Adapting SLP To Ad-Hoc Environment

Janne Pietiäinen, Jussi Saarinen, Pekka Vuorela, and Tommi Mikkonen

Institute of Software Systems, Tampere University of Technology
P.O.Box 553, FIN-33101 Tampere, Finland
{pietiain,saarin24,pvuorela,tjm}@cs.tut.fi

Abstract

Ad-hoc networking, where network structure is created dynamically as nodes enter and leave the network, has recently become an active reseach subject. As majority of existing network protocols has been targeted to be used in an environment, where a static network configuration and the option of using registry repositories is enabled, they need tailoring for ad-hoc networking. In this paper, we discuss how Service Location Protocol (SLP) can be modified for such a dynamic environment starting from the requirements of applications that are to be run, and user's intentions. The adaptations we have implemented include passive service discovery where the amount of network traffic needed for service discovery can be reduced, security related features for improved privacy, gateway function that offers connectivity to external networks, and service discovery proxies that assist in the discovery of services between ad-hoc and fixed networks. The paper also addresses implementation of these features.

1 Introduction

While a majority of application level protocols that are readily available for networking are based on an assumption that a fixed structure exists, it is often possible to augment the protocols with extensions that enable ad-hoc networking. This approach is further supported with the option to use the same protocol in both ad-hoc and infrastructure assisted network, e.g. the Internet [8]. Furthermore, application requirements and the convenience of the user should be the driving force for new features.

One crucial issue in ad-hoc networking is the ability to locate services in the network. In this paper, we will use Service Location Protocol (SLP) [2] as a vehicle for studying the modifications needed for ad-hoc networking in

a fashion that is adequate for applications and convenient for the user. The extensions we have implemented to SLP for supporting operation in ad-hoc network include the following:

- Passive service discovery where network traffic can be reduced;
- Security related features for improved privacy;
- Gateway function that offers connectivity to external networks;
- Service discovery proxy function that assists in the discovery of services between ad-hoc and fixed networks.

The run-time environment of our implementation consists of laptop computers with built-in WLAN connectivity and 2.6 Linux kernels.

The rest of this paper discusses these issues as follows. Section 2 introduces SLP and its most relevant features from the viewpoint of ad-hoc networking. Sections 3, 4, 5, and 6 discuss the above improvements we have implemented using OpenSLP [9] as the baseline system. Then, Section 7 concludes the paper.

2 Service Location Protocol v2

The goal of Service Location Protocol (SLP) version 2 is to enable effortless autoconfiguration in fixed networks. In addition to discovering services of a certain type SLP also allows discovery based on service attributes. SLP also provides means for service browsing: a user may discover all available service types, search for attributes associated with a certain service type, and also issue a request for attributes of a single service. The use of DHCP and multicast in the initialization of SLP framework enable it to scale from a single local area network to an enterprise network. SLP supports also administrative grouping of services with so-called scopes.

SLP has three types of entities: User Agents (UA), Service Agents (SA), and Directory Agents (DA). UA represents a client that searches for services, SA represents a service provider, and DA operates as a centralized service repository. The most general operations and messages associated with them are illustrated in Figure 1.

UAs issue three multi- or broadcast Service Request (`SrvRqst`) messages during each discovery operation, to which SAs that have a matching service in their local databases respond with unicast Service Reply (`SrvRply`) message. If DAs are present SAs must register their services to them with unicast Service Registration (`SrvReg`) messages that DAs respond to with unicast Service Acknowledgement (`SrvAck`) messages. UAs are required to request services from them with unicast `SrvRqst` messages that DAs respond to with unicast `SrvRply` messages. UAs and SAs may actively search for DAs by issuing multi- or broadcast `SrvRqst` messages. DAs respond to these messages

with unicast DA Advertisement (`DAAdvert`) messages. DAs may also periodically send multi- or broadcast `DAAdvert` messages to enable UAs and SAs to passively discover them.

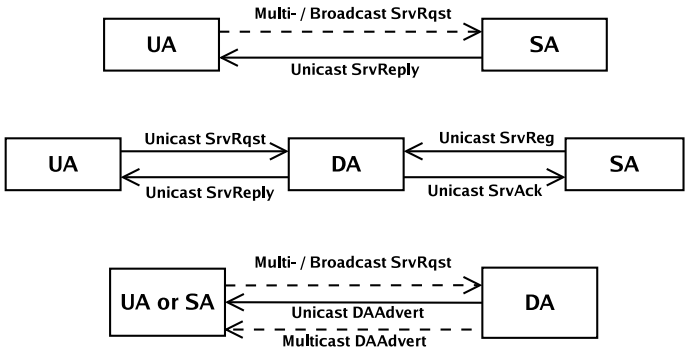

Fig. 1. SLP agents and most common messages in the protocol.

SLP has a security scheme that enables UAs to verify authenticity of SAs. It is aimed at preventing forged service information from being propagated in the framework. The scheme relies on signatures generated with public key cryptography. They are carried inside authentication blocks along with Security Parameter Index (SPI) strings that indicate which public key can be used to verify the signature. The signature is generated by forming a hash from relevant fields of a message with SHA-1 [3] and then encrypting it with the private DSA [4] key of the sender. These fields depend on the message type. The authentication block structure is illustrated in Figure 2.

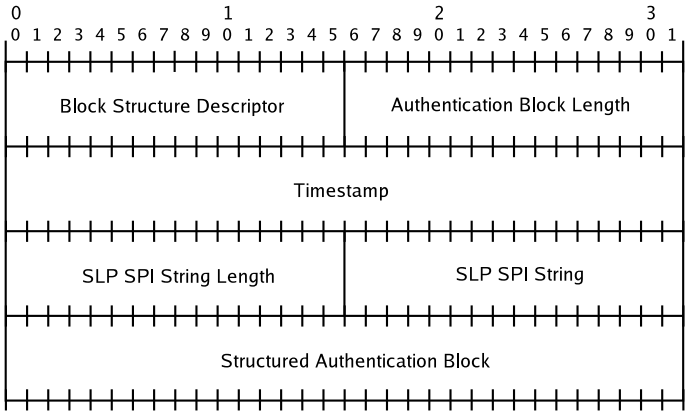

Fig. 2. The SLP Authentication Block.

