Secure Service Invocation in a Peer-to-Peer Environment Using JXTA-SOAP

Maria Chiara Laghi, Michele Amoretti, and Gianni Conte

Information Engineering Department, University of Parma, 43100 Parma, Italy
laghi@ce.unipr.it, {michele.amoretti,gianni.conte}@unipr.it
http://dsg.ce.unipr.it

Abstract. The effective convergence of service-oriented architectures (SOA) and peer-to-peer (P2P) is an urgent task, with many important applications ranging from e-business to ambient intelligence. A considerable standardization effort is being carried out from both SOA and P2P communities, but a complete platform for the development of secure, distributed applications is still missing. In this context, the result of our research and development activity is JXTA-SOAP, an official extension for JXTA enabling Web Service sharing in peer-to-peer networks. Recently we focused on security aspects, providing JXTA-SOAP with a general security management system, and specialized policies that target both J2SE and J2ME versions of the component. Among others, we implemented a policy based on Multimedia Internet KEYing (MIKEY), which can be used to create a key pair and all the required parameters for encryption and decryption of service messages in consumer and provider peers running on resource-constrained devices.

Keywords: service, peer-to-peer, security.

1 Introduction

Peer-to-peer (P2P) is clearly arising as a disruptive paradigm enabling highly scalable decentralized applications based on resource sharing. A peer-to-peer architectural model defines application-level protocols and policies for the construction and maintenance of an overlay network, where each peer node is itself a resource, being discoverable and contributing to the operation of the whole system.

While for many years the focus of P2P research has been on performance optimization of content sharing applications, there is now a growing interest for service-oriented P2P systems, in which peers offer consumable resources, *i.e.* resources that cannot be acquired (by replication) once discovered, but may only be directly used upon contracting with their hosts [11].

The convergence of service-oriented and peer-to-peer architectures requires standards for (1) overlay network construction and maintenance, (2) message routing among peers, (3) service advertising, discovery and interaction, (4) security. While SOAs have sound, widely accepted standard protocols covering

A.U. Schmidt and S. Lian (Eds.): MobiSec 2009, LNICST 17, pp. 202–213, 2009.

points (3) and (4), for example within Web Service technologies, on the other hand P2P systems lack of unified views and solutions.

Among others, Sun MicroSystems' JXTA is probably the most advanced attempt to provide a standard set of P2P protocols [12], covering all listed points to a certain degree. JXTA protocols should allow a vast class of networked devices (smartphones, PDAs, PCs and servers) to communicate and collaborate seamlessly in a highly decentralized fashion. The JXTA framework defines a naming scheme, advertisements, peergroups, pipes, and a number of core policies, while the JXTA middleware implements the specifications in Java and C++.

In a previous work [13] we introduced the JXTA-SOAP component, which is an official extension for JXTA, enabling Web Service sharing in peer-to-peer networks. Here we provide more implementation details, focusing on security aspects, for both J2SE and J2ME versions of JXTA-SOAP.

The paper is organized as follows. In section 2 we present related work on peer-to-peer security, focusing on service interaction issues. In section 3 we describe the architecture of JXTA-SOAP, with reference to adopted technologies. Section 4 illustrates secure service invocation mechanisms, with different policies depending on the characteristics of the device that hosts the peer. Finally we conclude the paper with a summary and an outline of future work.

2 Background

Making peer-to-peer systems secure is a significant challenge. A malicious node might give erroneous responses to requests, both at the application level, returning false data, or at the network level, returning false routes and partitioning the network. Moreover, the P2P system must be robust against a conspiracy of a malicious collective, *i.e.* a group of nodes acting in concert to attack reliable ones. Attackers may have a number of goals, including traffic analysis against systems that try to provide anonymous communication, and denial of service against systems that try to provide high availability.

Security attacks in P2P systems are classified into two broad categories: passive and active [5]. Passive attacks are those in which the attacker just monitors activity and maintains an inert state. The most significant passive attacks are:

- *Eavesdropping*, which involves capturing and storing all traffic between some set of peers searching for some sensitive information (such as personal data or passwords).
- *Traffic analysis*, where the attacker not only captures data but tries to obtain more information by analyzing its behavior and looking for patterns, even when its content remains unknown.

In active attacks, communications are disrupted by the deletion, modification or insertion of data. The most common attacks of this kind are:

- *Spoofing*, in which one peer impersonates another, or some outside attacker transforms communications data in order to simulate such an outcome.

- *Man-in-the-middle*, where the attacker intercepts communications between two parties, relaying messages in such a manner that both of them still believe they are directly communicating. This category includes data alteration between endpoints.
- *Playback or replay*, in which some data exchange between two legitimate peers is intercepted by the attacker in order to reuse the exact data at a later time and make it look like a real exchange. Even if message content is encrypted, such attacks can succeed so long as duplicate communications are allowed and the attacker can deduce the effect of such a repeat.
- *Local data alteration*, which goes beyond the assumption that attacks may only come from the network and supposes that the attacker has local access to the peer, where he can try to modify the local data in order to subvert it in some malicious way.

To cope with these attacks, security policies adopted at the overlay P2P network level usually consist of key management, authentication, admission control, and authorization. These are the strategies we took into account for securing consumer-to-service communication in JXTA-SOAP.

Transport-level security is mostly based on Secure Sockets Layer (SSL) and on its successor Transport Layer Security (TLS) that runs beneath HTTP, the most popular protocol for Web Services. HTTP is an inherently insecure protocol because all information is sent in clear text between unauthenticated peers over an insecure network. Transport-level security can be applied to secure HTTP; the purpose of TLS is to create a secure pipe between two web servers. Authentication occurs at the time the secure pipe is created, and confidentiality and integrity mechanisms are applied only while the message is in the secure pipe. When running over an TLS-protected session, the server and client can authenticate one another and negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. At the receiving endpoint the messages are decrypted by the server.

Once a Web Service transmits a message it does not always know where that message will be traveling to, and how it will be processed before it reaches its intended destination [14]. A number of intermediary services may be involved in the message path; a transport-level security technology will protect the privacy of the message contents while it is being transmitted between these intermediaries and number of additional technologies are required to facilitate the message-level security required for end-to end protection. WS-Security (Web Services Security Language) can be used to bridge gaps between disparate security models but also goes beyond traditional transport-level security to provide a standard end-to-end security model for SOAP messages through the use of header blocks that are sent with the message. Transport-level and message-level security mechanisms can be combined to obtain a higher level of security in service invocation.

In [8], security issues that occur in the underlying routing protocol are described, as well as fairness and trust issues that occur in file sharing and other P2P applications. The creation of a secure routing primitive is a technique that can be used to address this problem. In this model, application specific objects

are assigned unique identifiers, called keys, selected from the same id space. Each key is mapped by the overlay to a unique live node, called the key's root. Such a primitive ensures that when a non-faulty node sends a message with a specified key, the message reaches all non-faulty members in the set of replica roots with a very high probability. Implementing the secure routing primitive requires the solution of three problems: securely assigning identifiers to nodes, to prevent an attacker from choosing the value of identifiers assigned to the nodes he controls; securely maintaining the routing tables, in order to ensure that the fraction of faulty nodes appearing in the routing tables of correct nodes does not exceed, on average, the fraction of faulty nodes in the entire overlay; finally securely forwarding messaging assures that at least one copy of a message sent to a key reaches each correct replica root for that key with high probability.

In [6] a general architecture is proposed that enhances security aspects by leveraging trusted computing technology, which is built on a trusted platform module and provides a mechanism for building trust into the application layer. A trusted reference monitor (TRM) in the platform of each peer can monitor and verify the information a peer provides and ensure data authenticity. Using credentials, a TRM can also digitally sign data from a peer and provide verification mechanisms for a remote site. The proposed general trusted platform with a TRM assumes an homogeneous environment, *i.e.* each platform is evenly equipped with the necessary trusted computing hardware.

Intrusion detection is another important issue in any security framework, in particular for mobile ad hoc networks, for which attacks on nodes can disrupt communications. Any node can observe part of the total traffic, making distributed intrusion detection a suitable approach: each node is responsible for detecting intrusions in its local neighborhood. In this form of intrusion detection, a node's neighbors observe that node's external behavior and form a judgement about a node' "trustworthiness". In [7] the common problem of how to combine observational data from multiple nodes is afforded with the Dempster-Shafer evidence theory. This theory addresses the problem of combining evidence from multiple observers representing uncertainty the form of belief functions. The idea is that an observer can obtain degrees of belief about a proposition from a related proposition's subjective probabilities. The Dempster-Shafer theory, compared for example with the Bayesian approach, is appealing because it can handle uncertainty or ignorance.

3 JXTA-SOAP

The JXTA-SOAP component is an official extension for the Java version of JXTA middleware, enabling Web Service sharing in peer-to-peer networks. JXTA-SOAP has been designed having in mind ubiquitous computing needs, to reduce the complexity otherwise required to build and deploy peer-to-peer service-oriented applications. In a previous work we described the internal architecture of JXTA-SOAP, with the purpose of conceptualizing its main features at a high abstraction level [13]. In particular we focused on service deployment, publication, lookup and invocation.

JXTA peers use pipes in order to exchange messages and access available resources. JXTA messages are XML documents with ordered message elements which may contain any type of payload. Messages are the basic data exchange unit in JXTA and all protocols are defined as a set of messages exchanged between peers. Pipes provide an asynchronous, unidirectional and unreliable communication channel by default. However, bidirectional reliable channels may be provided on top of them. They offer two operation methods: unicast pipes, which allow one-to-one communications, and propagation pipes, which allow one-to-many. JXTA pipes are an abstraction and are not bound to a specific IP address or port. They have a unique ID and are published just like any resource in the JXTA network, so any peer may update them whenever its physical location changes. Both input pipes, used for message reception, and output pipes, used for message sending, are considered pipe endpoints, an actual destination in the physical network. JXTA-SOAP uses pipes for carrying consumer-to-service requests, and service responses.

The internal architecture of a JXTA-SOAP peer is illustrated in figure 1.

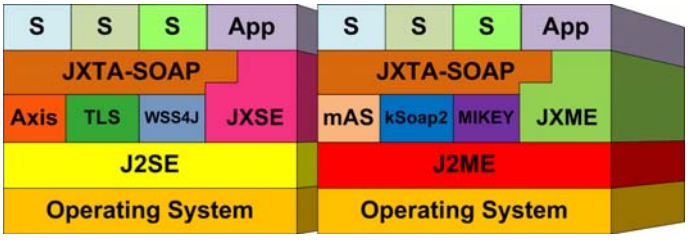

Fig. 1. Architectural layers of service-oriented peers based on JXTA-SOAP. The J2SE version (on the left) is based on Axis and JXSE, while the J2ME version (on the right) is based on kSoap2 and mAS.

We implemented two (interoperable) versions of JXTA-SOAP: J2SE-based, extending JXTA-J2SE, and J2ME-based, extending JXTA-J2ME. In the following we describe their features and the different technological solutions they rely on.

3.1 JXTA-SOAP for Java Standard Edition (J2SE)

The J2SE version of the JXTA-SOAP component supports service deployment, discovery, and invocation, with optional use of standard mechanisms to secure communications among peers. The core of the component is the Apache Axis engine (v1.4), which is a producer/consumer of SOAP messages. Usually Axis is deployed in a Web application server, such as Apache Tomcat, together with the implementations of the Web Services to be deployed, while client applications use the Axis Java API to create request instances. The Axis engine provides the processing logic, either client or server. When running, it invokes a series of Handlers according to a specific order which is determined by two factors - deployment configuration, and whether the engine is a client or a server. The

object which is passed to each Handler invocation is a MessageContext, *i.e.* is a structure which contains several important parts: 1) a "request" message, 2) a "response" message, and 3) a bag of properties.

Once a service instance has been initialized, the Axis engine may dispatch multiple remote invocations to it. After that, when the Axis engine determines that the service instance needs to be removed from service of handling remote invocations, it destroys it. In the implementation of the destruction functionality, the service object releases its resources.

For remote service invocation, a consumer peer needs to instantiate a Call object. JXTA-SOAP's Call class extends Axis' default one, overloading the use of service URLs with the use of the Service Descriptor and the public pipe advertisement of the service. To create Call instances, the peer uses the implementation of the Call Factory class provided by Axis.

3.2 JXTA-SOAP for Java Micro Edition (J2ME)

The J2ME version of the architecture supports Connected Device Configuration (CDC) and Personal Profile. We implemented the API which enables the development of peers that are able to deploy, provide, discover and consume Web Services in a JXTA-SOAP network. Since Axis is not available for the CDC platform, we adopted kSoap2 [2] as SOAP parser (for consumer functionalities) and, for service provision, we integrated the mAS [3] lightweight engine.

