives the session secret as a reply to this request. Using this secret it now sends an application to a Web service (denoted service#1 in fig <>), over the message authenticator. On receiving this message the service is given the session handle and asked to join the session. A new instance is spawned by the service and control is transferred. SA, following a join-session request admits this instance to the session. The same session key is also passed on to this instance. Since the instance has the session secret it now authenticates the received message and obtains payload. If the service instance needs to delegate its operation to some other service, it can send a message the other service via session authenticator. The same process is repeated till second service instance also obtains the session secret.

Thus ultimately all session participants obtain the same session secret and can trust each other once they are admitted to the session. See Fig <5>
3.5 Offline Session Management

The online scenario requires extra communication in order to acquire the session secret. By using the `<forward-session>` message, the requester can create a new service instance and send this message. The newly created instance now has the session secret. If it needs to further delegate to a third Web Service and is allowed to do so by the admission policy, it can re-forward this session. The requestor itself can act as SA in this scenario also. The `<forward-session>` message could be piggybacked on the `<notify-session>` messages to further reduce communication overhead during session management. However in the offline scenario, the list of session participants cannot always be maintained accurately.

3.6 Drawbacks of the Session Authentication Protocol

- The protocol does not provide a unique identifier for each participant in a session.
- An attacker who has compromised a Web Service instance and obtained the session secret can communicate with other session participants and gain their trust.
- The SA component does not have a measure to validate the identity of a Web Service instance that applies to enter a session. Any Web service instance as long as it holds the session ID can contact SA and apply to enter the session.

The Session Management protocol proposed by [] does not provide a unique identifier for each participant of the session, leaving loopholes for attackers who can compromise an instance. Dacheng Zhang and Jie Xu in their work [], proposed a protocol where, within a session each instance holds a unique key, which can therefore be used to identify the instance.

4.1 Instance ID Authenticator Protocol

The dynamism of Web service instances gives rise to the need for dynamic and unique identifiers for the instances of a given session. Within a session, each instance holds a unique security key, which can be used to identify that instance. The Diffie Hellman key exchange protocol is used to exchange a key safely among service instances. For e.g. if there are two instances A and B which agree on a large prime p and a number g which is a primitive root of p then g\text{mod} p, g^2 \text{mod} p, g^3 \text{mod} p, ....... g^{p-1} \text{mod} p are distinct integers from 1 to p-1 in some permutation. Following are the notations used in the protocol:

\begin{itemize}
  \item \(p\): a large prime number
  \item \(g\): a primitive root of \(p\)
  \item \(R\): random no. chosen by a service instance of the session
  \item \(g^R\): \(g^R \text{mod} p\) is the unique identifier used by the instance.
  \item \(K_{xy}\): security key shared by service instance \(x\) and \(y\).
  \item \(\text{MAC}_{xy}(M)\): MAC code for msg \(M\) under key \(K_{xy}\).
  \item \(U(R, g^R)\): Service Instance U with pvt msg \(R\) and identifier \(g^R\).
\end{itemize}

4.2 Basic Protocol Steps

A service instance wanting to communicate with another instance for the first time sends a \(<\text{start-communication}>\) message. The recipient on receiving and verifying this message sends back a Boolean value “true” or “false”.

1. Web service prepares a SOAP envelope in order to send a message to participants of a given session
2. The instance inserts the authentication information (i.e. session handle, identifier of sending instance, sender service’s certificate, receiver service’s certificate) into the header of the envelope.
3. Sending instance uses Diffie-Hellman algorithm to calculate a secret key with its private message and the identifier of the intended receiver.
4. It then applies MAC code generated with secret key to the SOAP message. Finally sending instance sends SOAP message to the receiving instance.
5. Upon receiving message, receiver instance uses its own private key and the identifier of the sender to regenerate the secret key. By comparing newly generated MAC with the MAC appended to message, the receiver can verify identity of the sender.

E.g. consider two instances A and B with identifiers \(g^x\) and \(g^y\). A sends \(g^x \text{mod} p\) to B and B sends \(g^y \text{mod} p\) to A. B now computes \(K_2= (g^x \text{mod} p)^y = g^{xy} \text{mod} p\).

Similarly A computes \(K_1= (g^y \text{mod} p)^x = g^{yx} \text{mod} p\).
Thus both have the same secret key $g^{xy}$ mod $p$.

### 4.3 Example Scenario

Consider the following scenario, which initially consists of a session manager service and a user instance (UI $(x, g^x)$).

1. UI contacts the session manager.
2. Session manager invokes service instance (MI$(y, g^y)$). Fig <2>

![Figure 6: Example of authentication process](image)

3. MI and UI exchange identifiers $g^x$ and $g^y$ and a session instance is initialized.
4. Suppose UI wants to contact service #1. A new service instance (NI$(z, g^z)$) is invoked to manage UI’s request.
5. UI and NI exchange identifiers $g^x$ and $g^z$ and some related information (e.g. URL of MI and $g^y$).
6. UI then needs to send <Introduction-of-new-Instance> message to MI, before MI accepts NI as a participant of the session. Fig <>.

![Figure 7: Interactions between UI and NI](image)

7. Suppose NI needs to contact another service #2. So another service instance (NI2$(n, g^n)$) is invoked by NI and recommended to MI.
8. 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.
9. On getting UI’s identifier from MI, MI2 will send a `<start-communication>` message to UI.
10. After receiving 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 then verifies MAC appended with the message and sends a reply back to NI2. See Fig. <4>

![Interaction between UI and NI2](image)

**Figure 8: Interaction between UI and NI2**

### 4.4 Key Observations
- Each service instance of a session has a private secret and a public identifier.
- The public identifier is transported over the Internet.
- Private secrets are kept by the instances securely.
- Attackers cannot get the key shared by the instance participating in a communication from the plain-text and MAC code.
- It is computationally difficult for an attacker to impersonate one or more participants.
- Service instance that attempts to join a session must be recommended by a session participant to “SA” first. So a malicious instance will be detected by session manager. This further improves the security of the system.

### 4.5 Session Management Protocol

The Multi-party authentication protocol stated above also takes care of complicated cases like a participating service instance having local operations not supposed to be known to other participants. The session management protocol uses co-ordinated atomic action mechanism to dynamically manage sessions. A CA action is a mechanism for co-ordinating multi-thread interaction. Web Services are concurrent execution threads. These when participating in a
given CA action, enter and leave the action synchronously. Within the CA action, operations on objects can be performed co-operatively.

In the session management protocol, a set of nested CA actions structure a workflow. Nested actions could be pre-defined or invoked dynamically. A service instance or a role in a CA action may only communicate with other roles in the same action. A CAA manager must ensure that there is no error or exception unsolved. There are separate protocol operations that provide functions to manage a set of nested CA actions.

4.6 Analysis of the protocol.

5. Own thoughts…..

6. Conclusions

7. Acknowledgements: