

[MS-GRVSPMR]: Management Server to Relay Server Groove SOAP Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: <http://www.microsoft.com/interop/osp>) or the Community Promise (available here: <http://www.microsoft.com/interop/cp/default.mspx>). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplq@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial Availability
06/27/2008	1.0	Major	Revised and edited the technical content
12/12/2008	1.01	Editorial	Revised and edited the technical content
07/13/2009	1.02	Major	Changes made for template compliance
08/28/2009	1.03	Editorial	Revised and edited the technical content
11/06/2009	1.04	Editorial	Revised and edited the technical content
02/19/2010	2.0	Minor	Updated the technical content
03/31/2010	2.01	Editorial	Revised and edited the technical content
04/30/2010	2.02	Editorial	Revised and edited the technical content
06/07/2010	2.03	Editorial	Revised and edited the technical content
06/29/2010	2.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	2.05	Minor	Clarified the meaning of the technical content.
09/27/2010	2.05	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	2.05	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	2.05	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	2.05	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	2.05	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1	Introduction	8
1.1	Glossary	8
1.2	References.....	9
1.2.1	Normative References.....	9
1.2.2	Informative References	10
1.3	Protocol Overview (Synopsis)	10
1.3.1	Service Initialization	10
1.3.1.1	Server Bootstrapping	10
1.3.1.2	Client Bootstrapping.....	11
1.3.1.3	Out-Of-Band Certificate Exchange.....	11
1.3.2	Service Interfaces	11
1.3.2.1	Client/Server Registration.....	11
1.3.2.2	Changing Server Settings	11
1.3.2.3	Changing Server States.....	11
1.3.2.4	Adding Users.....	11
1.3.2.5	Enabling/Disabling User Relay Services	12
1.3.2.6	Purging User Data	12
1.3.3	Protocol Security	12
1.4	Relationship to Other Protocols.....	12
1.5	Prerequisites/Preconditions	13
1.6	Applicability Statement.....	13
1.7	Versioning and Capability Negotiation.....	13
1.8	Vendor-Extensible Fields.....	13
1.9	Standards Assignments	13
2	Messages.....	14
2.1	Transport.....	14
2.2	Message Syntax	14
2.2.1	Namespaces	14
2.2.2	Common Schema	14
2.2.2.1	Common Elements.....	17
2.2.2.1.1	Fault Element	18
2.2.2.1.2	fragment Element	18
2.2.2.1.3	Payload Element	19
2.2.2.1.4	Version Element.....	19
2.2.2.2	Common Types	20
2.2.2.2.1	BooleanType.....	20
2.2.2.2.2	NumericType	20
2.2.2.2.3	PayloadType.....	20
2.2.2.2.4	ServiceFaultResponseType	21
2.2.2.2.5	ServiceRequestType	22
2.2.2.2.6	ServiceResponseType	22
2.2.2.3	Common Attributes.....	22
2.2.2.3.1	data	23
2.2.2.3.2	epoch	23
2.2.2.3.3	rowCount	23
2.2.2.3.4	userId.....	23
2.2.2.3.5	EC.....	23
2.2.2.3.6	IV	23
2.2.2.3.7	MAC	24

2.2.3	Message Definitions	24
2.2.3.1	accountModify Request Message	24
2.2.3.1.1	accountModify Request Payload	24
2.2.3.2	accountModify Response Message	25
2.2.3.2.1	accountModify Response Payload	25
2.2.3.3	Fault Response Message	26
2.2.3.4	Registration Request Message	26
2.2.3.4.1	Registration Request Payload	26
2.2.3.5	Registration Response Message	28
2.2.3.5.1	Registration Response Payload	28
2.2.3.6	RelayDefault Request Message	29
2.2.3.6.1	RelayDefault Request Payload	29
2.2.3.7	RelayDefault Response Message	30
2.2.3.7.1	RelayDefault Response Payload	30
2.2.3.8	RelayQuiescent Request Message	31
2.2.3.8.1	RelayQuiescent Request Payload	31
2.2.3.9	RelayQuiescent Response Message	31
2.2.3.9.1	RelayQuiescent Response Payload	31
2.2.3.10	userAdd Request Message	32
2.2.3.10.1	userAdd Request Payload	32
2.2.3.11	userAdd Response Message	33
2.2.3.11.1	userAdd Response Payload	33
2.2.3.12	userPurge Request Message	33
2.2.3.12.1	userPurge Request Payload	33
2.2.3.13	userPurge Response Message	34
2.2.3.13.1	userPurge Response Payload	34
3	Protocol Details	36
3.1	Common Details	36
3.1.1	Common Message Templates	36
3.1.1.1	Request Message Template	36
3.1.1.2	Response Message Templates	36
3.1.1.3	Payload Data Template	37
3.1.2	Common Security Processing Rules	38
3.1.2.1	MARC4 (Modified Alleged RC4)	38
3.1.2.2	Secure and Serialize Message to be Added as Secured Payload	38
3.1.2.2.1	Inputs	39
3.1.2.2.2	Serialize Header Element	39
3.1.2.2.3	Serialize Payload Element	39
3.1.2.2.4	Compute Message Digest	40
3.1.2.2.5	Encrypt Serialized Payload Element	40
3.1.2.2.6	Compute Message Authentication Code	40
3.1.2.2.7	Create Encrypted Element	40
3.1.2.2.8	Create Authenticator Element	40
3.1.2.2.9	Serialize Secured Fragment, Output	40
3.1.2.3	Open Secured Payload Element	41
3.1.2.3.1	Inputs	41
3.1.2.3.2	Parse Encrypted Element	41
3.1.2.3.3	Parse Authenticator Element	41
3.1.2.3.4	Delete Encrypted Element and Authenticator Element	41
3.1.2.3.5	Serialize Header Element	41
3.1.2.3.6	Decrypt Serialized Payload Element	42
3.1.2.3.7	Compute Message Digest	42

3.1.2.3.8	Verify Data Integrity	42
3.1.2.3.9	Deserialize Decrypted Content and Output	42
3.2	Server Details	42
3.2.1	Abstract Data Model	42
3.2.2	Timers	44
3.2.3	Initialization	44
3.2.3.1	Relay Server Certificate	44
3.2.3.1.1	Relay Server Certificates XML File	44
3.2.3.1.2	Certificate Content	45
3.2.4	Message Processing Events and Sequencing Rules	47
3.2.4.1	Registration	47
3.2.4.1.1	Registration Request	48
3.2.4.1.2	Registration Response	48
3.2.4.1.3	Registration Message Processing	49
3.2.4.1.3.1	Parse Incoming Envelope	49
3.2.4.1.3.2	Parse Secured elements	49
3.2.4.1.3.3	Decrypt Shared Key	50
3.2.4.1.3.4	Delete Encrypted Element and Authenticator Element	50
3.2.4.1.3.5	Serialize Header Element	50
3.2.4.1.3.6	Decrypt Serialized Payload Element	50
3.2.4.1.3.7	Compute Message Digest	50
3.2.4.1.3.8	Verify Signature	50
3.2.4.1.3.9	Authenticate Client	51
3.2.4.1.3.10	Prepare Registration Response	51
3.2.4.1.3.11	Secure Message, Prepare Envelope, Send	51
3.2.4.2	Server Side Common Application Message Processing Rules	51
3.2.4.2.1	Parse Incoming Envelope and Get Shared Key	51
3.2.4.2.2	Open Secured Content	51
3.2.4.2.3	Process Application Request and Prepare Application Response	52
3.2.4.2.4	Secure Message	52
3.2.4.2.5	Prepare Envelope and Send	52
3.2.4.3	RelayDefault	52
3.2.4.3.1	RelayDefault Request	52
3.2.4.3.2	RelayDefault Response	52
3.2.4.4	RelayQuiescent	53
3.2.4.4.1	RelayQuiescent Request	53
3.2.4.4.2	RelayQuiescent Response	53
3.2.4.5	userAdd	53
3.2.4.5.1	userAdd Request	53
3.2.4.5.2	userAdd Response	53
3.2.4.6	userPurge	53
3.2.4.6.1	userPurge Request	54
3.2.4.6.2	userPurge Response	54
3.2.4.7	accountModify	54
3.2.4.7.1	accountModify Request	54
3.2.4.7.2	accountModify Response	54
3.2.5	Timer Events	54
3.2.6	Other Local Events	54
3.3	Client Details	54
3.3.1	Abstract Data Model	54
3.3.2	Timers	55
3.3.3	Initialization	55
3.3.4	Message Processing Events and Sequencing Rules	55

3.3.4.1	Registration	55
3.3.4.1.1	Registration Message Processing	56
3.3.4.1.1.1	Generate Shared Key and Prepare Payload and Secured elements	56
3.3.4.1.1.2	Add Certificate Information	56
3.3.4.1.1.3	Encrypt Shared Key and Add to Secured element	56
3.3.4.1.1.4	Serialize Fragment Element	57
3.3.4.1.1.5	Compute Message Digest.....	57
3.3.4.1.1.6	Encrypt Serialized Payload Element and Add to Encrypted Element	57
3.3.4.1.1.7	Compute Signature and Add Authenticator Element	57
3.3.4.1.1.8	Serialize Header Element.....	57
3.3.4.1.1.9	Prepare Envelope and Send	57
3.3.4.1.1.10	Parse Incoming Response Envelope.....	58
3.3.4.1.1.11	Open Secured Content	58
3.3.4.1.1.12	Post Registration Processing.....	58
3.3.4.2	Client Side Common Application Message Processing Rules	58
3.3.4.2.1	Prepare Application Request	58
3.3.4.2.2	Secure Message	58
3.3.4.2.3	Prepare Envelope and Send	59
3.3.4.2.4	Parse Incoming Response Envelope	59
3.3.4.2.5	Open Secured Content.....	59
3.3.4.3	RelayDefault	60
3.3.4.4	RelayQuiescent.....	60
3.3.4.4.1	Build/Rebuild the Server's User Database.....	60
3.3.4.5	userAdd.....	60
3.3.4.6	userPurge	60
3.3.4.7	accountModify	61
3.3.5	Timer Events	61
3.3.6	Other Local Events	61
4	Protocol Examples.....	62
4.1	Serialization	62
4.2	Registration	62
4.3	Build Relay Server's User Database Example	65
4.4	Enable/Disable User Relay Service	70
4.4.1	Disable User Relay Services	70
4.4.2	Enable User Relay Services.....	72
5	Security.....	75
5.1	Security Considerations for Implementers.....	75
5.1.1	Use of semi-weak algorithms	75
5.1.2	Use of weak algorithms	75
5.1.3	Insufficient encryption of protocol messages.....	75
5.1.4	Use of the same key for encryption and MAC	75
5.1.5	Lack of nonce or sequence number to prevent replay attacks.....	75
5.1.6	Out-of-band Certificate Exchange	75
5.2	Index of Security Parameters	75
6	Appendix A: Message Schemas.....	79
6.1	Request Message Schemas	79
6.2	Response Message Schemas	80
7	Appendix B: Product Behavior.....	83
8	Change Tracking.....	85

9 Index	86
----------------------	-----------

1 Introduction

This document specifies the Management Server to Relay Server Groove SOAP Protocol. The management server communicates with the relay server to set up and manage relay service for users. The relay server provides services for users when direct peer-to-peer communication is not possible. This protocol is built on top of a custom secure implementation of an early version of SOAP. It differs from the standard SOAP, and it does not support WSDL.

The protocol consists of service interfaces for:

- Management server and relay server registration for secure communication
- Changing relay server settings
- Changing relay server's states for building the server's user database
- Adding new users to relay server's user database
- Enabling and disabling user's relay services
- Purging user's data on relay servers

The protocol depends on the proper use of these interfaces. This document specifies the proper use of all service interfaces.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

certificate
decryption
Distinguished Encoding Rules (DER)
encryption
Hash-based Message Authentication Code (HMAC)
Message Authentication Code (MAC)
private key
public key
Rivest-Shamir-Adleman (RSA)
secret key
self-signed certificate
symmetric key
Unicode

The following terms are defined in [\[MS-OFCGLOS\]](#):

management server
relay server

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [\[RFC2119\]](#). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, <http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624>, as an additional source.

[MS-GRVSSTP] Microsoft Corporation, "[Simple Symmetric Transport Protocol \(SSTP\) Specification](#)"

[MS-GRVSSTPS] Microsoft Corporation, "[Simple Symmetric Transport Protocol \(SSTP\) Security Protocol Specification](#)"

[PKCS1] RSA Laboratories, "PKCS #1: RSA Cryptography Standard", PKCS #1, Version 2.1, June 2002, <http://www.rsa.com/rsalabs/node.asp?id=2125>

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/rfc/rfc2119.txt>

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, <http://www.ietf.org/rfc/rfc2616.txt>

[RFC3174] Eastlake III, D., and Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001, <http://www.ietf.org/rfc/rfc3174.txt>

[RFC3280] Housley, R., Polk, W., Ford, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002, <http://www.ietf.org/rfc/rfc3280.txt>

[RFC4634] Eastlake III, D., and Hansen, T., "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, July 2006, <http://www.ietf.org/rfc/rfc4634.txt>

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, <http://www.ietf.org/rfc/rfc4648.txt>

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099.