Service invocation is allowed by a kSoap2 based implementation of the Call Factory class. The latter instantiates a kSoap2's Soap Object, and sets all the properties for message exchanging through JXTA pipes. Soap Object is a highly generic class which allows to build SOAP calls, by setting up a SOAP envelope. We have maintained the same structure of J2SE-based version for Call Factory, to allow portability of service consumer applications from desktop PCs or laptops to PDAs. Internally, the Call Factory class creates a Soap Object passing references to the Service Descriptor, the public pipe advertisement of the service and the peergroup as parameters for the creation of the Call object.

After instantiating the transport using the Call Factory class, the consumer peer creates the request object, indicating the name of the remote method to invoke and setting the input parameters as additional properties. This object is assigned to a Soap Serialization Envelope, as the outbound message for the soap call; Soap Serialization Envelope is a kSoap2 class that extends the basic Soap Envelope, providing support for the SOAP Serialization format specification and simple object serialization. The same class provides a getResponse method that extracts the parsed response from the wrapper object and returns it.

For service provision, we integrated the Server class of the Micro Application Server (mAS) into the basic service class of the JXTA-SOAP API. mAS implements the Chain of Responsibility pattern [4], the same used in Axis. It avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request; receiving objects are chained and and the request passed along the chain until an object handles it. Moreover, mAS allows service invocation by users and service deployment by administrator; it also supplies

browser management of requests, distinguishing if the HTTP message contains a Web page request or a SOAP envelope.

4 Secure Service Invocation

Currently, JXTA-SOAP supports secure service invocation by means of two orthogonal mechanisms. The first one, *transport-level security*, allows to create a TLS-based secure channel which guarantees the integrity and confidentiality of exchanged information, by means of mutual authentication between parties (using X.509 certificates) and data encoding. The other approach is WSS-based *message-level security*, for which SOAP messages sent by service consumers contain security parameters (tokens) which are extracted by service providers to check for consumers' compliance with the security policy of the invoked service. Figure 2 summarizes the interaction of a peer with a discovered secure service.

The `net.soap.jxta.security.policy` package provides two important modules, *i.e.* `Policy` and `PolicyManager`. The latter is a class which allows to associate a service with a security policy (currently: TLS, WSS, or both). The `Policy` interface provides methods which are commonly implemented by all policy classes. They allow to extract authentication parameters from the client request message and to verify their correctness according with the service security requirements. If all parameters are validated, the client is allowed to invoke the service, otherwise the request is refused. Authentication parameters are stored in a vector of `SOAPService` class that is used to keep a list of authenticated peers.

Current implementations of the `Policy` interface are `DefaultTLSPolicy`, based on `JXTAUnicastSecure` pipes which subsume default (unsecure) JXTAUnicast pipes, and DefaultWSSPolicy, which uses Apache's WSS4J to provide SOAP messages with security headers and fill them with tokens. Policies are associated to services by means of two extensions of the `<Parm>` field of the `ModuleSpecAdvertisement`. The first extension is the `<security>` tag, whose value (true or false) indicates whether a service is secured or not. The other extension is the `<InvocationCharter>`, a XML document which is inserted in the `<Parm>` field of the ModuleSpecAdvertisement if the `<security>` tag is set to true. The secure invocation model is illustrated in figure 3 (TLS-based case), with particular emphasis on multithreaded handling of concurrent invocations.

The `net.soap.jxta.security.certificate` package provides classes that enable JXTA-SOAP to use X.509 certificates for authentication and to extract them from a PeerAdvertisement; in particular, it uses the Key Store associated to JXTA PSEMembershipService, that is initialized when the peer boots in the network group. The `net.soap.jxta.security.wss4j` package implements a WSSecurity class, that enables the client to create a custom security header for invocation requests, according to the Invocation Charter (i.e. encryption, body signature).