We chose version 1.0.11 of OpenSLP C language implementation as the basis for our Service Discovery (SD) module. OpenSLP is an open source implementation of SLP version 2. It is divided to two main elements: libSLP and SLPD. LibSLP implements Service Location API [6] and provides UA functionality. SLPD is a daemon that implements both SA and DA functionalities. The SD module consists of these elements and of a general SD API that was developed to enable changing of the actual implementation protocol. Due to the highly dynamic target environment, OpenSLP was decentralized by modifying it to use broadcasting in service discovery instead of DAs. This means that each node can advertise its locally registered services by itself.

3 Passive Discovery of Services

In a link-local ad-hoc network consisting of mobile devices, it is important to minimize network transmissions and thus conserve power. Since the set of available services in an ad-hoc network is constantly changing as nodes enter and leave, a UA must repeat service discovery periodically to maintain an up to date list of available services. As we discarded the use of SLP DAs as such in the ad-hoc network, UAs can only discover new services by performing a service discovery and waiting for replies from SAs. In a limited bandwidth network with a large number of UAs, this may result in a significant constant network traffic, since each SLP service request consists of three broadcast messages from the UA and possibly a unicast service reply message from an SA.

Passive Discovery (PD) is an extension to SLP that was designed to ease the problem discussed above. It allows SAs to broadcast advertisements of their services so that UAs can passively accumulate a list of services they are interested in. A typical scenario for using PD would consist of a relatively large number of UA nodes interested in a service offered by a relatively few SA nodes. This new discovery method is intended to supplement SLP's normal "active" discovery. A UA can start discovering services by issuing an active discovery to get a snapshot of currently available services, and then use passive discovery to stay informed when new services become available and old services disappear. Unlike the active discovery in SLP, passive discovery is not a blocking operation, so the application is free to perform other tasks while passive discovery is running.

The implementation of passive discovery is divided between the SD module and the SLP library (libSLP) which both reside within a PD enabled application, and the SLP daemon (SLPD). LibSLP relays service registrations and deregistrations from the application and SD module to SLPD, which in turn is responsible for sending the actual outgoing service advertisements, receiving incoming service advertisements, and relaying the advertisements for relevant services back to the application via libSLP and SD module. LibSLP and local SLPD communicate through a TCP connection on the loopback interface.

LibSLP sends to SLPD special control messages encapsulated in a custom SLP message type (`CtrlMsg`) using SLPD's existing messaging infrastructure. SLPD communicates back to libSLP with similar control messages, but without the SLP encapsulation. These communications are all initiated by libSLP. One of our goals was to minimize the necessary changes to OpenSLP during the implementation process and furthermore, PD works very differently from its design, therefore libSLP was not directly modified. Parts of PD that are related to libSLP are implemented as a parallel system and compiled to the same library as the original OpenSLP implementation. SLPD was modified by adding a custom SLP message (`SrvAdvert`) for service advertisements and a new subsystem, which manages service listener registrations from libSLP, processes the incoming service advertisements, and sends the outgoing service advertisements as broadcast messages. The subsystem also contains two databases, one for outgoing service advertisements, i.e. services registered by local libSLP instances, and another for the service types that local libSLP instances are interested in so that received advertisements for matching services can be relayed to them. A single process can initiate multiple simultaneous passive discoveries, and multiple processes are able to utilize passive discovery within one device. Despite the modifications, PD enhanced OpenSLP should be able to interwork with other nodes running rfc2608 compliant SLPv2 implementations.

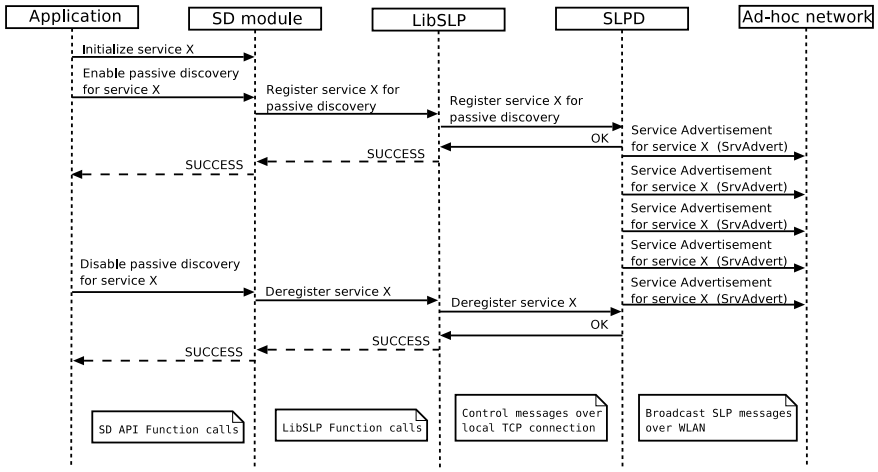

Fig. 3. Registering and deregistering services with Passive Discovery.

When a PD enabled application proceeds to register a service for passive discovery (Figure 3), it must first initialize a data structure representation of the service and then advice the SD module to enable passive discovery for the service. SD module relays the registration to libSLP which prepares a registration message, connects to SLPD, and transmits the message. It then waits

for a reply message containing an error code from SLPD, which is returned to SD module and finally to the application. SLPD processes the service registration message, and, if all is in order, it stores the registration in a database for outgoing service advertisements. Services in the database are advertised at predefined intervals until either their lifetime expires or they are deregistered.

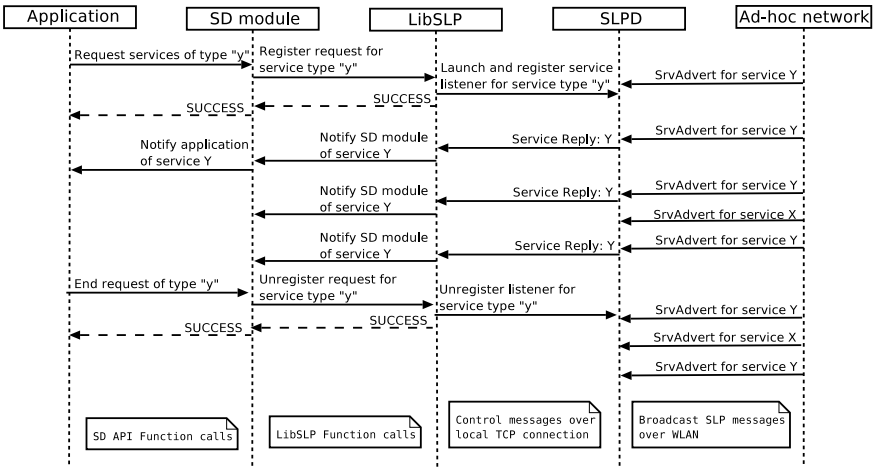

Fig. 4. Service discovery with Passive Discovery.

When a UA initiates passive service discovery (Figure 4), libSLP launches a service listener thread, which opens a TCP connection to SLPD, registers the requested service type for filtering incoming service advertisements and then remains waiting for results. Upon receiving the service request, SLPD adds the information to a request database. When SLPD receives incoming service advertisements, it compares them against active requests and relays the matching advertisements to interested parties service listeners, which in turn forwards them to applications' SD modules. When SD module receives a result to a request, it checks an internal list of known services to see if it has already recently received an advertisement for the service. If it has not, then the service information is passed on to SD module, and the service is added to the list of known services. Subsequent advertisements for known services are disregarded since the application is already aware of their existence. If no advertisement for a known service has been received for a certain period of time, SD module removes its information from the list of known services and notifies the application that the service disappeared.

In our tests PD worked well. However, due to time and resource constraints, the implementation contains some restrictions. Actively and passively advertised services are currently completely separate, i.e. two services can have an identical name but different attributes, lifetime etc. if one is made available for active discovery and the other for passive discovery. In a typical use sce-

nario for PD there would be a large number of UA nodes looking for a service, such as a printer or a gateway, advertised by a relatively few SA nodes. Let us assume that we have an ad-hoc network with 3 SA nodes offering a gateway service, and 50 UA nodes interested in such services. If the nodes would try to discover the services using active model once every 30 seconds, the traditional SLP solution would consist of each UA node broadcasting an SLP service request message three times and SAs sending a unicast service reply message to each UA which amounts to 300 sent messages per discovery cycle. In contrast the same scenario with passive discovery, SA nodes advertising every 30 seconds, results in 3 sent messages per discovery cycle. If we assume that the sent messages are approximately 110 bytes in size per message, the passive model generates only 1% of the messages and data to be sent in this scenario in comparison to active model.

4 Secure Service Discovery

When compared to traditional fixed networks, an ad-hoc network imposes new security requirements for service discovery protocols. It is highly dynamic, more open and unsecure. Therefore protocols used in such environment need to be augmented with effective security features. We enhanced OpenSLP by enabling two security levels, "Authentication" and "Confidentiality". This system allows the whole scheme adapt to varying resource constraints of mobile nodes.

The implementation uses Authentication and Authorization (AA) module to store access control rules, certificates, and related keys. The AA module is together with cryptographic helper module that utilizes OpenSSL library 0.9.7d or later [10], used to perform all required cryptographic operations. It is described more thoroughly in [7]. To establish connection between each service and required security properties, we use abstract part of the SLP service type as a service ID. Due to this design choice, service browsing capabilities of SLP were disabled.

Authentication level enables two-way role based access control, authentication, and authorization between UA and SA. Furthermore, it protects integrity of most fields in the SLP messages and uses a logical timestamping system instead of the real-time system used by the original SLPv2 security scheme described in Section 2. This level was enabled for both active and passive discovery by adding SLP Extension blocks that carry modified SLP Authentication blocks to the end of `SrvRqst`, `SrvRply`, `AttrRqst`, `AttrRply` and `SrvAdvert` messages. Each modified SLP Authentication block contains sender's role dependent user ID. It is used together with the ID of the requested service by the receiver to determine which keys and access control rules should be used. The signature attached to the modified block is generated the same way as the signature in the original SLPv2 security scheme but it covers all fields of the message excluding the message length. The logical

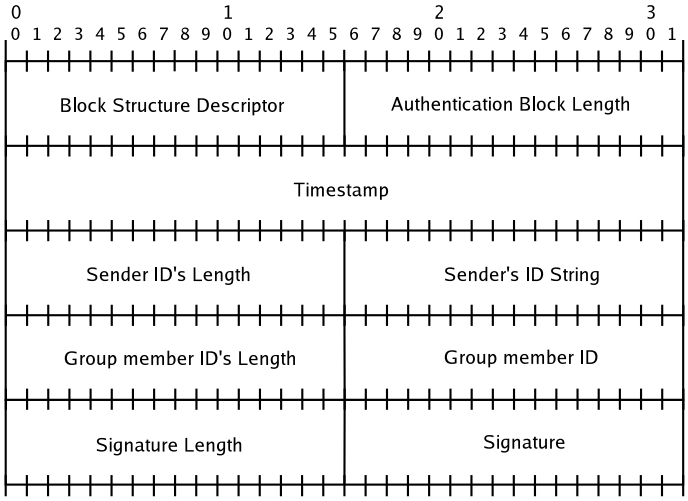

Fig. 5. The Modified SLP Authentication Block.

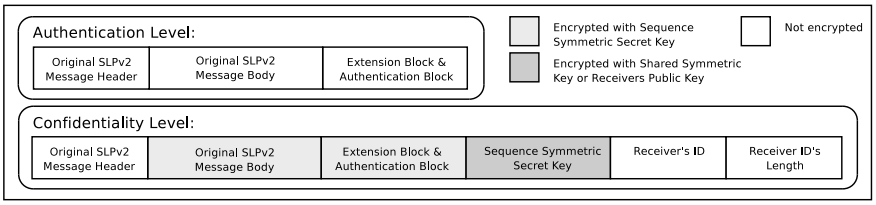

Fig. 6. The Message Structures on Authentication and Confidentiality Security Levels.

timestamping system uses positive integer valued timestamps that are increased each time a message is sent. All user ID and timestamp value pairs of each sent and received message are stored in a database for a limited lifetime. Since the timestamp and the sender's user ID are both signed this method enables nodes to detect replayed messages without synchronized clocks. They just need to compare user IDs and timestamps of received messages to the ones stored previously in its database. The limited lifetimes of timestamps also enable them to resolve situations in which one of them resets and thus loses its timestamp database. The structure of the modified Authentication block is illustrated in Figure 5.

Confidentiality level adds a partial message encryption on top of the features of the Authentication level. It was enabled only for active discovery and is therefore used with the same messages as the Authentication level excluding `SrvAdverts`. Message body and the Extension block are encrypted with a symmetric AES [5] key that is different for each message exchange sequence

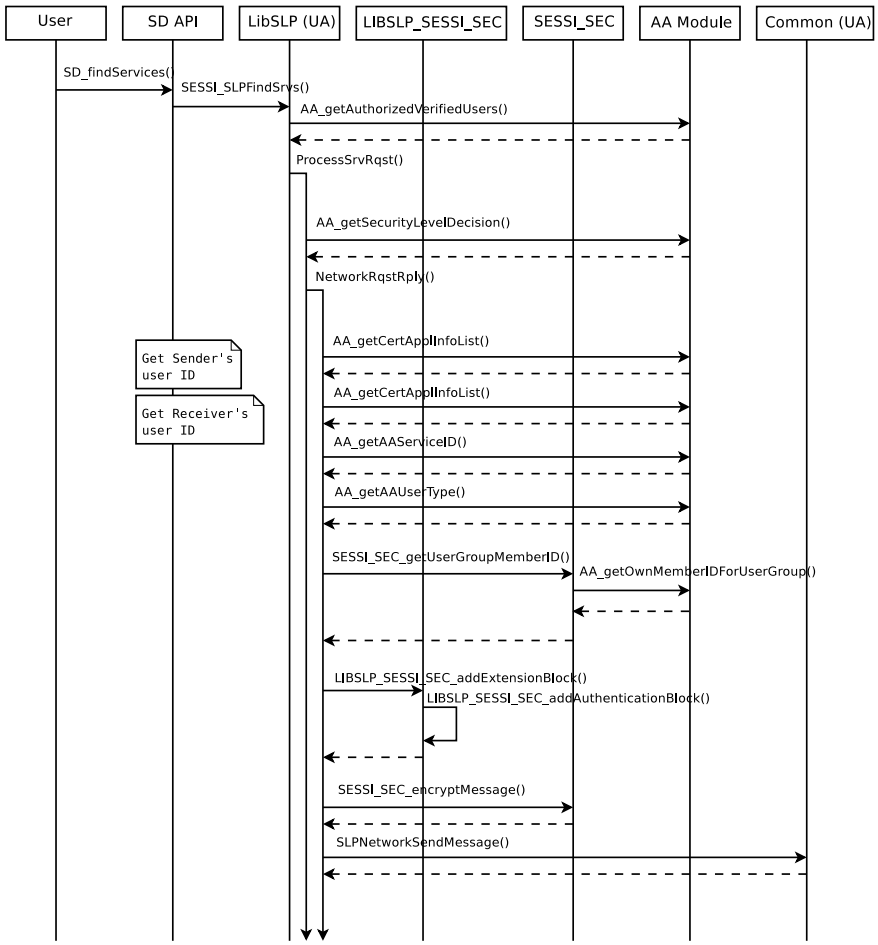

Fig. 7. The Secure Service Discovery Operation on UA Side Part 1.

consisting of one Request-Reply message pair. The symmetric key is then encrypted with the intended receiver's asymmetric RSA [11] public key. This lessens the computational load caused by the encryption process. Receiver's user ID is finally added to the end of the message accompanied by its length enabling receivers to identify messages intended for them and thus avoid unnecessary decryption attempts. The SLP header is not encrypted because it contains message length information that is required in message transmission over TCP connections. Furthermore information in it was not regarded too sensitive. The message structure is illustrated in Figure 6.

A detailed description of the Confidentiality level functionality on UA is described in Figures 7, 8, and 9. The first diagram shows how user initiates service discovery by calling function `SD_findServices`. Then users

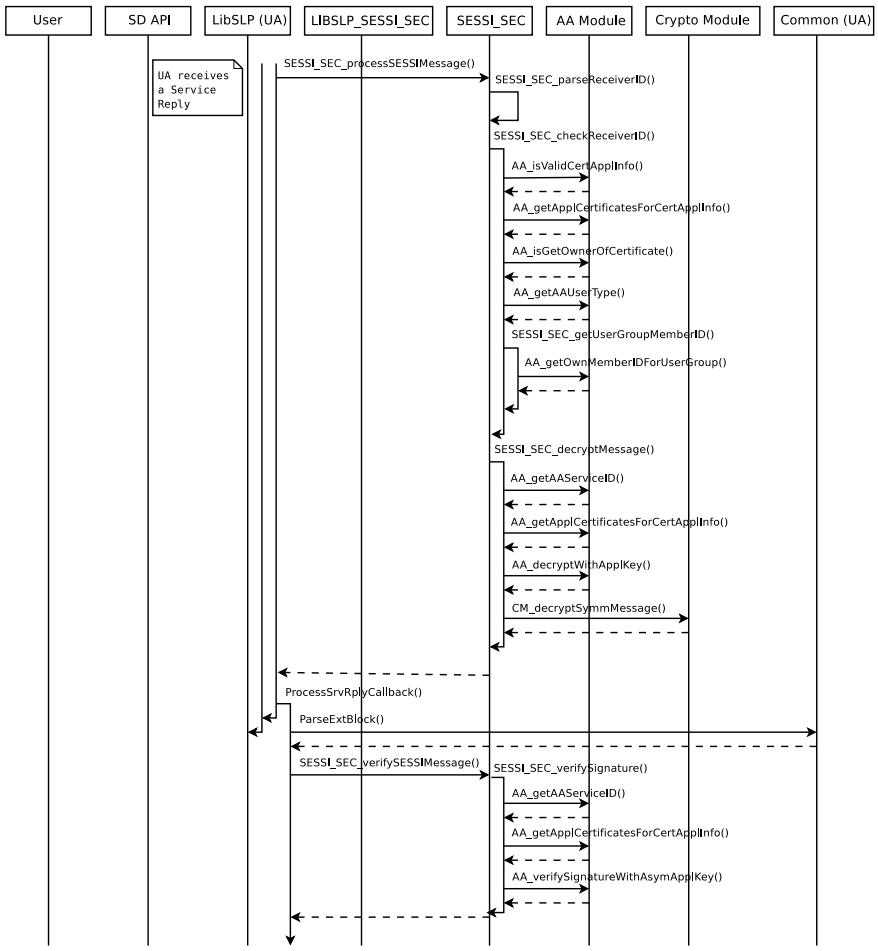

Fig. 8. The Secure Service Discovery Operation on UA Side Part 2.

who are authorized and verified are retrieved from AA module by calling `AA_getAuthorizedVerifiedUsers` and for each of them a security level is decided by calling for `AA_getSecurityLevelDecision`. Finally the Extension block is added to the message, it is then encrypted and receiver's user ID is attached to the end before transmission. The second diagram shows how `SESSI_SEC_processSESSIMessage` is used to determine the message security level and decrypt it. After this the signature of the message is verified with `SESSI_SEC_verifySESSIMessage`. The final diagram shows how the message timestamp is verified with `LIBSLP_SESSI_SEC_verifyTimestamp`, the sender's authorization to provide the service is checked with `SESSI_SEC_isAuthorizedForService`, and

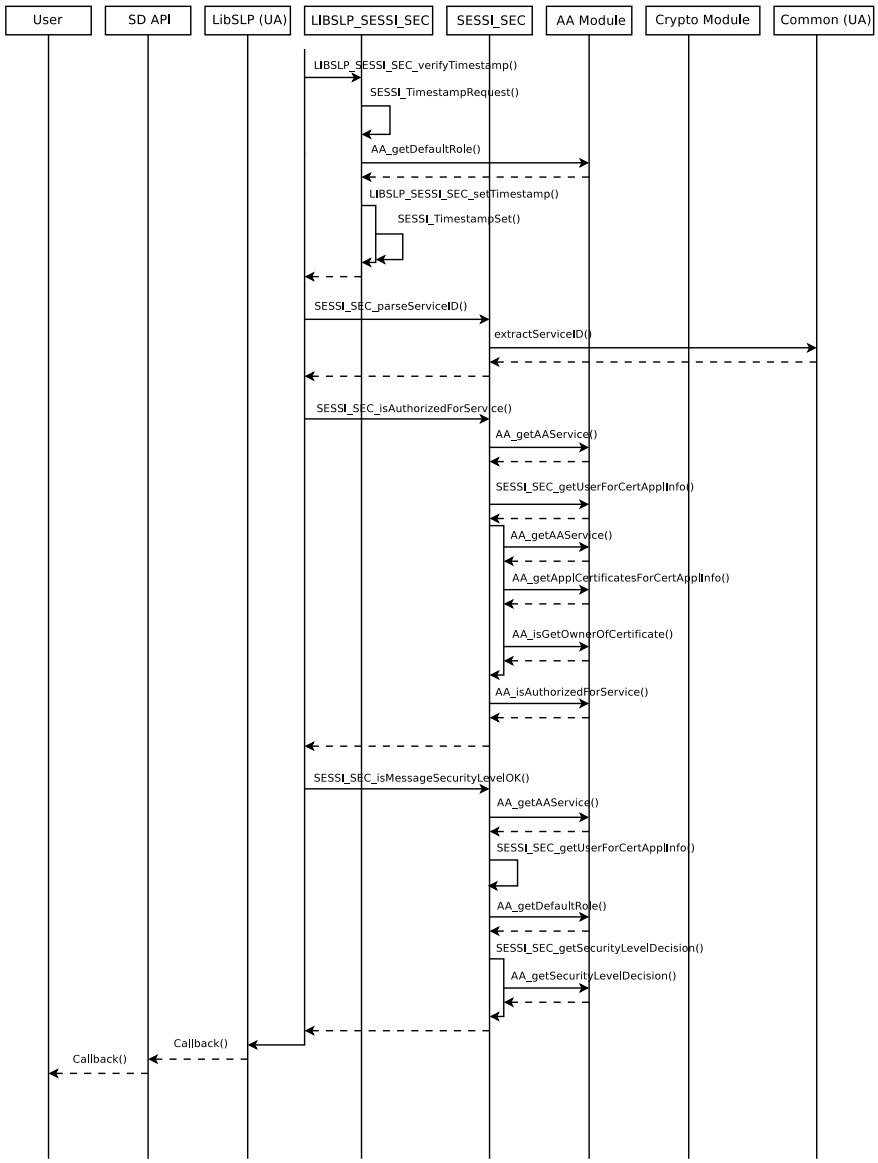

Fig. 9. The Secure Service Discovery Operation on UA Side Part 3.

`SESSI_SEC_isMessageSecurityLevelOK` is finally used to verify that the received message's security level is sufficient.

A detailed description of the Confidentiality level functionality on SA is described in a similar manner in Figures 10, 11, and 12. The first diagram shows what happens when SA receives a SLP message.

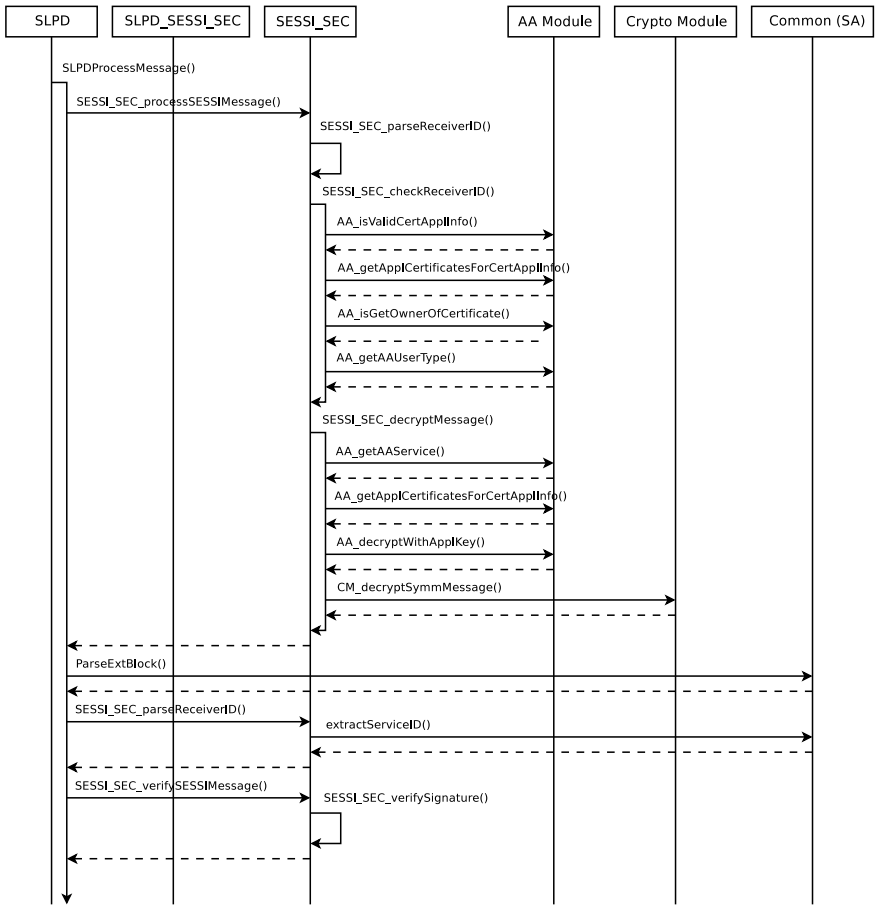

Fig. 10. The Secure Service Discovery Operation on SA Side Part 1.

Function `SESSI_SEC_processSESSIMessage` is used in the same purpose as in UA and the signature of the message is verified with `SESSI_SEC_verifySESSIMessage`. The second diagram presents how the message's timestamp is verified with `SLPD_SESSI_SEC_verifyTimestamp`, the message sender's authorization to access the requested service is checked with `SESSI_SEC_isAuthorizedForService` and the message security level is again verified with `SESSI_SEC_isMessageSecurityLevelOK`. After these steps the SA will check if the service is found in its database. The final diagram shows how `SLPD_SESSI_SEC_processOutgoingSESSIMessage` is called to perform necessary actions for the SA's reply message before it is transmitted. It adds the Extension block to the message, encrypts it, and adds the receiver's user ID to the end of the message.

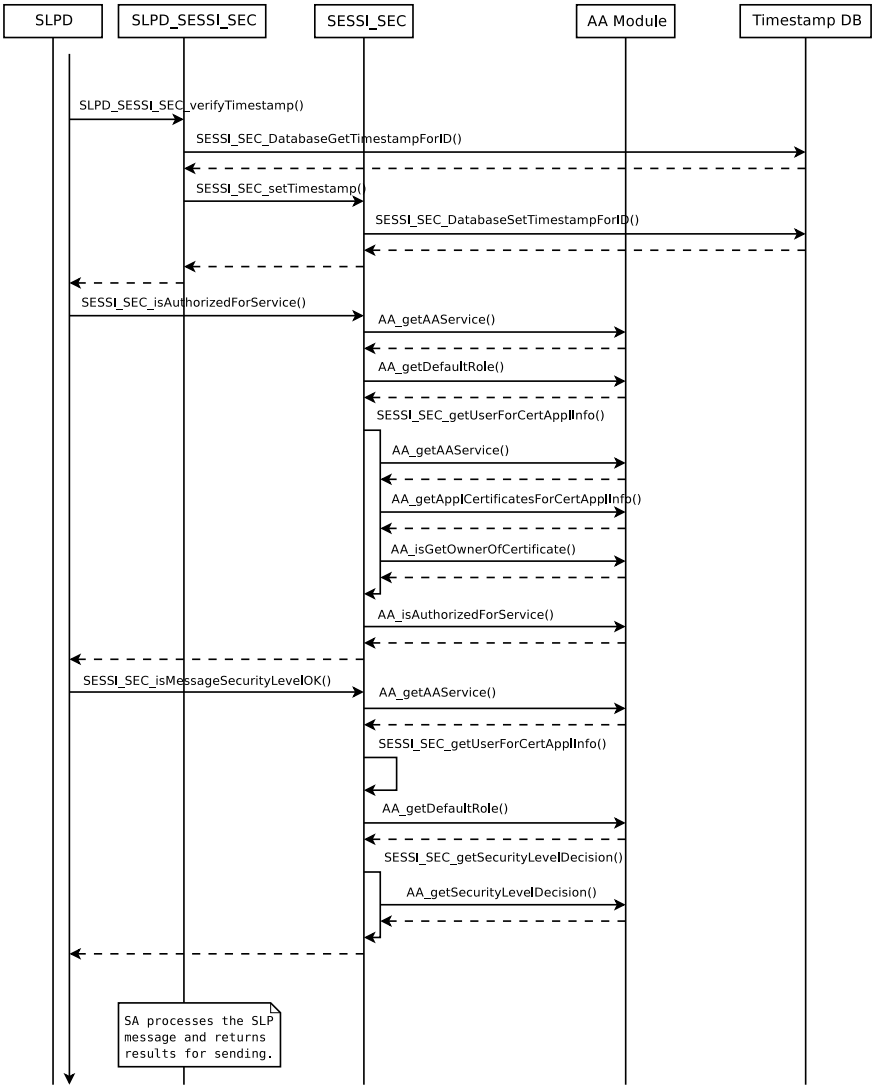

Fig. 11. The Secure Service Discovery Operation on SA Side Part 2.

Our implementation of the security extension achieves the goals we set and has been identified to cooperate with other components of our framework i.e. Passive Discovery and Gateway. It should also maintain interoperability with nodes running rfc2608 compatible SLPv2 implementations when no security features are used. On the downside the cryptographic operations can be fairly heavy for mobile devices with limited resources. However, since their capabilities are constantly expanding, this will most likely cease to be an issue in the near future. Our security scheme relies on predistributed certificates and