If you have any trouble finding [SCHNEIER], please check [here](#).

[XMLNS] Bray, T., Hollander, D., Layman, A., et al., Eds., "Namespaces in XML 1.0 (Third Edition)", W3C Recommendation, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208/>

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>

[XMLSCHEMA2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001, <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

[XPATh] Clark, J. and DeRose, S., "XML Path Language (XPath), Version 1.0", W3C Recommendation, November 1999, <http://www.w3.org/TR/xpath>

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "[Windows Protocols Master Glossary](#)".

[MS-OFGLGLOS] Microsoft Corporation, "[Microsoft Office Master Glossary](#)".

1.3 Protocol Overview (Synopsis)

The protocol is a request and response based client/server protocol. It is built on top of a custom and secure implementation of a SOAP protocol that uses HTTP 1.1 as its transport. The protocol enables the client to manage the server's services to the users. Using the protocol, the client can change the server's default service settings, add users to the server for services, enable/disable services for users, or purge user data on the server.

The following figure shows the protocol service interfaces.

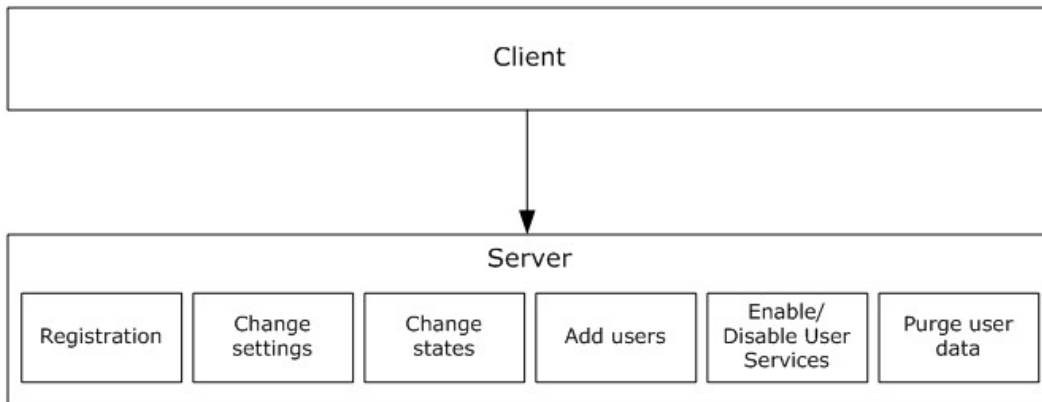


Figure 1: Protocol service interfaces

The system involves two active entities: the **management server** (the protocol client), and the **relay server** (the protocol server). Unless explicitly stated otherwise, the term "client" refers to a management server, and the term "server" refers to a relay server.

The following figure shows the message exchange.

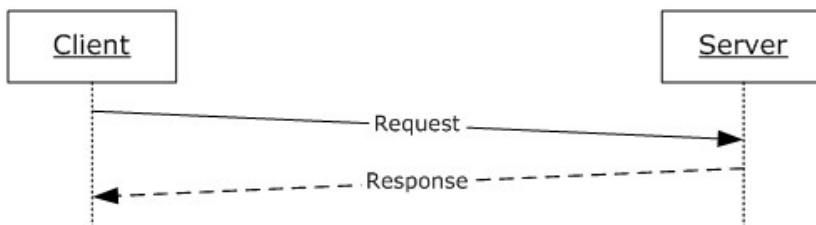


Figure 2: Message exchange between protocol client and protocol server

1.3.1 Service Initialization

1.3.1.1 Server Bootstrapping

Server bootstrapping is an initialization step that the server completes before it services any client requests. As part of the server bootstrapping, the server generates two sets of private- **public key**

pairs, one for **encryption** purposes and one for signature purposes, if such two sets of keys do not exist yet. These keys are 2048-bit long **Rivest-Shamir-Adleman (RSA)** keys. The server also generates a **self-signed certificate**, containing the public keys, if the **certificate (1)** does not exist yet.

1.3.1.2 Client Bootstrapping

Client bootstrapping is a set of initialization steps that the client completes before it communicates with a server. In this case, the client is a management server, so this is the management server's initialization step. As part of the bootstrapping, the client generates two sets of private-public key pairs, one for encryption purposes and one for signature purposes, if such two sets of keys do not exist yet. These keys are 2048 bits long RSA keys. The client also generates a self-signed certificate, containing the public keys, if the certificate does not exist yet.

1.3.1.3 Out-Of-Band Certificate Exchange

Because the certificates are self-signed, the protocol security depends on an out-of-band means to exchange of the certificates between the two entities. That is, it is required that before the communication from a management server to a relay server happens, the management server has already obtained the relay server's certificate and the relay server has already obtained the management server's certificate, so that each has access to the public keys of the other.

1.3.2 Service Interfaces

1.3.2.1 Client/Server Registration

The client registers with the server before any other service request. The registration is a mechanism for exchanging a shared **secret key** used in message data encryption, **decryption**, and message integrity protection and verification.

1.3.2.2 Changing Server Settings

This service interface changes the server's default settings. The client can optionally use this service to customize the behavior of the server.

1.3.2.3 Changing Server States

After client and server registration, the server has two states: active and inactive. In the active state, the server provides relay services to users. In the inactive state, the server does not provide relay services to users. The active and inactive states are used for building or rebuilding the server's user database.

Before building or rebuilding a server's user database, the client sets the server to the inactive state. Once the server is in the inactive state, the client can use the add user service requests to add users to the server's user database. After the completion of building or rebuilding the server's user database, the client sets the server to the active state so that the server can provide services to users.

1.3.2.4 Adding Users

This service interface adds users to the server's user database. It is used to build or rebuild the server's user database or to add new users to the existing user database.

1.3.2.5 Enabling/Disabling User Relay Services

This service interface enables or disables relay services for specified users.

1.3.2.6 Purging User Data

This service interface purges user's relay service data from the server without deleting the users.

1.3.3 Protocol Security

The protocol messages are encrypted and data integrity protected to prevent an attacker from reading or modifying these messages.

Prior to sending application requests to a relay server, a client first registers with the server. A 160-bit symmetric secret key shared only between the client and the server is established during the registration process. This key is thereafter used for protecting the confidentiality and integrity of the data in the protocol.

In the registration message, the client-generated shared key is encrypted using the encryption public key of the server. This message also contains an authenticator obtained by signing the digest of the message, using the client's signature **private key**.

The server, upon receiving such a registration message, decrypts the shared key using its encryption private key, and verifies the signature using the client's signature public key.

The server saves the shared key and returns a response indicating the success of the registration.

Once the shared key has been exchanged with the registration process, the client encrypts subsequent message payloads and integrity-protects them using **Hash-based Message Authentication Code (HMAC)**. The server decrypts the message with the shared key and verifies the integrity before processing the message. Then the server prepares the response, encrypts and integrity-protects it with the shared key as well. Upon receiving such a response, the client decrypts and verifies with the shared key to get the response payload.

If the server does not recognize the client (hence the shared key has not been previously exchanged), the server returns a fault response message indicating that registration is required. The client then goes through the registration process to establish the shared key.

1.4 Relationship to Other Protocols

The protocol uses a custom secure implementation of SOAP for formatting request and response messages. It transmits these messages using HTTP over TCP/IP.

The following diagram shows the transport stack used by the protocol.

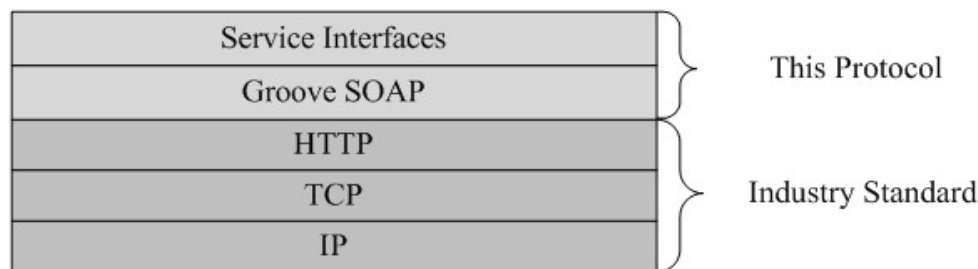


Figure 3: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

The protocol security depends on an out-of-band means to exchange the certificates between the client and the server. It is required that the client and the server have already obtained each other's certificate before exchanging messages so that each has access to the public keys of the other.

1.6 Applicability Statement

This protocol manages relay services for users.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The protocol MUST support message transport over HTTP (as specified in [RFC2616](#)) over TCP/IP.

2.2 Message Syntax

This section defines the grammar of messages used by the protocol using XML (Extensible Markup Language) Schema. For more details about XML Schema, see [XMLSCHEMA1](#) and [XMLSCHEMA2](#). XPath (see [XPATH](#)) is used to address the elements and attributes in XML Schemas.

2.2.1 Namespaces

This specification uses the following XML namespaces (see [XMLNS](#) for details about XML namespaces).

Prefix	Namespace URI
SOAP-ENC	http://schemas.xmlsoap.org/soap/encoding/
SOAP-ENV	http://schemas.xmlsoap.org/soap/envelope/
xs	http://www.w3.org/2001/XMLSchema
xsd	http://www.w3.org/1999/XMLSchema
xsi	http://www.w3.org/1999/XMLSchema-instance
g	urn:groove.net

2.2.2 Common Schema

The following specifies the protocol message schemas:

Request Message Schemas

```
<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:import/>
<xs:element name="Envelope">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Body">
        <xs:complexType>
          <xs:sequence>
            <xs:choice>
              <xs:element ref="accountModify"/>
              <xs:element ref="Registration"/>
              <xs:element ref="RelayDefault"/>
              <xs:element ref="RelayQuiescent"/>
              <xs:element ref="userAdd"/>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

        <xs:element ref="userPurge"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
</xs:complexType>
</xs:element>
<xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>

```

The referenced child elements of the **Body** element are specified in the following schema:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/1999/XMLSchema-instance"/>

  <xs:element name="accountModify" type="ServiceRequestType"/>
  <xs:element name="Registration" type="ServiceRequestType"/>
  <xs:element name="RelayDefault" type="ServiceRequestType"/>
  <xs:element name="RelayQuiescent" type="ServiceRequestType"/>
  <xs:element name="userAdd" type="ServiceRequestType"/>
  <xs:element name="userPurge" type="ServiceRequestType"/>

  <xs:complexType name="ServiceRequestType">
    <xs:sequence>
      <xs:element name="Version" type="NumericType"/>
      <xs:element name="Payload" type="PayloadType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="NumericType">
    <xs:simpleContent>
      <xs:extension base="xsd:int">
        <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
          ref="xsi:type" user="required" fixed="xsd:int"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="PayloadType">
    <xs:attribute name="data" type="xs:base64Binary" use="required"/>
    <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      ref="xsi:type" user="required" fixed="binary"/>
  </xs:complexType>
</xs:schema>

```

The referenced "xsi:type" is specified in the following schema:

```

<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="type" type="xs:string"/>
</xs:schema>

```

Response Message Schemas

```
<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import/>
  <xs:element name="Envelope">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Body">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:element ref="accountModify"/>
                <xs:element ref="Registration"/>
                <xs:element ref="RelayDefault"/>
                <xs:element ref="RelayQuiescent"/>
                <xs:element ref="userAdd"/>
                <xs:element ref="userPurge"/>
              </xs:choice>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>
```

The referenced child elements of the **Body** element are specified in the following schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/1999/XMLSchema-instance"/>

  <xs:element name="accountModify" type="ServiceResponseType"/>
  <xs:element name="Registration" type="ServiceResponseType"/>
  <xs:element name="RelayDefault" type="ServiceResponseType"/>
  <xs:element name="RelayQuiescent" type="ServiceResponseType"/>
  <xs:element name="userAdd" type="ServiceResponseType"/>
  <xs:element name="userPurge" type="ServiceResponseType"/>

  <xs:complexType name="ServiceResponseType">
    <xs:sequence>
      <xs:element name="Payload" type="PayloadType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="NumericType">
    <xs:simpleContent>
      <xs:extension base="xsd:int">
        <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
          ref="xsi:type" use="required" fixed="xsd:int"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```



```

    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="PayloadType">
  <xs:attribute name="data" type="xs:base64Binary" use="required"/>
  <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    ref="xsi:type" use="required" fixed="binary"/>
</xs:complexType>

</xs:schema>

```

The referenced "xsi:type" is specified in the following schema:

```

<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="type" type="xs:string"/>
</xs:schema>

```

The following specifies the service fault response message schema:

```

<xs:schema xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Envelope">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Body">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Fault" type="ServiceFaultResponseType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="ServiceFaultResponseType">
    <xs:sequence>
      <xs:element name="faultCode" type="xs:int"/>
      <xs:element name="faultString" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>

```

2.2.2.1 Common Elements

The following table summarizes the set of common XML Schema element definitions defined by this specification. The messages and the XML Schema elements that are specific to a single message are defined in section [2.2.3](#) of this document.

Element	Description
Fault	Service fault response
fragment	Data fragment
Payload	Base64 encoded data for a service
Version	Service version

2.2.2.1.1 Fault Element

The **Fault** element is used in a service fault response.

```
<xs:element name="Fault" type="ServiceFaultResponseType"/>
```

The ServiceFaultResponseType is specified in section [2.2.2.2.4](#).

2.2.2.1.2 fragment Element

This **fragment** element is used for the **PayloadType** data for all messages except for the **Registration** (sections [2.2.3.4](#) and [3.2.4.1](#)) request message. The **PayloadType** is specified in section [2.2.2.2.3](#).

```
<xs:schema xmlns:g="urn:groove.net" attributeFormDefault="unqualified"
  elementFormDefault="qualified" targetNamespace="urn:groove.net"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import/>
  <xs:element name="fragment">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Payload"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Enc">
          <xs:complexType>
            <xs:attribute name="EC" type="xs:base64Binary" use="required"/>
            <xs:attribute name="IV" type="xs:base64Binary" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="Auth">
          <xs:complexType>
            <xs:attribute name="MAC" type="xs:base64Binary" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The referenced **Payload** element is specified as:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="urn:groove.net"/>
  <xs:element name="Payload">
    <xs:complexType>
      <xs:sequence>
        <xs:element xmlns:g="urn:groove.net" ref="g:SE"/>
      </xs:sequence>
      <xs:attribute name="ManagementServer" type="xs:string" use="required"/>
      <xs:attribute name="Method" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The referenced "g:SE" is specified in the "fragment" element schema that is defined in this section.

The following table describes the elements and attributes.

XPath	Description
/fragment	Fragment element
/fragment/Payload	Payload element
/fragment/Payload/@ManagementServer	The management server
/fragment/Payload/@Method	The requested service name
/fragment/Payload/SE	Secure data element
/ fragment/Payload/SE /Enc	Encrypted content element
/ fragment/Payload/SE /Enc/@EC	Encrypted content
/ fragment/Payload/SE /Enc/@IV	Initialization vector for the encryption and decryption
/ fragment/Payload/SE /Auth	Authenticator element
/ fragment/Payload/SE /Auth/@MAC	Message Authentication Code (MAC)

2.2.2.1.3 Payload Element

The **Payload** element contains the payload data for a service.

```

<xs:element name="Payload" type="PayloadType"/>

```

The PayloadType is specified in section [2.2.2.2.3](#).

2.2.2.1.4 Version Element

The **Version** element contains service version information.

```

<xs:element name="Version" type="NumericType"/>

```

The NumericType is specified in section [2.2.2.2.2](#).

2.2.2.2 Common Types

The following table summarizes the set of common XML Schema type definitions defined by this specification. XML Schema type definitions that are specific to a particular message are defined in section [2.2.3](#) of this document.

Type	Description
BooleanType	A restricted version of the xs:boolean type
NumericType	Extended from xs:int type
PayloadType	Service payload type
ServiceFaultResponseType	Service fault response type
ServiceRequestType	Service request type
ServiceResponseType	Service response type

2.2.2.2.1 BooleanType

The **BooleanType** is specified as:

```
<xs:simpleType name="BooleanType">
  <xs:restriction base="xs:boolean">
    <xs:pattern value="0 | 1"/>
  </xs:restriction>
</xs:simpleType>
```

2.2.2.2.2 NumericType

The **NumericType** is specified as:

```
<xs:complexType name="NumericType">
  <xs:simpleContent>
    <xs:extension base="xsd:int">
      <xs:attribute ref="xsi:type" user="required" fixed="xsd:int"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

2.2.2.2.3 PayloadType

The **PayloadType** contains service specific payload data.

```
<xs:complexType name="PayloadType">
  <xs:sequence>
    <xs:attribute name="data" type="xs:base64Binary" use="required"/>
    <xs:attribute ref="xsi:type" user="required" fixed="binary"/>
  </xs:sequence>
</xs:complexType>
```

The following table describes the attributes of the type.

XPath	Description
/PayloadType/@data	Service payload data
/PayloadType/@type	Payload data type

The /PayloadType/@data attribute contains service specific message data specified in section [2.2.3](#).

2.2.2.2.4 ServiceFaultResponseType

The **ServiceFaultResponseType** contains a fault code and a fault string.

```
<xs:complexType name="ServiceFaultResponseType">
  <xs:sequence>
    <xs:element name="faultCode" type="xs:int"/>
    <xs:element name="faultString" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

The following table describes the contained elements.

XPath	Description
/ServiceFaultResponseType/faultCode	The service fault reason code element
/ServiceFaultResponseType/faultString	The service fault description element

A fault code MUST be one of the following.

Fault Code	Description
301	Required HTTP content type header missing
302	Invalid target
303	Payload missing
304	Registration required
305	Decryption failed
306	COM exception
307	Not used
308	COM Fault message exception
309	COM invalid method
310	COM invalid property value
311	COM no rows
312	COM no properties
313	SOAP disabled

2.2.2.2.5 ServiceRequestType

The **ServiceRequestType** contains service request data.

```
<xs:complexType name="ServiceRequestType">
  <xs:sequence>
    <xs:element name="Version" type="NumericType"/>
    <xs:element name="Payload" type="PayloadType"/>
  </xs:sequence>
</xs:complexType>
```

The **NumericType** and the **PayloadType** are specified in sections [2.2.2.2.2](#) and [2.2.2.2.3](#).

The following table describes the contained elements.

XPath	Description
/ServiceType/Payload	Service request payload element
/ServiceType/Version	Protocol version element

2.2.2.2.6 ServiceResponseType

The **ServiceResponseType** contains service response data.

```
<xs:complexType name="ServiceResponseType">
  <xs:sequence>
    <xs:element name="Payload" type="PayloadType"/>
  </xs:sequence>
</xs:complexType>
```

The **PayloadType** is specified in section [2.2.2.2.3](#).

The following table describes the contained element.

XPath	Description
/ServiceType/Payload	Service request payload element

2.2.2.3 Common Attributes

The following table summarizes the set of common XML Schema attribute definitions defined by this specification. XML Schema attributes that are specific to a particular message are defined in section [2.2.3](#) of this document.

Attribute	Description
data	Service payload data
epoch	Indicates if the server's user database needs to be built or rebuilt
rowCount	Child element count
userId	A globally unique identifier (GUID) uniquely identifying a user

Attribute	Description
EC	Encrypted content
IV	Initialization vector for the encryption and decryption
MAC	Message Authentication Code

2.2.2.3.1 data

The **data** attribute specifies a service response payload data.

```
<xs:attribute name="data" type="xs:base64Binary"/>
```

2.2.2.3.2 epoch

The **epoch** attribute in a response message indicates if the server's user database needs to be built or rebuilt. A value of 0 indicates that the client [1](#) MUST build or rebuild the server's user database. Nonzero values indicate that the client SHOULD NOT build or rebuild the server's user database.

```
<xs:attribute name="epoch" type="xs:int"/>
```

2.2.2.3.3 rowCount

The **rowCount** attribute specifies the number of the contained elements.

```
<xs:attribute name="rowCount" type="xs:int"/>
```

2.2.2.3.4 userId

The **userId** attribute specifies a GUID uniquely identifying the user.

```
<xs:attribute name="userId" type="xs:string"/>
```

2.2.2.3.5 EC

The **EC** attribute specifies the encrypted and base64 encoded content.

```
<xs:attribute name="EC" type="xs:base64Binary"/>
```

2.2.2.3.6 IV

The **IV** attribute specifies the initialization vector for the encryption and decryption.

```
<xs:attribute name="IV" type="xs:base64Binary"/>
```

2.2.2.3.7 MAC

The **MAC** attribute specifies the Message Authentication Code.

```
<xs:attribute name="MAC" type="xs:base64Binary"/>
```

2.2.3 Message Definitions

The following table lists the protocol messages.

Message	Description
accountModify request	Request for enabling/disabling user's relay services
accountModify response	Response for the accountModify request
Fault response	Service fault response
Registration request	Request for registration
Registration response	Response for the Registration request
RelayQuiescent request	Request for changing the server's state
RelayQuiescent response	Response for the RelayQuiescent request
RelayDefault request	Request for changing the server's default settings
RelayDefault response	Response for the RelayDefault request
userAdd request	Request for adding users to the server's user database
userAdd response	Response for the userAdd request
userPurge request	Request for purging user data from the server
userPurge response	Response for the userPurge request

2.2.3.1 accountModify Request Message

The **accountModify** request message enables or disables relay services for users.

```
<xs:element name="accountModify" type="ServiceRequestType"/>
```

The ServiceRequestType is specified in section [2.2.2.2.5](#).

2.2.3.1.1 accountModify Request Payload

The /accountModify/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#). In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="accountModify">
  <xs:complexType>
    <xs:sequence>
```



```

<xs:element name="user" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="lockout" type="BooleanType" use="required"/>
    <xs:attribute name="userId" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="rowCount" type="xs:int" use="required"/>
</xs:complexType>
</xs:element>

```

The **BooleanType** is specified in section [2.2.2.2.1](#).

The following table describes the XML elements and attributes.

XPath	Description
/accountModify	accountModify request element
/accountModify/@rowCount	The count of the user elements
/accountModify/user	The user element
/accountModify/user/@lockout	Control to enable or disable relay services for the user. To disable the relay services for the user, it MUST be set to 1. To enable the relay services for the user, it MUST be set to 0.
/accountModify/user/@userId	User identification, a GUID

2.2.3.2 accountModify Response Message

The **accountModify** response message is the non-fault response message for the **accountModify** request.

```

<xs:element name="accountModify" type="ServiceResponseType"/>

```

The ServiceResponseType is specified in section [2.2.2.2.6](#).

2.2.3.2.1 accountModify Response Payload

The /accountModify/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```

<xs:element name="accountModify">
  <xs:complexType>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>

```

The following table describes the element and attribute.

XPath	Description
/accountModify	accountModify response element
/accountModify/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .

2.2.3.3 Fault Response Message

The **Fault** response message is the response message for any service request fault.

```
<xs:element name="Fault" type="ServiceFaultResponseType"/>
```

The ServiceFaultResponseType is specified in section [2.2.2.2.4](#).

2.2.3.4 Registration Request Message

The **Registration** request message requests registration with the server.

```
<xs:element name="Registration" type="ServiceRequestType"/>
```

The ServiceRequestType is specified in section [2.2.2.2.5](#).

2.2.3.4.1 Registration Request Payload

The /Registration/Payload/@data attribute contains the payload fragment data specified as:

```
<xs:schema xmlns:g="urn:groove.net" attributeFormDefault="unqualified"
  elementFormDefault="qualified" targetNamespace="urn:groove.net"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import/>
  <xs:element name="fragment">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Payload"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Cert">
          <xs:complexType>
            <xs:attribute name="EPKAlgo" type="xs:string" use="required"
              fixed="RSA"/>
            <xs:attribute name="EPubKey" type="xs:base64Binary" use="required"/>
            <xs:attribute name="EncAlgo" type="xs:string" use="required"
              fixed="RSA"/>
            <xs:attribute name="SPKAlgo" type="xs:string" use="required"
              fixed="RSA"/>
            <xs:attribute name="SPubKey" type="xs:base64Binary" use="required"/>
            <xs:attribute name="SigAlgo" type="xs:string" use="required"
              fixed="RSA"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```

<xs:element name="Enc">
  <xs:complexType>
    <xs:attribute name="EC" type="xs:base64Binary" use="required"/>
    <xs:attribute name="IV" type="xs:base64Binary" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="Auth">
  <xs:complexType>
    <xs:attribute name="Sig" type="xs:base64Binary" use="required"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="EncryptedKey" type="xs:base64Binary"
  use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>

```

The referenced **Payload** element is specified in the following schema:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="urn:groove.net"/>
  <xs:element name="Payload">
    <xs:complexType>
      <xs:sequence>
        <xs:element xmlns:g="urn:groove.net" ref="g:SE"/>
      </xs:sequence>
      <xs:attribute name="ManagementServer" type="xs:string" use="required"/>
      <xs:attribute name="Method" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The referenced "g:SE" element is specified in the "fragment" schema that is defined in this section.

The following table describes the elements and attributes.

XPath	Description
/fragment	Fragment element
/fragment/Payload	Payload element
/fragment/Payload/@ManagementServer	The protocol client URL (Uniform Resource Locator)
/fragment/Payload/@Method	Service name
/fragment/Payload/SE	Secured element
/fragment/Payload/SE/@EncryptedKey	Encrypted shared secret key
/fragment/Payload/SE/Cert	Certificate element
/fragment/Payload/SE/Cert/@EPKAlgo	Encryption public key algorithm. The value MUST be "RSA".
/fragment/Payload/SE/Cert/@EPubKey	Encryption public key. The key MUST be Distinguished

XPath	Description
	Encoding Rules (DER) -encoded.
/fragment/Payload/SE/Cert/@EncAlgo	Encryption algorithm. The value MUST be "RSA".
/fragment/Payload/SE/Cert/@SPKAlgo	Signature public key algorithm. The value MUST be "RSA".
/fragment/Payload/SE/Cert/@SPubKey	Signature public key. The key MUST be DER-encoded.
/fragment/Payload/SE/Cert/@SigAlgo	Signature algorithm. The value MUST be "RSA".
/fragment/Payload/SE/Enc	Encrypted content element
/fragment/Payload/SE/Enc/@EC	Encrypted content
/fragment/Payload/SE/Enc/@IV	Initialization vector for the encryption and decryption
/fragment/Payload/SE/Auth	Authenticator element
/fragment/Payload/SE/Auth/@Sig	Message signature

The `fragment/Payload/SE/Enc/@EC` attribute contains an empty **Payload** XML element before encryption specified as:

```
<xs:element name="Payload"/>
```

2.2.3.5 Registration Response Message

The **Registration** response message is the non-fault response message for the **Registration** request.

```
<xs:element name="Registration" type="ServiceResponseType"/>
```

The `ServiceResponseType` is specified in section [2.2.2.2.6](#).

2.2.3.5.1 Registration Response Payload

The `/Registration/Payload/@data` attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the `/fragment/Payload/SE/Enc/@EC` attribute data before encryption is specified as:

```
<xs:element name="Registration">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Registration">
        <xs:complexType>
          <xs:attribute name="ErrorMessage" type="xs:string" use="required"/>
          <xs:attribute name="Status" type="xs:int" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
```

```
</xs:element>
```

The following table describes the elements and attributes.

XPath	Description
/Registration	Registration response element
/Registration/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .
/Registration/Registration	Registration response data element
/Registration/Registration/@ErrorMessage	Error message describing an error
/Registration/Registration/@Status	Service status. It MUST be 0 for a successful request.

2.2.3.6 RelayDefault Request Message

The **RelayDefault** request message changes the server's default settings.

```
<xs:element name="RelayDefault" type="ServiceRequestType"/>
```

The **ServiceRequestType** is specified in section [2.2.2.2.5](#).

2.2.3.6.1 RelayDefault Request Payload

The /RelayDefault/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="RelayDefault">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="relay">
        <xs:complexType>
          <xs:attribute name="deviceLifetime" type="xs:int" use="required"/>
          <xs:attribute name="deviceTargetQuotaSize" type="xs:int" use="required"/>
          <xs:attribute name="identityLifetime" type="xs:int" use="required"/>
          <xs:attribute name="identityTargetQuotaSize" type="xs:int" use="required"/>
          <xs:attribute name="purgeEnabled" type="BooleanType" use="required"/>
          <xs:attribute name="quotaEnabled" type="BooleanType" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The **BooleanType** is specified in section [2.2.2.2.1](#).

The following table describes the elements and attributes.

XPath	Description
/RelayDefault	RelayDefault request element
/RelayDefault/relay	Relay default data element
/RelayDefault/relay /@deviceLifetime	Device life time in days
/RelayDefault/relay/@deviceTargetQuotaSize	Device target quota size in megabytes
/RelayDefault/relay/@identityLifetime	Identity life time in days
/RelayDefault/relay/@identityTargetQuotaSize	Identity target quota size in megabytes
/RelayDefault/relay/@purgeEnabled	Indicates if purge is enabled. A value of 1 indicates that purge is enabled; a value of 0 indicates that purge is disabled.
/RelayDefault/relay/@quotaEnabled	Indicates if quota is enabled. A value of 1 indicates that quota is enabled; a value of 0 indicates that quota is disabled.

2.2.3.7 RelayDefault Response Message

The **RelayDefault** response message is the non-fault response message for the **RelayDefault** request.

```
<xs:element name="RelayDefault" type="ServiceResponseType"/>
```

The **ServiceResponseType** is specified in section [2.2.2.2.6](#).

2.2.3.7.1 RelayDefault Response Payload

The /RelayDefault/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="RelayDefault">
  <xs:complexType>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the element and attribute.

XPath	Description
/RelayDefault	RelayDefault response element
/RelayDefault/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .

2.2.3.8 RelayQuiescent Request Message

The **RelayQuiescent** request message changes the server's state.

```
<xs:element name="RelayQuiescent" type="ServiceRequestType"/>
```

The **ServiceRequestType** is specified in section [2.2.2.2.5](#).

2.2.3.8.1 RelayQuiescent Request Payload

The /RelayQuiescent/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="RelayQuiescent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="relay">
        <xs:complexType>
          <xs:attribute name="status" type="xs:int" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The following table describes the elements and attributes.

XPath	Description
/RelayQuiescent	RelayQuiescent request element
/RelayQuiescent/relay	Relay status element
/RelayDefault/relay /@status	The server state to be set. To set the server to the inactive state, status MUST be set to 1. To set the server to the active state, status MUST be set to 0.

2.2.3.9 RelayQuiescent Response Message

The **RelayQuiescent** response message is the non-fault response message for the **RelayQuiescent** request.

```
<xs:element name="RelayQuiescent" type="ServiceResponseType"/>
```

The **ServiceResponseType** is specified in section [2.2.2.2.6](#).

2.2.3.9.1 RelayQuiescent Response Payload

The /RelayQuiescent/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="RelayQuiescent">
  <xs:complexType>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the element and attribute.

XPath	Description
/RelayQuiescent	RelayQuiescent response element
/RelayQuiescent/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .

2.2.3.10 userAdd Request Message

The **userAdd** request adds users to the server's user database.

```
<xs:element name="userAdd" type="ServiceRequestType"/>
```

The **ServiceRequestType** is specified in section [2.2.2.2.5](#).

2.2.3.10.1 userAdd Request Payload

The /userAdd/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="userAdd">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="userId" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="rowCount" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the elements and attributes.

XPath	Description
/userAdd	User add request element

XPath	Description
/userAdd/@rowCount	The count of users
/userAdd/user	User element
/userAdd/user/@userId	A GUID uniquely identifying a user.

2.2.3.11 userAdd Response Message

The **userAdd** response message is the non-fault response message for the **userAdd** request.

```
<xs:element name="userAdd" type="ServiceResponseType"/>
```

The **ServiceResponseType** is specified in section [2.2.2.2.6](#).

2.2.3.11.1 userAdd Response Payload

The /userAdd/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="userAdd">
  <xs:complexType>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the element and attribute.

XPath	Description
/userAdd	User add response element
/userAdd/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .

2.2.3.12 userPurge Request Message

The **userPurge** request message purges the user's relay service data from the server.

```
<xs:element name="userPurge" type="ServiceRequestType"/>
```

The **ServiceRequestType** is specified in section [2.2.2.2.5](#).

2.2.3.12.1 userPurge Request Payload

The /userPurge/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="userPurge">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="userId" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="rowCount" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the elements and attributes.

XPath	Description
/userPurge	User data purging request element
/userPurge/@rowCount	The count of users
/userPurge/user	A user element
/userPurge/user/@userId	A GUID uniquely identifying a user.

2.2.3.13 userPurge Response Message

The **userPurge** response message is the non-fault response message for the **userPurge** request.

```
<xs:element name="userPurge" type="ServiceResponseType"/>
```

The **ServiceResponseType** is specified in section [2.2.2.2.6](#).

2.2.3.13.1 userPurge Response Payload

The /userPurge/Payload/@data attribute contains the payload fragment data as specified in section [2.2.2.1.2](#).

In the payload fragment data, the /fragment/Payload/SE/Enc/@EC attribute data before encryption is specified as:

```
<xs:element name="userPurge">
  <xs:complexType>
    <xs:attribute name="epoch" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

The following table describes the element and attribute.

XPath	Description
/userPurge	User purge response element
/userPurge/@epoch	Indicates if the server's user database needs to be built or rebuilt as specified in section 2.2.2.3.2 .

3 Protocol Details

3.1 Common Details

3.1.1 Common Message Templates

3.1.1.1 Request Message Template

The following is a detailed common request message template using the message definitions in section [2.2.3](#):

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <[[[- MethodName -]]>
      <Version xsi:type="xsd:int">1</Version>
      <Payload data="[[[- payloadData -]]" xsi:type="binary"/>
    </[[[- MethodName -]]>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

[[[- MethodName -]]: Requested operation method name. It MUST be one of the following as specified in section [2.2.2](#):

- **accountModify**
- **Registration**
- **RelayDefault**
- **RelayQuiescent**
- **userAdd**
- **userPurge**

[[[- PayloadData -]]: Base64 encoded payload data. Section [3.1.1.3](#) provides a detailed common payload data template.

3.1.1.2 Response Message Templates

The following is a detailed common non-fault response message template using the message definitions in section [2.2.3](#):

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <[[[- MethodName -]]>
      <Payload data="[[[- PayloadData -]]" xsi:type="binary"/>
    </[[[- MethodName -]]>
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

[[- MethodName -]]: Requested operation method name. It MUST be the same as the corresponding request message's "**[[- MethodName -]]**". For example, if the request message's "**[[- MethodName -]]**" is "userAdd", the response message's "**[[- MethodName -]]**" MUST be "userAdd".

[[- PayloadData -]]: Base64 encoded payload data. Section [3.1.1.3](#) provides a detailed common payload data template.

The following is the detailed **Fault** response message template using the **Fault** response message in section [2.2.3.3](#):

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultCode>[[ - faultCode - ]]</faultCode>
      <faultString>[[ - faultString - ]]</faultString>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

[[- faultCode -]]: A fault code, specified as an integer. It MUST be one of the fault code listed in the **ServiceFaultResponseType** in section [2.2.2.4](#).

[[- faultString -]]: A fault description, specified as a string.

3.1.1.3 Payload Data Template

The following is a detailed common payload data template using the message definitions in section [2.2.3](#), except for the **Registration** request message:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="[[ - URL - ]]" Method="[[ - MethodName - ]]">
    <g:SE>
      <g:Enc EC="[[ - ECData - ]]" IV="[[ - IV - ]]" />
      <g:Auth MAC="[[ - MAC - ]]" />
    </g:SE>
  </Payload>
</g:fragment>
```

[[- MethodName -]]: Requested operation method name. It MUST be the same as the request message's "**[[- MethodName -]]**". For example, if the request message's "**[[- MethodName -]]**" is "userAdd", the response message's "**[[- MethodName -]]**" MUST be "userAdd".

[[- URL -]]: The client URL, specified as a string.

[[- ECData -]]: The encrypted and base64 encoded data. All **[[- ECData -]]** MUST starts with `<?xml version='1.0'?><?groove.net version='1.0'?>` before encryption.

For a request message, the **[[- ECData -]]** contains payload data specific to the request.

The following is a detailed `[- ECData -]` template for a response message, except for the **Registration** response:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<[- MethodName -] epoch="[- epoch -]"/>
```

`[-MethodName -]`: The request method name. It **MUST** be the same as the request method element name in the message **Body**.

`[- epoch -]`: Indicates if the server's user database needs to be built or rebuilt, specified as an integer. A value of 0 indicates that the server's user database **MUST** be built or rebuilt.

`[- IV -]`: The initialization vector for the encryption and decryption, base64 encoded.

`[- MAC -]`: The Message Authentication Code, base64 encoded.

Sections [3.1.2](#), [3.2.4.2](#), and [3.3.4.2](#) specify message encryption/decryption and other security related processing rules.

3.1.2 Common Security Processing Rules

The protocol messages are encrypted and integrity-protected using the key shared by a management server and a relay server. This section specifies how messages are encrypted/decrypted and integrity protected/verified.

Every application message, request or response, is secured with the shared secret key established via the Registration message. The security processing rules to open a secured request and the security processing rules to secure a response are the same for every application message (every message except for Registration). The application message security processing is specified in this section.

3.1.2.1 MARC4 (Modified Alleged RC4)

This protocol security uses a variation of the RC4 algorithm called Modified Alleged RC4, or MARC4.

RC4 algorithm is described in [SCHNEIER].

MARC4 is different from RC4 as following:

- Both encryption and decryption **MUST** use the initialization vector (IV), and IV **MUST** be the same size as the secret key.
- New random IV **MUST** be generated before each time the data is encrypted. Matching IV **MUST** be used each time the data is decrypted.
- IV **MUST** be XOR-ed with the secret key, and the result **MUST** be used as the secret key for RC4.
- The first 256 bytes of the keystream **MUST** be discarded. Subsequent bytes of the keystream **MUST** be used in the same way as they are used with RC4.

3.1.2.2 Secure and Serialize Message to be Added as Secured Payload

This section specifies the security processing rules for securing and serializing a message into a secured payload. It applies to a client securing application request payload, and it applies to a server securing an application response payload. Such a secured payload is to be added into an

envelope in a Request Message or a Response Message (as defined in section [2.2.2](#)) for transmission.

To create a secure payload, the client or server MUST start with the inputs specified in section [3.1.2.2.1](#) and then perform, in order, the procedures specified in sections [3.1.2.2.2](#) through [3.1.2.2.9](#).

3.1.2.2.1 Inputs

Header element: secured fragment element itself as defined in section [2.2.2.1.2](#). As the input, this element MUST have the name of "fragment" in the namespace of "urn:groove.net". It MUST have one child element named "Payload". The **Payload** element MUST contain one child element named "SE" in the namespace of "urn:groove.net". The "g:SE" element is referred as secured element. It MUST contain no content.

Payload element: application request or response element. This element contains content to be secured.

Shared Key: Key used for encryption and data-integrity protection. This is the key shared by a client and a server.

3.1.2.2.2 Serialize Header Element

Header element MUST be serialized into a byte representation. As part of the serialization, all attributes for each element MUST be sorted alphabetically, with no compression and no extra white spaces. UTF8 encoding MUST be used. In addition, the following rules MUST be followed to ensure successful security processing:

- The following MUST be present at the beginning of the serialized data:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
```

- The top level fragment element MUST use the namespace prefix "g", with "xmlns" attribute defining "g" as "urn:groove.net".
- The second level **Payload** element MUST NOT have any namespace prefix.
- The third level "g:SE" element and its sub-elements MUST use the namespace prefix "g" without any "xmlns" attribute.

3.1.2.2.3 Serialize Payload Element

Payload element MUST be serialized into a byte representation. As part of the serialization, all attributes for each element MUST be sorted alphabetically, with no compression and no extra white spaces. UTF8 encoding MUST be used. In addition, the following unique rules MUST be followed to ensure successful security processing:

- The following MUST be present at the beginning of the serialized data:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
```

- Namespace prefix or "xmlns" attribute MUST NOT be added.

3.1.2.2.4 Compute Message Digest

Message digest MUST be computed using SHA1 algorithm, as defined in [\[RFC3174\]](#).

Message digest MUST include serialized header element (as defined in section [3.1.2.2.2](#)), and the serialized payload element (as defined in section [3.1.2.2.3](#)) – in this order.

3.1.2.2.5 Encrypt Serialized Payload Element

Byte representation of the serialized payload element MUST be encrypted using MARC4 algorithm (as defined in section [3.1.2.1](#)), with the shared key.

Unique initialization vector MUST be generated for each message, and used during encryption of that message.

3.1.2.2.6 Compute Message Authentication Code

Message Authentication Code MUST be computed using HMAC-SHA1 algorithm (as defined in [\[RFC4634\]](#)), with the shared key.

Message Authentication Code MUST be computed with the message digest, as defined in section [3.1.2.2.4](#).

3.1.2.2.7 Create Encrypted Element

Encrypted element named "g:Enc" in the namespace of "urn:groove.net" MUST be created as a child of the secured element, as defined in section [2.2.2.1.2](#).

Encrypted element MUST have the attributes as defined in section [2.2.2.1.2](#). Specifically:

- **EC** attribute MUST have the encrypted serialized payload element, as defined in section [3.1.2.2.5](#).
- **IV** attribute MUST be the initialization vector for the encrypted serialized payload element, as defined in section [3.1.2.2.5](#).

3.1.2.2.8 Create Authenticator Element

Authenticator element named "g:Auth" in the namespace of "urn:groove.net" MUST be created as child of the secured element, as defined in section [2.2.2.1.2](#).

Authenticator element MUST have the attributes as defined in section [2.2.2.1.2](#). Specifically:

- **MAC** attribute MUST have the value of the Message Authentication Code, as defined in section [3.1.2.2.6](#).

3.1.2.2.9 Serialize Secured Fragment, Output

At this stage, the header element becomes secured **fragment** element.

Secured **fragment** element, containing secured element, MUST be serialized into a byte representation. The same serialization rules specified in section [3.1.2.2.2](#) MUST be applied here.

The serialized secured **fragment** is the output.

3.1.2.3 Open Secured Payload Element

This section specifies the security processing rules for opening a secured **Payload** element, the decrypted content contains the application request or response. This applies to a client opening a secured response **Payload**, or a server opening a secured request **Payload**.

To open a secured **Payload** element, the client or server MUST start with the inputs specified in section [3.1.2.3.1](#) and then perform, in order, the procedures specified in sections [3.1.2.3.2](#) through [3.1.2.3.9](#).

3.1.2.3.1 Inputs

Secured **fragment** element: secured **fragment** element as defined in section [2.2.2.1.2](#). As the input, this element MUST have the name of "fragment" with a namespace prefix of "g" and an "xmlns" attribute defining "g" as the namespace "urn:groove.net". It MUST have one child element named "Payload", with attributes **ManagementServer** and **Method** set. **Payload** element MUST contain one child element named "SE" in the namespace of "urn:groove.net". The "g:SE" element MUST be as defined in section [2.2.2.1.2](#), with all sub-elements and attributes. The "g:SE" element is referred as the secured element.

Shared Key: Key used for decryption and data-integrity verification. This is the key shared by a client and a server.

3.1.2.3.2 Parse Encrypted Element

Encrypted element "g:Enc" MUST be a child of the secured element and MUST have the attributes as defined in section [2.2.2.1.2](#).

The following attributes MUST be parsed and saved as inputs into decryption:

- **EC** attribute is the encrypted serialized payload element.
- **IV** attribute is the initialization vector for the encrypted serialized payload element.

3.1.2.3.3 Parse Authenticator Element

Authenticator element "g:Auth" MUST be a child of the secured element and MUST have the attributes as defined in section [2.2.2.1.2](#).

The following attributes MUST be parsed and saved as inputs into data integrity verification:

- **MAC** attribute is the Message Authentication Code.

3.1.2.3.4 Delete Encrypted Element and Authenticator Element

Encrypted element "g:Enc" and authenticator element "g:Auth" MUST be deleted as child objects of the secured element, which is part of the input secured fragment element.

Once this is done, secured **fragment** element becomes a header element, as defined in section [3.1.2.2.1](#).

3.1.2.3.5 Serialize Header Element

Header element MUST be serialized as defined in section [3.1.2.2.2](#).

3.1.2.3.6 Decrypt Serialized Payload Element

Encrypted serialized **Payload** element MUST be decrypted using MARC4 algorithm (as defined in section [3.1.2.1](#)), with the shared key.

Corresponding per-message initialization vector from the encrypted element MUST be used, as defined in section [3.1.2.3.2](#).

3.1.2.3.7 Compute Message Digest

Message digest MUST be computed using SHA1 algorithm, as defined in [\[RFC3174\]](#).

Message digest MUST include serialized header element (as defined in section [3.1.2.3.5](#)), and the decrypted content (the result from section [3.1.2.3.6](#)) – in this order.

3.1.2.3.8 Verify Data Integrity

Message Authentication Code MUST be computed using HMAC-SHA1 algorithm (as defined in [\[RFC4634\]](#)), with the shared key.

Message Authentication Code MUST be computed with the message digest, as computed in section [3.1.2.3.7](#).

Comparison of this Message Authentication Code with the one saved from section [3.1.2.3.2](#) MUST be performed. If the Message Authentication Codes do not match, it means the data integrity verification has failed.

If a client is performing the procedure defined in section [3.1.2.3](#) to open a secured **Payload** element in a response message, this failure MUST cause the client to treat the message as a bad response and stop processing the message.

If a server is performing the procedure defined in section [3.1.2.3](#) to open a secured **Payload** element in a request message, this failure MUST cause the server to return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

3.1.2.3.9 Deserialize Decrypted Content and Output

Serialized **Payload** element, which is the decrypted content from section [3.1.2.3.6](#), MUST be deserialized from a byte representation into an XML representation, as the payload element. This **Payload** element is the output.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Registered Client: The server maintains the client that has successfully finished the Registration request with the server.

Server Defaults: The server maintains the following default service settings:

- Device life time: Message life time in days for a device targeted message. After the life time, the message is purged if purge is enabled.
- Device target quota size: Target quota size in megabytes for a device. When quota is enabled, this is the maximum amount of data the server can store for a device.
- Identity life time: Message life time in days for an identity targeted message. After the life time, the message is purged if purge is enabled.
- Identity target quota size: Target quota size in megabytes for an identity. When quota is enabled, this is the maximum amount of data the server can store for an identity.
- Purge enabled: Indicates if purging is enabled.
- Quota enabled: Indicates if quota is enabled.

Users: The server maintains users in a database. A user database entry contains the following data:

- User ID: A GUID uniquely identifying a user.
- Enabled: Indicates if the user is enabled for relay services.

Server States: When communicating with the client, the server maintains the following states:

- Unregistered state
- Active state
- Inactive state

The following figure is the server state transition diagram.

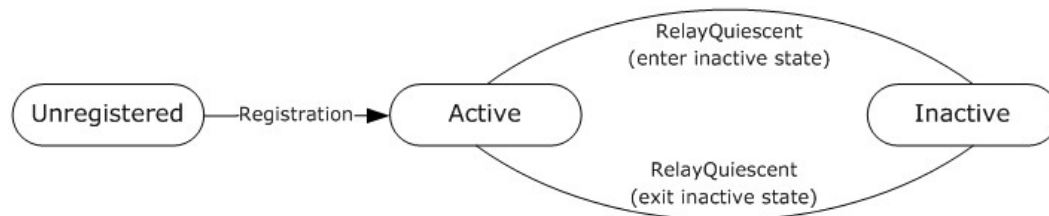


Figure 4: Server state transitions

The server enters the active state from the unregistered state after a successful **Registration** request. The server enters the inactive state after a successful **RelayQuiescent** request to set the server into the inactive state. From the inactive state, the server returns to the active state after a **RelayQuiescent** request to set the server to the active state.

Server Key Pairs: Two sets of asymmetric public-private key pairs, one for encryption, and one for signing.

Server SOAP Certificate: Contains the public keys from the server key pairs.

Shared Secret Key: The server maintains the shared secret key established with the client through the Registration message.

Note that the preceding conceptual data can be implemented by using a variety of techniques. Any data structure that stores the preceding conceptual data can be used in the implementation.

3.2.2 Timers

None.

3.2.3 Initialization

For secure communication, in the initialization process, the server **MUST** generate server key pairs and the server certificate if they do not exist. Specifically, the following initialization steps **MUST** be performed:

1. If the server key pairs do not exist, two new sets of server key pairs, one for encryption and one for signing, **MUST** be generated and stored. They **MUST** be RSA 2048 bits long.
2. If the relay server SOAP certificate does not exist, a new certificate **MUST** be generated in X509.v3 format, and DER encoded, with extensions for signature public key and related information. See section [3.2.3.1.2](#) for the definitions of the certificate extensions. The subject and issuer Common Name (CN) fields of the certificate are set to the relay server SOAP URL. The certificate and relay server identity information are stored in a file as specified in section [3.2.3.1.1](#).
3. The relay server certificate and identity information from the last step **MUST** be passed to the management server through an out-of-band means.

The relay server imports the certificate and management server identity information as part of the out-of-band means to acquire a management server's public keys before any protocol communication can be processed. This makes it possible for the server to authenticate a client (a management server) when the server receives a request. A server **MUST** reject any unknown client.

[<2>](#)

3.2.3.1 Relay Server Certificate

3.2.3.1.1 Relay Server Certificates XML File

The server generates a file containing the following XML segment. This step is performed to support interoperability:

```
<xs:element name="RelayAttributes">
  <xs:attribute name="IsRelay" type="xs:int" use="xs:required"/>
  <xs:attribute name="IsXMPPProxy" type="xs:int"/>
  <xs:attribute name="RelayDeviceURL" type="xs:uri" use="xs:required"/>
  <xs:attribute name="SOAPCertificate" type="xs:base64Binary" use="xs:required"/>
  <xs:attribute name="SOAPURL" type="xs:uri" use="xs:required"/>
  <xs:attribute name="SSTPCertificate" type="xs:base64Binary" use="xs:required"/>
</xs:element>
```

The fields are as follow.

Attribute	Description
IsRelay	This required attribute defines whether the server is serving as a Relay server. This value MUST be 1.
IsXMPPProxy	This is an optional attribute. If the attribute is present, its value MUST be 0.
RelayDeviceURL	This required attribute defines the relay URL for the relay server.

Attribute	Description
SOAPCertificate	This required attribute defines the certificate for the protocol communication. The value of this attribute is the relay server SOAP certificate specified in section 3.2.3, first DER-encoded, and then base64-encoded as specified in [RFC4648].
SOAPURL	This required attribute defines the relay server SOAP URL for the communication from the client to the server in this protocol. This is the entry point for the protocol.
SSTPCertificate	This required attribute defines the certificate for the other Relay communication protocol (SSTP, defined in [MS-GRVSSTP]). The value of this attribute is the relay SSTP certificate as specified in [MS-GRVSSTPS] section 3.1.1.1, first DER-encoded, and then base64-encoded as specified in [RFC4648]. The certificate MUST have extensions as described in section 3.2.3.1.2.

The following is a sample ServerID.xml file:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<RelayAttributes IsRelay="1" IsXMPPProxy="0"
  RelayDeviceURL="grooveDNS://relay.contoso.com"
  SOAPCertificate="MIIET...THNsVxvVAY="
  SOAPURL="http://relay.contoso.com:8009/SOAP"
  SSTPCertificate="MIEGz...19Zz0/6FU="/>
```

As described in section 1.5, a client imports this file as part of the out-of-band means to acquire the server's public keys before it can communicate to the relay server.

3.2.3.1.2 Certificate Content

The certificates generated by a client and a server MUST be in X509.v3 format as defined in [RFC3280]. These certificates MUST also have the following protocol-specific extension object identifiers (OID) (2).

Extension OID	Description
2.16.840.1.114227.1.1.1	Encryption public key. The value MUST be 2048-bit RSA encryption key, DER-encoded.
2.16.840.1.114227.1.1.2	Encryption public key algorithm. The value MUST be "RSA" Unicode string without NULL terminator.
2.16.840.1.114227.1.1.3	Encryption algorithm. The value MUST be "RSA" Unicode string without NULL terminator.

These certificates are self-signed, with the issuer and subject being the same, and validity set to 100 years. Because they are self-signed, the out-of-band certificate exchange MUST happen as part of the bootstrapping to establish trusted relationship.

The following is a list of fields and values for a sample relay server SOAP certificate:

```
X509 Certificate:
Version: 3
Serial Number: 6b8810e2a84bbf42a5271f1892cd681810121da9
Signature Algorithm:
  Algorithm ObjectID: 1.2.840.113549.1.1.5 sha1RSA
  Algorithm Parameters:
```

```

05 00
Issuer:
CN=http://relay.contoso.com

NotBefore: 8/1/2007 9:27 AM
NotAfter: 8/1/2107 9:27 AM

Subject:
CN=http://relay.contoso.com

Public Key Algorithm:
Algorithm ObjectId: 1.2.840.113549.1.1.1 RSA
Algorithm Parameters:
05 00
Public Key Length: 2048 bits
Public Key: UnusedBits = 0
0000 30 82 01 08 02 82 01 01 00 af f2 ab 53 03 40 98
0010 11 fd 56 c0 1e c8 21 7d ef 17 a3 1b a8 69 48 eb
0020 fb 88 ae 30 32 68 b9 dd 50 62 a1 4d 55 ee 81 77
0030 77 bb f3 df 05 30 33 2d 5c 68 34 b2 a6 df 78 89
0040 8a 5b 37 8b da 6a 92 f7 5e 03 a2 9a d8 d6 6b dc
0050 02 f4 26 42 ef de da 87 09 fb d0 f6 dd 18 5d 0f
0060 cb a0 f4 aa 46 b4 7c 03 9f 10 4f 7a 70 91 49 ae
0070 6b 4a f3 58 f3 a7 05 0b 8e 38 e4 71 35 98 a5 43
0080 d3 d1 e4 32 c4 24 35 c2 1b e0 9b 26 94 65 ec b6
0090 a4 f8 41 35 3a a1 bc e8 e9 0a 6d 18 13 8d de c9
00a0 74 6f 90 67 7f 54 fe b2 d6 9c 85 5f d9 dd 13 c8
00b0 a3 06 d2 0d ec 42 8a 47 62 d2 70 ea 2e f5 f6 d8
00c0 79 3c 70 84 ce 8c 47 83 60 2a 3c 70 35 4b e7 9d
00d0 59 bb 07 14 57 dd 4d fb d5 d7 f1 51 94 9d 76 dd
00e0 ad 00 ff bf 65 17 a6 71 28 6c 85 f6 77 79 75 37
00f0 ef a0 cb 22 95 47 1f c0 d7 6b 00 cf c4 1b c8 42
0100 a2 d1 92 d7 7e 40 bb 8a 75 02 01 11

Certificate Extensions: 3
2.16.840.1.114227.1.1.1: Flags = 0, Length = 10c
Unknown Extension type

0000 30 82 01 08 02 82 01 01 00 8c f9 af 9f e6 56 b4 0.....V.
0010 bc d0 1a b4 4d e6 11 a1 47 f0 c1 ec 66 c9 78 64 ....M...G...f.xd
0020 bb b2 64 70 c2 b7 2b 85 1c ee 87 76 4b 9f 86 38 ..dp..+....vK..8
0030 d4 f8 60 1b 6b d2 ed d5 80 17 3c f3 cf da e0 b2 ..`.k.....<....
0040 36 d3 e5 e5 60 41 8c 29 e2 ff 4e f6 d1 0c 23 a2 6...`A.)..N...#.
0050 69 2b 58 f8 dd fd a5 92 e7 2c 5d db 80 53 a1 7d i+X.....,]..S.)
0060 10 05 17 41 60 75 6c 1d c9 93 a9 fc e2 a2 75 07 ...A`ul.....u.
0070 83 cb 52 87 c2 68 11 e3 f8 af 03 75 28 77 d8 8e ..R..h.....u(w..
0080 14 e9 2c f9 81 58 7e d9 1c a7 f8 9c 59 f1 99 6d ...X~.....Y..m
0090 e7 b1 50 4e e4 24 b1 fa bf b9 d6 eb a0 d6 ba c9 ..PN.$.....
00a0 44 7e 47 67 79 74 df 84 1b 5f ab 4c 37 6d 75 87 D~Ggyt..._.L7mu.
00b0 84 1f 65 31 eb 43 c4 d0 a6 42 11 f4 90 92 c3 c5 ..e1.C...B.....
00c0 7b 7e 5f b8 09 fa 84 4a d5 cc 70 93 eb 39 c5 3d {~_....J..p..9.=
00d0 e2 7e 62 49 be 9b a7 6e 81 f6 9f 64 ab c6 c2 bf ..~bI...n...d....
00e0 18 c0 c0 d4 a8 85 68 3f eb 82 64 37 35 ab 53 11 .....h?...d75.S.
00f0 0e 74 7d 5d 3e f5 02 d1 65 d3 db 88 37 21 85 02 .t}]>...e...7!..
0100 85 ca 38 fe b3 c0 34 c7 69 02 01 11 ..8...4.i...

2.16.840.1.114227.1.1.2: Flags = 0, Length = 6
Unknown Extension type

0000 52 00 53 00 41 00 R.S.A.

```

2.16.840.1.114227.1.1.3: Flags = 0, Length = 6
Unknown Extension type

0000 52 00 53 00 41 00

R.S.A.

Signature Algorithm:

Algorithm ObjectId: 1.2.840.113549.1.1.5 sha1RSA

Algorithm Parameters:

05 00

Signature: UnusedBits=0

0000 06 54 6f 5c b1 cd 31 b9 c6 9e 52 5e 0b 81 f1 83
0010 96 9c f1 6e 65 40 01 f8 72 2b 79 4f 13 bd d1 41
0020 88 03 69 28 09 c3 a4 e4 d9 9c 2a 4c 1a cb ee ca
0030 3a 9b 50 0b 72 19 04 51 61 0c fc 6d 9c f0 a5 d3
0040 6d c5 68 50 0f 35 14 95 e9 af 66 e5 eb 1a 30 26
0050 0a 62 2b 59 a1 9e 76 3d 14 b9 f0 86 82 e5 22 73
0060 79 e5 b2 6b 9a 4e 26 2c 46 3f 3e 39 45 7c 56 d2
0070 9e d9 6a 3e 8a 15 74 d4 b9 5a dc 8f 32 25 ef fd
0080 27 77 7a 9e 04 10 5f 1c e3 36 39 87 4e 3e 16 b3
0090 61 35 8e 11 1b b1 1f f5 7e a8 dd 1c 03 40 f8 7e
00a0 e4 ac 8b f7 c6 2c 8e 98 24 f7 1a 40 34 62 a2 00
00b0 cb 97 10 ad 56 be a6 33 66 83 7a 72 94 ce c4 bc
00c0 9c da 34 ec f9 f9 f5 2f ed 2c 20 9d 30 3d 07 db
00d0 80 e4 f4 e9 f5 ad ef 49 ff 9e 2e 23 a4 97 e2 57
00e0 66 18 98 2c c9 15 30 54 26 7f bd 02 8c c5 5f 91
00f0 10 b2 b0 19 39 8a 1d b3 f1 2d 44 1f 91 d3 fa aa

Signature matches Public Key

Root Certificate: Subject matches Issuer

Key Id Hash(sha1): 63 43 bd d3 bc a4 85 17 b9 df e5 a1 1a f1 ff 97 5c f5 f0 d9

Cert Hash(md5): 17 1e 16 3d b9 cc 95 6e f3 88 27 f7 7e fe bf fe

Cert Hash(sha1): 15 97 70 c8 e6 96 e3 a2 39 9d a5 e6 75 d6 7d ee ff 56 2a d0

3.2.4 Message Processing Events and Sequencing Rules

The following table is a list of all operations.

Operation	Description
accountModify	This operation enables or disables relay services for users.
Registration	This operation is a mechanism for exchanging the shared secret key used in message data encryption/decryption and message integrity protection and verification.
RelayQuiescent	This operation sets the server to active or inactive states.
RelayDefault	This operation changes the server's default service settings.
userAdd	This operation adds users to the server for services.
userPurge	This operation purges user data on the server without deleting the users.

3.2.4.1 Registration

The client **MUST** register with the server before making other service requests. If a service request is made before the registration, a fault response message **MUST** be returned with the fault code = 304, indicating that the registration is required.

The **Registration** message is a mechanism for exchanging the shared secret key used in message data encryption/decryption and message integrity protection and verification. The client **MUST** generate the shared secret key for the registration.

3.2.4.1.1 Registration Request

The **Registration** request element is specified in section [2.2.3.4](#). Section [3.1.1.1](#) provides a detailed request message template. The following is a detailed payload data template for the **Registration** request message:

The `[- PayloadData -]` in the request template **MUST** use the following template:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="[- URL -]" Method="Registration">
    <g:SE EncryptedKey="[- EncryptedKey -]">
      <g:Cert
        EPKAlgo="RSA"
        EPubKey="[- EPubKey -]"
        EncAlgo="RSA"
        SPKAlgo="RSA"
        SPubKey="[- SPubKey -]"
        SigAlgo="RSA"
      />
      <g:Enc EC="[- ECData -]" IV="[- IV -]"/>
      <g:Auth Sig="[- Sig -]"/>
    </g:SE>
  </Payload>
</g:fragment>
```

`[- URL -]`: The management server URL, specified as a string.

`[- EncryptedKey -]`: The encrypted and base64 encoded shared secret key.

`[- EPubKey -]`: The encryption public key, DER-encoded then base64 encoded.

`[- SPubKey -]`: Signature public key, DER-encoded then base64 encoded.

`[- ECData -]`: The encrypted and base64 encoded data. It **MUST** use the following template before encryption:

```
<?xml version='1.0'?><?groove.net version='1.0'?><Payload/>
```

`[- IV -]`: The initialization vector for the encryption and decryption, base64 encoded.

`[- Sig -]`: The message signature, base64 encoded.

3.2.4.1.2 Registration Response

If no faults occur during the operation, the server **MUST** return the **Registration** response element specified in section [2.2.3.5](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server **MUST** return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template. The server **MUST** return a fault response message if the registration is not successful for any reason.

For a non-fault registration response, the `[- ECDATA -]` in the payload data template provided in section [3.1.1.3](#) MUST use the following template before encryption:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<Registration epoch="[- epoch -]">
  <Registration ErrorMessage="Success Registration." Status="0"/>
</Registration>
```

`[- epoch -]`: Indicates if the server's user database needs to be built or rebuilt, specified as an integer. A value of 0 indicates that the database MUST be built or rebuilt.

3.2.4.1.3 Registration Message Processing

To process the **Registration** message, the server MUST perform, in order, the procedures specified in sections [3.2.4.1.3.1](#) through [3.2.4.1.3.11](#).

3.2.4.1.3.1 Parse Incoming Envelope

Upon receiving a **Registration** message, the server MUST follow the request schema as defined in section [2.2.2](#) and **Registration** request as defined in section [2.2.3.3](#) to parse the message.

If any required element or attribute is missing, the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

The server MUST extract the fragment element, which is the base64-decoded content from the **Data** attribute of the **Payload** element from the service request, as defined in section [2.2.2.2.5](#). This **fragment** element is defined in section [2.2.2.1.2](#), and it contains secured content.

The server MUST read from the attribute **ManagementServer** from the **fragment** element. This value identifies the client.

3.2.4.1.3.2 Parse Secured elements

The secured element "g:SE" MUST be parsed as defined in section [2.2.3.4.1](#).

The following information MUST be saved as the client's security information:

- The encryption public key algorithm.
- The encryption public key, DER-encoded.
- The encryption algorithm.
- The signature public key algorithm.
- The signature public key, DER-encoded.
- The signature algorithm.

The following information MUST be saved as inputs to decryption:

- The encrypted serialized payload element.
- The initialization vector for the encrypted serialized payload element.

The following information MUST be saved as inputs to data integrity verification:

- The message signature.

Encrypted key MUST be saved as inputs to shared key decryption.

3.2.4.1.3.3 Decrypt Shared Key

Encrypted key MUST be decrypted using the server's encryption private key with RSA algorithm, as defined in [\[PKCS1\]](#). The result is saved as the shared key for the client and the server, indexed with the client name (the value of the **ManagementServer** attribute, as specified in section [3.2.4.1.3.1](#)).

3.2.4.1.3.4 Delete Encrypted Element and Authenticator Element

Encrypted element "g:Enc" and authenticator element "g:Auth" MUST be deleted as child objects of the secured element "g:SE", which is part of the fragment element.

Once this is done, the **fragment** element becomes a header element, as defined in section [3.1.2.2.1](#).

3.2.4.1.3.5 Serialize Header Element

Header element MUST be serialized as defined in section [3.1.2.2.2](#).

3.2.4.1.3.6 Decrypt Serialized Payload Element

Encrypted serialized **Payload** element MUST be decrypted using MARC4 algorithm (as defined in section [3.1.2.1](#)), with the shared key (section [3.2.4.1.3.3](#)).

Corresponding per-message initialization vector from the encrypted element MUST be used, as defined in section [3.2.4.1.3.2](#).

3.2.4.1.3.7 Compute Message Digest

Message digest MUST be computed using SHA1 algorithm, as defined in [\[RFC3174\]](#).

Message digest MUST include serialized header element (as defined in section [3.2.4.1.3.5](#)), and the decrypted content (the result from the procedure in section [3.2.4.1.3.6](#)) – in this order.

3.2.4.1.3.8 Verify Signature

Message signature MUST be verified by using the client's signature public key, as obtained in section [3.2.4.1.3.2](#), with RSA algorithm, as defined in [\[PKCS1\]](#).

Message signature MUST be computed with the message digest, as computed in section [3.2.4.1.3.7](#).

Comparison of this message signature with the one saved by the procedure in section [3.2.4.1.3.2](#) MUST be performed. If the signatures do not match, the data integrity and authenticity verification has failed. The message MUST be rejected. The server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

3.2.4.1.3.9 Authenticate Client

The server MUST compare the client certificate information obtained in section [3.2.4.1.3.2](#) with the certificate information imported as part of the initialization step using an out-of-band means. Specifically, the server MUST compare all the public keys and the corresponding algorithms.

If they do not match, the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

3.2.4.1.3.10 Prepare Registration Response

The server has successfully processed the **Registration** request message. The server MUST prepare a response element as defined in section [2.2.3.5.1](#), as a **Registration** element.

3.2.4.1.3.11 Secure Message, Prepare Envelope, Send

The server MUST follow the steps in section [3.1.2.2](#) to secure the response, prepare the response envelope and send.

3.2.4.2 Server Side Common Application Message Processing Rules

This section specifies the common processing rules that a server MUST perform when receiving a new request. This applies to all application messages (any message except for the **Registration** message) in sections [3.2.4.3](#) to [3.2.4.7](#). These sections omit the details of the detailed security processing because it is defined in this section.

When processing an application message, the server MUST perform the procedures, in order, specified in sections [3.2.4.2.1](#) through [3.2.4.2.5](#).

3.2.4.2.1 Parse Incoming Envelope and Get Shared Key

Upon receiving an application message, the server MUST follow the common request schema as defined in section [2.2.2](#) and the specific application message definition in section [2.2.3](#) to parse the message.

If any required element or attribute is missing, the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

The server MUST extract the **fragment** element, which is the base64-decoded content from the **Data** attribute of the **Payload** element from a service request, as defined in section [2.2.2.2.5](#). This **fragment** element is as defined in section [2.2.2.1.2](#), and it contains secured content.

The server MUST read from the attribute **ManagementServer** from the **fragment** element. This value identifies the client. The server MUST use this value to retrieve the shared key between the client and itself.

If the server does not have the key, the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code MUST be set to 304 to indicate that the client needs to register first. The fault string can be any string describing the error.

3.2.4.2.2 Open Secured Content

The server MUST take the **fragment** element and the shared key, and follow rules specified in section [3.1.2.3](#) to open the secured content to retrieve the application payload element.

If this step fails (data integrity check fails), the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

The application **Payload** element contains the original application request.

3.2.4.2.3 Process Application Request and Prepare Application Response

The server MUST process the application request and prepare the application response. The specific application processing rule is specified in the section for the specific message.

If the application request processing fails, the server MUST return a **Fault** message as defined in section [2.2.3.3](#). The fault code is defined in section [2.2.2.2.4](#) and can be any value in the fault code list except 304. The fault string can be any string describing the error.

3.2.4.2.4 Secure Message

The application response becomes the **Payload** element as defined in section [3.1.2.2.1](#).

The server MUST also prepare a **fragment** element with "fragment" name in the namespace "urn:groove.net". The **fragment** element MUST have one child element named "Payload". The server MUST set the **ManagementServer** and **Method** attributes for the **Payload** element. These values MUST match with those in the corresponding elements of the incoming request. The **Payload** element MUST have one child element named "SE" in the namespace of "urn:groove.net". The "g:SE" element MUST contain no content and is referred as secured element. This **fragment** element becomes the header element as defined in section [3.1.2.2.1](#).

With the **fragment** element, the application response, and the shared key, the server MUST follow the steps in section [3.1.2.2](#) to secure the application response and retrieve the serialized secured **fragment** element.

3.2.4.2.5 Prepare Envelope and Send

The server MUST follow the specific application message response envelope template (as defined in sections [3.1.2.2](#) and [2.2.3](#)) to prepare a response envelope, with the serialized secured payload element from section [3.2.4.2.4](#) as the value for the **Data** attribute for the **Payload** element of the **ServiceResponseType**, as defined in section [2.2.2.2.6](#).

The server then serializes the response envelope and sends it to the client.

3.2.4.3 RelayDefault

The RelayDefault message changes the server's default service settings.

3.2.4.3.1 RelayDefault Request

The **RelayDefault** request element is specified in section [2.2.3.6](#). Section [3.1.1.1](#) provides a detailed request message template.

3.2.4.3.2 RelayDefault Response

If no faults occur during the operation, the server MUST return the **RelayDefault** response element specified in section [2.2.3.7](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server MUST return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template.

3.2.4.4 RelayQuiescent

The **RelayQuiescent** message sets the server's state to active or inactive state. Before building/rebuilding the server's user database, the client MUST set the server to the inactive state. After completing the building/rebuilding, the client MUST set the server to the active state. See section [3.3.4.4](#) for details.

If the server is in the active state, the server MUST provide relay services to users. If the server is in the inactive state, the server MUST NOT provide services to users.

3.2.4.4.1 RelayQuiescent Request

The **RelayQuiescent** request element is specified in section [2.2.3.8](#). Section [3.1.1.1](#) provides a detailed request message template.

3.2.4.4.2 RelayQuiescent Response

If no faults occur during the operation, the server MUST return the **RelayQuiescent** response element specified in section [2.2.3.9](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server MUST return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template.

3.2.4.5 userAdd

The **userAdd** message adds users to the server's user database. The client sends **userAdd** messages to the server under two conditions:

1. The client is assigning one or more new users to the relay server.
2. The client is building or rebuilding the server's user database and is adding all users to the server's user database. See section [3.2.4.4](#) for details.

3.2.4.5.1 userAdd Request

The **userAdd** request element is specified in section [2.2.3.10](#). Section [3.1.1.1](#) provides a detailed request message template.

3.2.4.5.2 userAdd Response

If no faults occur during the **userAdd** operation, the server MUST return the **userAdd** response element specified in section [2.2.3.11](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server MUST return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template.

3.2.4.6 userPurge

The **userPurge** message purges user data from the server without deleting the users.

3.2.4.6.1 userPurge Request

The **userPurge** request element is specified in section [2.2.3.12](#). Section [3.1.1.1](#) provides a detailed request message template.

3.2.4.6.2 userPurge Response

If no faults occur during the **userPurge** operation, the server MUST return the **userPurge** response element specified in section [2.2.3.13](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server MUST return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template.

3.2.4.7 accountModify

The **accountModify** message enables or disables relay services for users. When a user is disabled, the server does not provide services to the user.

3.2.4.7.1 accountModify Request

The **accountModify** request element is specified in section [2.2.3.1](#). Section [3.1.1.1](#) provides a detailed request message template.

3.2.4.7.2 accountModify Response

If no faults occur during the **accountModify** operation, the server MUST return the **accountModify** response element specified in section [2.2.3.2](#). Section [3.1.1.2](#) provides a detailed response message template.

If faults occur during the operation, the server MUST return the **Fault** response specified in section [2.2.3.3](#). Section [3.1.1.2](#) provides a detailed **Fault** response message template.

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 Client Details

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The Server: The client maintains the server that it communicates with.

Users: The client manages the users of the server. Each managed user contains the following data:

- User ID: A GUID uniquely identifying a user.
- Enabled: Indicates if the user is enabled for the services provided by the server.

States: When communicating with the server, the client manages the following server states as specified in section [3.2.1](#):

- Unregistered
- Active state
- Inactive state

Client Key Pairs: Two sets of asymmetric public-private key pairs, one for encryption, and one for signing.

Client Certificate: The client creates a self-signed certificate containing the encryption and signature public keys.

Shared Secret Key: The client maintains the shared secret key established with the server through the Registration message.

Note that the preceding conceptual data can be implemented by using a variety of techniques. Any data structure that stores the preceding conceptual data can be used in the implementation.

3.3.2 Timers

None.

3.3.3 Initialization

For secure communication, in the initialization process, the client **MUST** generate client key pairs and the client SOAP certificate if they do not exist. Specifically, the following initialization steps **MUST** be performed:

1. If the client key pairs do not exist, two new sets of client key pairs, one for encryption and one for signing, **MUST** be generated and stored. They **MUST** be RSA 2048 bits long.
2. If the client certificate does not exist, a new client certificate **MUST** be generated in X509.v3 format, and DER encoded, with extensions for signature public key and related information. See section [3.2.3.1.2](#) for the definitions of the certificate extensions. The subject and issuer Common Name (CN) fields of the certificate are set to the management server hostname.
3. The certificate and management server identity information from the last step **MUST** be passed to the relay server through an out-of-band means.

3.3.4 Message Processing Events and Sequencing Rules

Section [3.2.4](#) lists all operations of the protocol.

3.3.4.1 Registration

Section [3.2.4.1](#) specifies the details of the Registration message. If the client receives a service fault response message with the fault code = 304 for any service request, the client **MUST** register with the server using the Registration request.

3.3.4.1.1 Registration Message Processing

To register with the server, the client MUST perform, in order, the procedures specified in sections [3.3.4.1.1.1](#) through [3.3.4.1.1.12](#).

3.3.4.1.1.1 Generate Shared Key and Prepare Payload and Secured elements

The client MUST generate a new shared key for the new **Registration** message. This MUST be a 160-bit MARC4 (as defined in section [3.1.2.1](#)) **symmetric key**.

The client MUST create an empty element named "Payload" as defined in the bottom of section [2.2.3.4.1](#). This element becomes the **Payload** element as defined in section [3.1.2.2.1](#). It MUST be serialized as defined in section [3.1.2.2.3](#). The result is a serialized **Payload** element.

The client MUST also create three empty elements, "fragment", "Payload", and "SE", all in the namespace "urn:groove.net". The client then MUST add "Payload" element as the child of "fragment" element, and "SE" element as the child of "Payload" element. The following attributes MUST be added to the "Payload" element:

- **ManagementServer** attribute is the name of the client
- **Method** attribute is the name of the request. It MUST have the value "Registration".

The "fragment" element is referred to as the **fragment** element, and the "SE" element is referred to as the secured element "g:SE" in the following sections within section [3.3.4.1.1](#).

3.3.4.1.1.2 Add Certificate Information

Certificate element "Cert" in the namespace of "urn:groove.net" MUST be created as a child of the secured element "g:SE". The following attributes that provide information about the client certificate MUST be added to the certificate element, as defined in section [2.2.3.4.1](#):

- **EPKAlgo** attribute is the encryption public key algorithm. This MUST be "RSA".
- **EPubKey** attribute is the encryption public key, DER-encoded.
- **EncAlgo** attribute is the encryption algorithm. This MUST be "RSA".
- **SPKAlgo** attribute is the signature public key algorithm. This MUST be "RSA".
- **SPubKey** attribute is the signature public key, DER-encoded.
- **SigAlgo** attribute is the signature algorithm. This MUST be "RSA".

3.3.4.1.1.3 Encrypt Shared Key and Add to Secured element

The newly generated shared key MUST be encrypted using the server's encryption public key with RSA Algorithm, as defined in [\[PKCS1\]](#). Corresponding server certificate information is imported in the initialization bootstrap step (section [1.5](#)).

The encrypted key MUST then be added as the value for the attribute **EncryptedKey** for the secured element "g:SE".

3.3.4.1.1.4 Serialize Fragment Element

The **fragment** element in section [3.3.4.1.1.1](#) becomes the header element as defined in section [3.1.2.2.1](#). It MUST be serialized as defined in section [3.1.2.2.2](#). The result is a serialized header element.