4.1 DefaultTLSTransport

TLS is the default technology used by JXTA-SOAP when the secure service is created. Each SOAP message between client and server is sent through a TLS

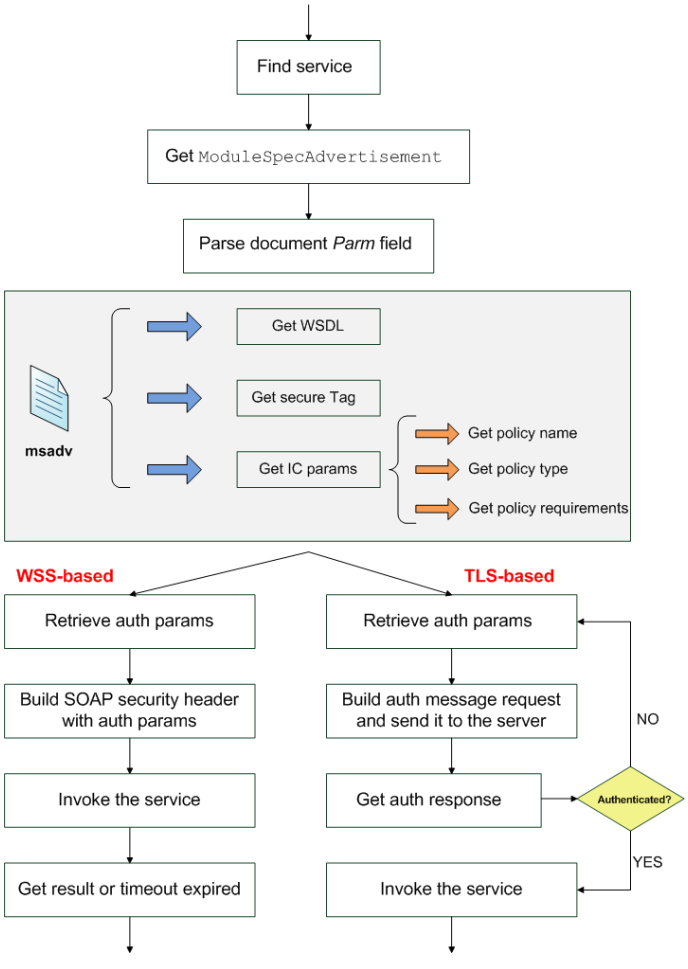

Fig. 2. Interaction with a discovered secure service

channel which requires previous authentication of involved entities. According to the implemented policy, a communication session is established, in which the client sends to the server its PeerAdvertisement with the peer self-signed certificate created by JXTA. The server is able to extract additional information about the requesting client (i.e. name, PeerID, group) and, if the authentication procedure succeeds, to update the list of authenticated peers with the corresponding entry:

[PeerID, X.509 Certificate]

Then a TTL (time to live) is associated to the authentication of the client. At the end of the period of validity, the entry is automatically deleted from the list.

Since the default TLS policy implementation uses JXTA's PSE keystore for storing/retrieving certificates, it is necessary that both client and server also