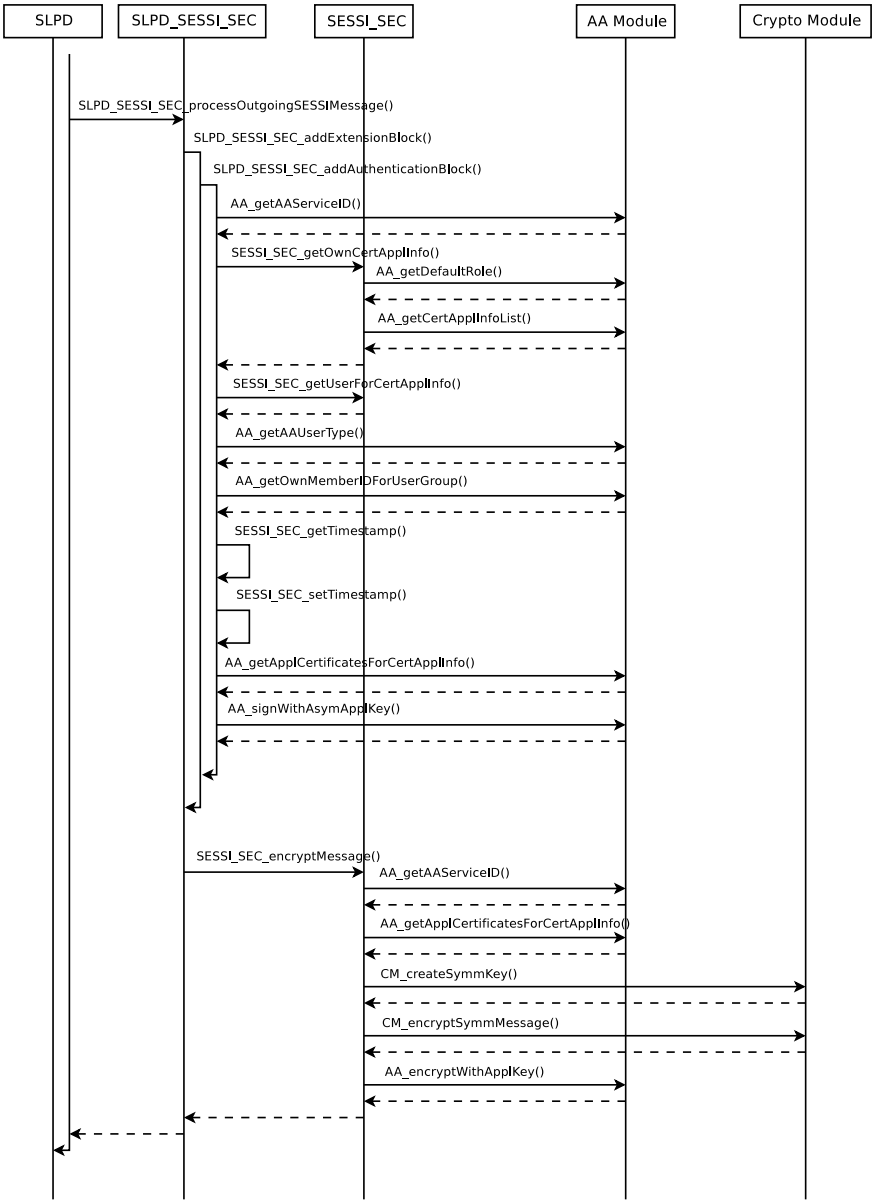

Fig. 12. The Secure Service Discovery Operation on SA Side Part 3.

keys due to the fact that use of distribution servers in ad-hoc network is not feasible. This characteristic imposes some flexibility constraints on our system that could be averted by relying on servers located in the fixed network. The approximate average message sizes in a typical ad-hoc network with 15 nodes

are the following: 110 bytes when no security features are used, 300 bytes on Authentication level, and 480 bytes on Confidentiality level. The SHA-1 signature is 64 bytes long and represents almost half of the overhead on Authentication level. The rest of the overhead on that level is due to the Extension block and Authentication block structures. The additional overhead generated on the Confidentiality level is due to the AES encrypted data, the RSA encrypted 16 bytes long AES key, and the 10 bytes long receiver's ID that is added to the end with two byte length field.

Different approaches to create a secure service discovery infrastructure have been presented in several papers. Many of them rely on servers located in the fixed network which is an infeasible option in our target environment. Czerwinski et al. [1] present a scheme that relies on servers that form a dynamic hierarchy. Use of external servers to distribute certificates and provide access control information also suggests that all ad-hoc nodes should always have connection to the fixed network. Zhu et al. [12] have developed a scheme that supports also privacy protection and location dependent service discovery. Their solution uses proxies to support the mobile nodes which therefore must have a stable connection to the fixed network.

5 Gateway

Ad-hoc users may want to communicate with users in the fixed network or use the services available there. To achieve this goal a network gateway is needed. Due to the dynamic nature of ad-hoc environment, the gateway should not be a static entity, but rather any node willing to provide the service to others should be able to act as one. Therefore the gateway should itself be a service, discoverable in the ad-hoc network, and the client nodes should be able to easily start using it. The gateway should also be able to enforce access control on users. Furthermore, the nodes that have not been authorized for external connectivity should not be able steal it from authorized nodes. These requirements were addressed in our design of the gateway service.

Our implementation consists of two main components. Gateway Manager (GM) resides in the gateway node, and is responsible for initializing the gateway service along with negotiating and managing incoming client connections. Gateway Client (GC) resides in an ad-hoc node that wants to use the gateway, and takes care of automating the steps necessary for locating a suitable gateway, negotiating the connection parameters and configuring the node for external connectivity via the gateway. An example of an ad-hoc network with three gateways, one hosted by a mobile operator and the other two by nodes in the ad-hoc network, and client nodes is presented in Figure 13.

The gateway is highly customizable. Its modular design makes it enabled to launch and manage other related services, such as various proxy servers, DNS server etc. The method in which the connection between GM and GCs is established is defined as a gateway mode. GM can offer many different modes

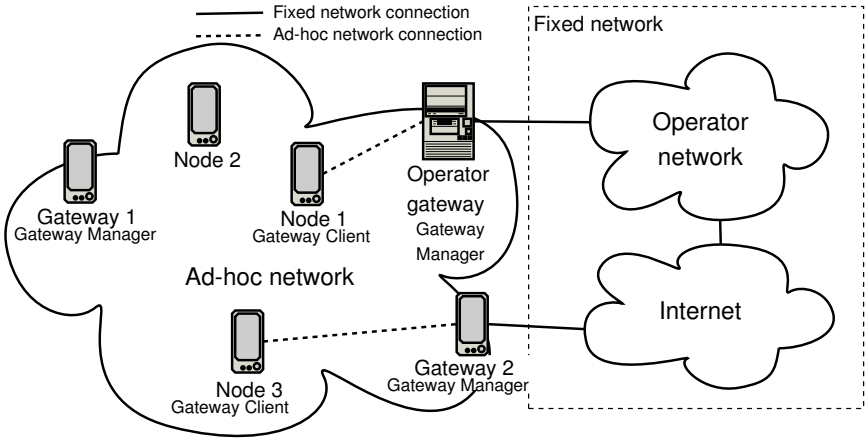

Fig. 13. An example of the gateway service.

which are then listed as attributes of the registered service. GC chooses one mode that it also supports, and begins the connection negotiations with GM. We implemented two modes. The first mode, named "open", is simply an open gateway without any access control. Any ad-hoc node can be configured to use the gateway operating in this mode. The second mode is named "secure". It implements a secure, encrypted tunnel from GC to GM, and requires users to be authenticated and authorized. All security features are implemented with the AA module. Gateways can be tagged with an operator string, which can then be used to target the gateway discovery to those that belong to a given operator. Commercial gateway services could authenticate and authorize customers using the security features in AA module.

Both GM and GC are command line tools, implemented as C language programs with shell script frontends. They both require an SLPD with our modifications to be running on the node. The gateway service worked well in our tests and supported other components, such as SD Proxy. It is also a good example of a service implemented with our platform.

6 Service Discovery Proxy

Connectivity established with our gateway between ad-hoc and fixed infrastructure network, allows using services from the other network. This creates a need to locate the services and therefore support from the service discovery service is necessary. To enable searches across the networks, we introduced a new entity called SD Proxy to the SLP network.

The first step in the implementation of searches between the networks was separating the services using SLP scopes. One scope was specified for the

services that are globally available and another one for the services that are available to the ad-hoc network.

The SD Proxy was based on the Directory Agent of the basic SLP, and it resides on the gateway node. Ad-hoc nodes willing to use or offer services to the fixed network can set the gateway node to be used as a DA for the global SLP scope. DA Advertisement messages were not used because the information on SD Proxy availability is obtained in the search for gateways. Further, rogue nodes could advertise DA service for the ad-hoc scope and unwary nodes would direct their ad-hoc searches to them. When a DA for the global scope is set to be used by the UAs, queries using that scope will be unicast to the DA, while queries using ad-hoc scope are still broad- or multicast to the ad-hoc network. On the fixed network side it was assumed that the nodes willing to access ad-hoc network or offer services to it are configured with the address of the SD Proxy.

The use of an SD Proxy enables service registrations and searches to be made from both ad-hoc and fixed networks. However, the address space of the ad-hoc network is commonly link-local and non-routable. This means that services cannot be offered to the fixed network using the ad-hoc addresses. There are two solutions for this problem: giving nodes in the ad-hoc network additional global addresses which can be used to register services, or using Network Address Translation (NAT) for the services. Both of these solutions can be used simultaneously. In our implementation global addresses can be acquired from the gateway service when forming the connection to the fixed network, but the focus was on the NAT support.

To enable NAT for the ad-hoc services the SD Proxy was modified to create port forwards. This requires that the NATted service URLs include both address and port number, i.e. `<IP_address>:<port>`. The port number needs to be explicit so the SD Proxy knows where to forward the incoming traffic. The services made available to the fixed network have their original ad-hoc address replaced with the SD Proxy's fixed network address and the forwarded port number.

Separate scopes and the SD Proxy provided an elegant solution for service discovery support in heterogenous environment. Also compatibility with basic SLP was preserved from the fixed network side. A known problem with the current network setup is the need to configure fixed network nodes to use the SD Proxy. Also, the NAT support has limitations, e.g. services using multiple ports are difficult to advertise.

7 Conclusions

Many research papers on ad-hoc networking have contributed to low-level problems. However, the effect of ad-hoc environment on application level aspects and the convenience to user has been studied less. Further motivation

for our work is on the option to use the same protocols in ad-hoc and infrastructure organized networks, but with certain new functions that are of crucial importance for ad-hoc networking. In this paper, we have addressed service discovery from this viewpoint.

As the sample service discovery protocol, we used SLP. Furthermore, we started with an assumption that link-local communication will be used to focus the work to the actual service discovery. The extensions we have identified are:

- Passive service discovery in order to avoid excessive communication when a group of nodes look for services;
- Security considerations restrict the visibility and availability of services;
- Connectivity to an external network via a node in the ad-hoc network;
- Service discovery proxy function that can be used for locating services in the ad-hoc network from some other network, and vice versa.

Furthermore, we discussed how we have implemented these features in an open source SLP implementation.

Like in any research, there are a number of topics that could be further studied. One option is how to enable the use of several nodes as a gateway at the same time or in turns for improved bandwidth. This would then allow the users to offer connectivity in turns to e.g. share the costs of the external connectivity. Also the SD Proxy could be enhanced by making it more proxy-like with a static DA in the fixed network. This would allow dynamic fixed network addresses for SD Proxies and ease having several SD Proxies on the network. No changes would be needed to the nodes in the ad-hoc network. Another direction for future study is to widen the scope of the approach to e.g. multihop networks.

References

1. S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An architecture for a secure service discovery service. In *Mobicom '99*, Seattle, Washington, USA, August 1999. ACM.
2. Guttman E., Perkins C., Veizades J., and Day M. Service location protocol, version 2. Technical report, The Internet Engineering Task Force, June 1999.
3. D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, The Internet Society, September 2001.
4. FIPS. Digital Signature Standard (DSS). Standard 186, National Institute of Standards and Technology (NIST), May 1994.
5. FIPS. Advanced Encryption Standard (AES). Standard 197, National Institute of Standards and Technology (NIST), November 2001.
6. Kempf J. and Guttman E. An API for service location. RFC 2614, The Internet Engineering Task Force, June 1999.
7. L. Källström, J. Saarinen, and S. Liimatainen. Secure service discovery protocol implementation for wireless ad-hoc networks. In *1st International Wireless Summit, Aalborg, 17-22 September*, 2005.

8. S. Leggio, S. Liimatainen, J. Manner, T. Mikkonen, J. Saarinen, and A. Ylä-Jääski. Towards service interworking among ad-hoc networks and the internet. In *14th IST Mobile and Wireless Communications Summit, Dresden, 19-23 June*, 2005.
9. OpenSLP Project Group website. At `http://www.openslp.org`, April 2004.
10. The OpenSSL project. OpenSSL: The open source toolkit for SSL/TLS.
11. R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystem. *Communications of the ACM*, 21(2):120–126, 1978.
12. F. Zhu, M. Mutka, and L. Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 235–242, March 2003.