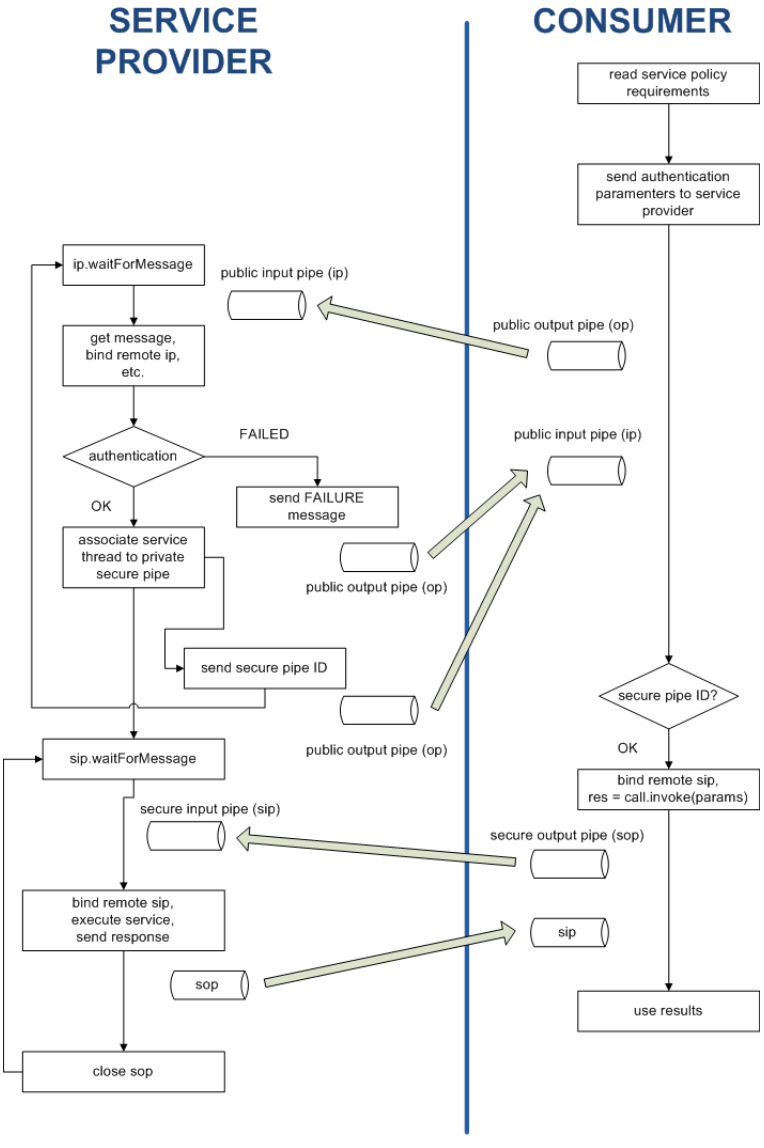

Fig. 3. TLS-based secure invocation model. Concurrent invocations are handled by separated service threads.

authenticate to the PSEMembershipService, which is the default membership service implemented in JXTA. When the client request is received by the server, an output pipe is created for invocation response and validation of authentication parameters; then the X.509 certificate is imported,a new thread is created for secure invocation management and the associated secure pipe is sent to the client for successive invocations.

4.2 DefaultWSSMessage

This is the default message security policy and uses the methods of WSSecurity class to build a security header which is included in all messages and invocation requests. During the authentication phase, the client attaches its X.509 certificate to the header using a `<wsse:BinarySecurityToken>` and digitally signs the request message body with its private key.

The WSS-based policy does not use JXTA PSE keystore, but requires that both client and server generate a couple of private/public keys and use them to create a self-signed X.509 certificate which is stored in their own keystore, whose integrity and privacy are granted by means of a password.

4.3 MIKEYPolicy for Mobile Applications

Since PSE membership classes are not available for JXME version of JXTA platform, we imported and modified them to be able to authenticate peers within a peer group. Moreover, J2ME does not support TLS, so it was impossible to use JXTA secure pipes for service invocation. We implemented a new type of pipe, `JxtaUnicastCrypto`, by which it is possible to cipher message contents, and we used them to define a new security policy, suitable for Connected Device Configuration (CDC) and Personal Profile.

`PipeService` and `PipeServiceImpl` classes in the `net.jxta.pipe` package create and use a SecretKey object containing the cipher algorithm and the corresponding key. The client and the server have to share the same key, so we introduced Multimedia Internet KEYing (MIKEY) [9] protocol to create the key pair and all the required parameters for encryption and decryption operations. Although memory and processing power have dramatically improved for handheld devices, encryption remains a resource-intensive task that requires consideration when designing protocols. MIKEY is a schema for management of cryptographic keys which can be used in real-time and peer to peer applications; it was developed with the intention to minimize latency when exchanging cryptographic keys between small interactive groups that reside in heterogeneous networks. The protocol is defined in RFC 3830 and in JXTA-SOAP project we introduced an implementation with RSA-R algorithm [10]. The standard describes mechanisms for negotiating keys between two or more parties who want to establish a secure channel of communication; to transport and exchange keying material, three methods are supported, Pre-shared secret key (PSK), Public Key encryption (PKE) and Diffie Hellmann (DH) key exchange. Each key-exchange mechanism (PSK, PKE, and Diffie-Hellman) defined in MIKEY is using the same approach of sending and receiving messages, but the message attributes (that is, headers, payloads, and values) differ from method to method.

To create a MIKEY message it is necessary to create an initial MIKEY message starting with the Common Header payload, and then concatenate necessary payloads of the message. As a last step create and concatenate the MAC/signature payload without the MAC/signature field filled in; calculate the MAC/signature over the entire MIKEY message, except the MAC/Signature

field, and add the MAC/signature in the field. The common header payload must be included at the beginning of each MIKEY message (request and response) because it provides necessary information about the Crypto Session with which it is associated. MIKEY defines several payloads to support the three key exchange methods and the corresponding architectural scenarios (that is, peer to peer, simple one to many, many to many, without a centralized control unit). The main characteristic of MIKEY protocol is that it minimizes message exchange; the negotiation of key material should be accomplished in one round trip, as described in figure 4.

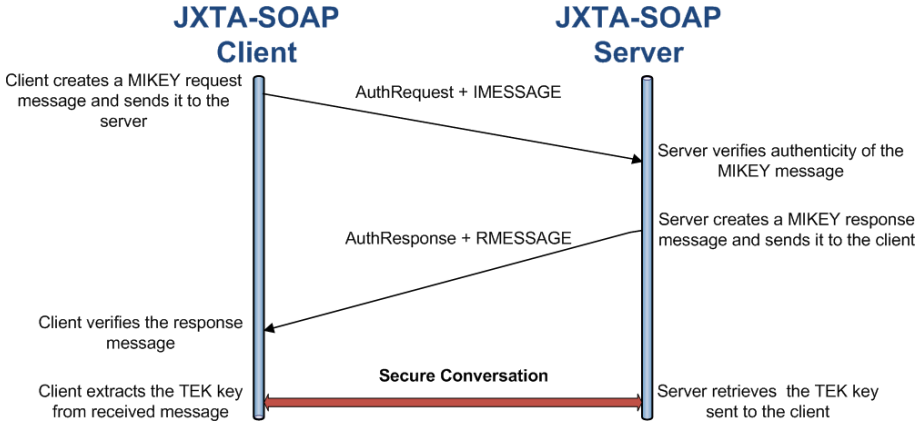

Fig. 4. MIKEY transaction example: the JXTA-SOAP Client is a peer that acts as service consumer, while the JXTA-SOAP Server is a peer that acts as service provider. Of course roles can be exchanged, since every peer can provide and consume services.

5 Conclusions

In this paper we presented the mechanisms that we implemented in the JXTA-SOAP component to support secure service invocation in a peer-to-peer network of service providers and consumers. In particular, we provided insights into the mobile version of JXTA-SOAP, targeting mobile devices with J2ME CDC profile.

Future work will mainly focus on supporting Web Service Security mechanisms also for the J2ME-based version of the component, in order to sign and verify SOAP Messages through the creation of security headers. This will ensure end-to-end security in service invocation when both server and client are resource constrained devices.

References

1. Amoretti, M., Bisi, M., Zanichelli, F., Conte, G.: Enabling Peer-to-Peer Web Service Architectures with JXTA-SOAP. In: IADIS International Conference e-Society 2008, Algarve, Portugal (April 2008)

2. Haustein, S., Seigel, J.: kSoap2 project, http://ksoap2.sourceforge.net
3. Plebani, P.: mAS project, https://sourceforge.net/projects/masproject
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley, Reading (1995)
5. Govoni, D., Soto, J.C.: JXTA and security, JXTA: Java P2P Programming, pp. 251–282 (2002)
6. Zhang, X., Chen, S., Sandhu, R.: Enhancing Data Authenticity and Integrity in P2P Systems. IEEE Internet Computing (November-December 2005)
7. Chen, T.M., Venkataramanan, V.: Dempster-Shafer Theory for Intrusion Detection in Ad Hoc Networks. IEEE Internet Computing (November-December 2005)
8. Wallach Dan, S.: A Survey of Peer-to-Peer Security Issues. In: Okada, M., Pierce, B.C., Scedrov, A., Tokuda, H., Yonezawa, A. (eds.) ISSS 2002. LNCS, vol. 2609, pp. 42–57. Springer, Heidelberg (2003)
9. MIKEY: Multimedia Internet KEYing, http://www.ietf.org/rfc/rfc3830.txt
10. MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY), http://www.ietf.org/rfc/rfc4738.txt
11. Tang, J., Zhang, W., Xiao, W., Tang, D., Song, J.: Self-Organizing Service-Oriented Peer Communities. In: International Conference on Internet and Web Applications and Services (ICIW 2006), Guadeloupe, French Caribbean, February 2006, pp. 99–103 (2006)
12. Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., Hugly, J.-C., Poyoul, E., Yeager, B.: Project JXTA 2.0 Super-Peer Virtual Network, Technical Report, Sun Microsystems (2003)
13. Amoretti, M., Bisi, M., Zanichelli, F., Conte, G.: Enabling Peer-to-Peer Web Service Architectures with JXTA-SOAP. In: IADIS International Conference e-Society 2008, Algarve, Portugal (April 2008)
14. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall PTR, Englewood Cliffs (2004)