3.3.4.1.1.5 Compute Message Digest

Message digest MUST be computed using SHA1 algorithm, as defined in [\[RFC3174\]](#).

Message digest MUST include the serialized header element (as defined in section [3.3.4.1.1.4](#)) and the serialized **Payload** element (as defined in section [3.3.4.1.1.1](#)) – in this order.

3.3.4.1.1.6 Encrypt Serialized Payload Element and Add to Encrypted Element

Byte representation of the serialized **Payload** element (as defined in section [3.3.4.1.1.1](#)) MUST be encrypted using MARC4 algorithm (as defined in section [3.1.2.1](#)), with the shared key generated in section [3.3.4.1.1.1](#).

Unique initialization vector MUST be generated for each message, and used during encryption of the message.

An Encrypted element **Enc** in the namespace of "urn:groove.net" MUST be created as the child of the secured element "g:SE", with the following attributes:

- **EC** attribute MUST have a value equal to the encrypted serialized payload element.
- **IV** attribute MUST be the initialization vector used for the encryption and decryption.

3.3.4.1.1.7 Compute Signature and Add Authenticator Element

Message signature MUST be computed using the client's signature private key with RSA algorithm, as defined in [\[PKCS1\]](#).

Message signature MUST be for the message digest, as defined in section [3.3.4.1.1.5](#).

An authenticator element **Auth** in the namespace of "urn:groove.net" MUST be created as the child of the secured element "g:SE", with the following attribute:

- **Sig** attribute MUST have the signature computed here.

3.3.4.1.1.8 Serialize Header Element

The client MUST serialize the header element (the **fragment** element, now fully built up to match the schema in section [2.2.3.4.1](#)) as defined in section [3.1.2.2.2](#) to produce the serialized secured **Payload** element.

3.3.4.1.1.9 Prepare Envelope and Send

The client MUST follow the **Registration** request envelope template (as defined in sections [3.1.1.1](#) and [2.2.3.4](#)) to prepare a request envelope, with the serialized secured **Payload** element (section [3.1.2.2.2](#)) as the value for the **Data** attribute for the **Payload** element of the **ServiceResponseType**, as defined in section [2.2.2.6](#).

The client then serializes the request envelope and sends it to the server. It waits for the response.

3.3.4.1.1.10 Parse Incoming Response Envelope

Upon receiving the **Registration** response message, the client MUST first check whether it is a **Fault** message (as defined in section [2.2.2.2.4](#)).

If it is a **Fault** message, the client processing for the message stops.

For non-fault response, the client MUST follow the common response schema as defined in section [2.2.2](#) and the **Registration** response definition as defined in section [2.2.3.5](#) to parse the message.

If any required element or attribute is missing, the client MUST treat this as a bad response and the client processing for the message stops.

The client MUST extract the **fragment** element, which is the base64-decoded content from the **Data** attribute of the **Payload** element from a service response, as defined in section [2.2.2.2.6](#). This **fragment** element is defined in section [2.2.2.1.2](#), and it contains secured content.

3.3.4.1.1.11 Open Secured Content

The client MUST start with the **fragment** element and the shared key, and perform the procedures specified in section [3.1.2.3](#) to open the secured content to retrieve the application **Payload** element, which is the unencrypted application response.

If this step fails (data integrity check fails), the client MUST treat this as a bad response and the client processing for the message stops.

3.3.4.1.1.12 Post Registration Processing

The registration has succeeded. If there are any application messages waiting (because the server responds with 304 fault (see section [3.3.4.2.4](#)), the client SHOULD resend them now using the newly registered shared key.

3.3.4.2 Client Side Common Application Message Processing Rules

This section specifies the common processing rules that a client MUST perform when preparing a new request and processing a corresponding response. This applies to all application messages (any message except for the **Registration** message) in sections [3.2.4.3](#) through [3.2.4.7](#).

To process an application message, the client MUST perform, in order, the procedures specified in sections [3.3.4.2.1](#) through [3.3.4.2.5](#).

3.3.4.2.1 Prepare Application Request

The client MUST prepare the application request. The specific application processing rule is specified in the section for the specific message.

3.3.4.2.2 Secure Message

The content of the application request is contained in the **Payload** element as defined in section [3.1.2.2.1](#).

The client MUST also prepare a **fragment** element with "fragment" name in the namespace "urn:groove.net". The **fragment** element MUST have one child element named "Payload". The **Payload** element MUST have one child element named "SE" in the namespace of "urn:groove.net" and have its **ManagementServer** and **Method** attributes set. The "g:SE" element MUST contain no

content and is referred as secured element. This **fragment** element becomes the header element as defined in section [3.1.2.2.1](#).

The client retrieves the secret key it shares with the server.

With the **fragment** element, the application request and the shared key, the client MUST follow the steps in section [3.1.2.2](#) to secure the application request and produce the serialized secured **fragment** element.

3.3.4.2.3 Prepare Envelope and Send

The client MUST follow the specific application message request envelope template (as defined in sections [3.1.1.1](#) and [2.2.3](#)) to prepare a request envelope, with the serialized secured **Payload** element (section [3.3.4.2.2](#)) as the value for the **Data** attribute for the **Payload** element of the **ServiceResponseType**, as defined in section [2.2.2.2.6](#).

The client then serializes the request envelope and sends it to the server.

The client waits for the server's response.

3.3.4.2.4 Parse Incoming Response Envelope

Upon receiving the response message, the client MUST first check whether it is a **Fault** message (as defined in section [2.2.2.2.4](#)).

If it is a **Fault** message, the client MUST check whether the fault code is 304. If it is, the client MUST prepare a **Registration** message (as defined in section [3.2.4.1](#)) to send to the server. Upon successful registration, the client SHOULD retry the application message that just failed before registration.

For other fault messages, the client processing for the message stops.

For non-fault response, the client MUST follow the common response schema as defined in section [2.2.2](#) and the specific application message response definition as defined in section [2.2.3](#) to parse the response message.

If any required element or attribute is missing, the client MUST treat this as a bad response and the client processing for the message stops.

The client MUST extract the **fragment** element, which is the base64-decoded content from the **Data** attribute of the **Payload** element from a service response, as defined in section [2.2.2.2.6](#). This **fragment** element is defined in section [2.2.2.1.2](#), and it contains secured content.

3.3.4.2.5 Open Secured Content

The client MUST start with the fragment element and the shared key, and perform the procedures specified in section [3.1.2.3](#) to open the secured content to retrieve the application **Payload** element, which is the unencrypted application response.

If this step fails (data integrity check fails), the client MUST treat this as a bad response and the client processing for the message stops. The

The client MUST process the application response.

3.3.4.3 RelayDefault

Section [3.2.4.3](#) specifies the details of the **RelayDefault** message. The client SHOULD use this request to change the server's default service settings when needed.

3.3.4.4 RelayQuiescent

Section [3.2.4.4](#) specifies the details of the **RelayQuiescent** message. When the client receives any service response message with the **epoch** attribute set to 0, the client MUST build or rebuild the server's user database.

3.3.4.4.1 Build/Rebuild the Server's User Database

The following figure is a high level sequence diagram that illustrates the operation for building/rebuilding the server's user database.

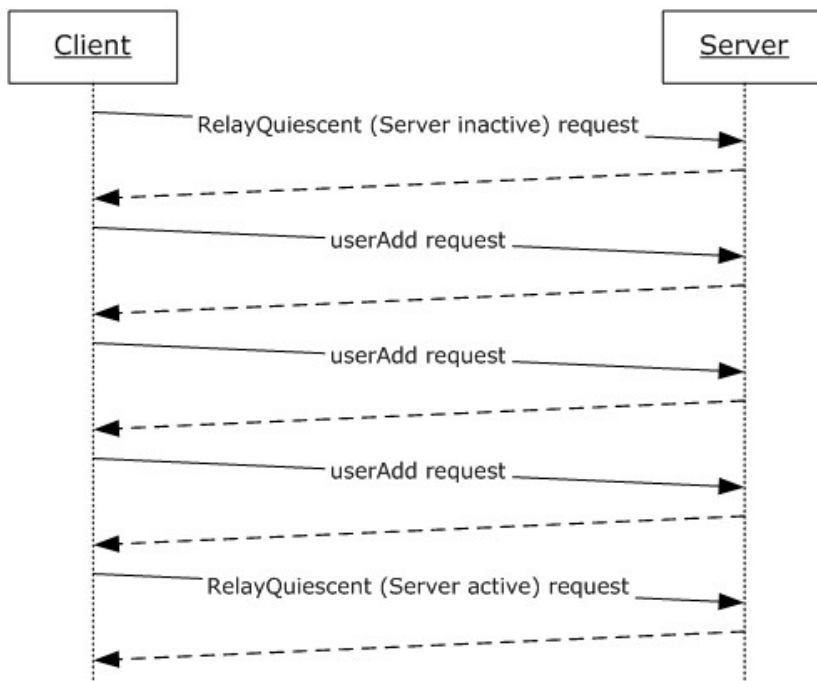


Figure 5: Building or rebuilding the server user database

Before building/rebuilding a server's user database, the client MUST set the server to the inactive state. Once the server is in the inactive state, the client uses one or more **userAdd** requests to add users to the server's user database. The client adds all users to the server's user database. After the server's user database has been built or rebuilt, the client MUST set the server to the active state.

3.3.4.5 userAdd

Section [3.2.4.5](#) specifies the details of the **userAdd** message.

3.3.4.6 userPurge

Section [3.2.4.6](#) specifies the details of the **userPurge** message.

3.3.4.7 **accountModify**

Section [3.2.4.7](#) specifies the details of the **accountModify** message. When a user is disabled on the server, the user does not receive relay services from the server.

3.3.5 **Timer Events**

None.

3.3.6 **Other Local Events**

None.

4 Protocol Examples

4.1 Serialization

The following is an example showing the result of serializing a secured **fragment** element as defined in section [3.1.2.2](#) (extra white spaces added for display purpose):

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="HostName1/gms.dll" Method="Registration">
    <g:SE>
      <g:Enc
EC="CnF8owJ2L/0hdYe+H3MMF175Doq6A7V+jB4salyupaloRVa5/+bu387bk3CQyz5g1fS2GD61oujJbMjpOuGw41ZtH
/tf0BtrQHGGJ15CAY9UjIxVgBf2nmi3xg0wqjZFTQxdqgb31ZQKsB+SqrByCu0sojyehedk2yH14ss3IdtQW4GDNAGrzYk
7Ac9E4dCoJ57irfRZicA==" IV="tWcSBZ2ThN5sAk2Hs2XKovupsAA=""/>
      <g:Auth MAC="IjE/R8ItYgetn75ecUzJ+gHcYxo=""/>
    </g:SE>
  </Payload>
</g:fragment>
```

4.2 Registration

Registration is required before any other service request. The following describes a typical registration scenario involving a client and a server.

1. The client sends a non-registration service request to the server.
2. The server returns a service **Fault** response message with a fault code of 304.

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultCode>304</faultCode>
      <faultString>GMS Registration Required. </faultString>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3. The client receives the service **Fault** response and sends a **Registration** request.

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <Registration>
      <Version xsi:type="xsd:int">1</Version>
      <Payload
data="PD94bWwgdmVyc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnpb249JzEuMCC/PjxnOmZyYWdtZW50
IHhtbG5zOmc9InVyb2pncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydGVyPSJU
ZXN0c2VydGVyL2dtcy5kbGwiIE1ldGhvZD0iUmVnaXN0cmF0aW9uIj48ZzpTRSBFbmnYexB0ZWRL
```

```
ZXk9Iks5Ndi9xeUhHZkR0d3g2NnU1K2R6K1NKRjRtSEQyRzN3MFZnd1U2SGNYVGxZNFp2eGJWOXUz
N21Vvm1rcTJrdDlyek9UZWqhEHNkcUluYm11aFdWWU1LVWhTdm5JT3FSTGdNaERtUmw1ZUJFM3lG
UXpuK0U0cFVEK0xUUE9mMUQxUEM5RFkwRGlTT015ci9tajBvQVNVKzBnOW94b2Z5ZGYxUndiMkFv
RitqcFhlRlV3ZXNKZkt6azcvdn1QR01GNTRsUC9ZS0o3cytrOWtvVzhlcHdzL3N6ZEFtb042SGVY
dDlQSzBKtGFPc0xzNERPZXRFVHRicldrNy9hSHBxVGhQelZTa3pvYlNsMjBXbFI1NW8xczVJK3Fz
VnlibHBSSEhVR1hqd0x6ZGhUYXBvY0N1UG1RNUJxWWNXYWY3bndnaVVjUEZrczNmNmVBOXQwQkNS
WjJLZz09Ij48ZzpDZXJ0IEVQS0FsZ289I1JTQSIgRVB1YktleT0iTu1JQkNBS0NBuUVBbEaOfk2
V2U1aFFRwTA5VThnOXRJZVRFRkxNkzBWVXNzSkvVStHDTThEcldaRVNaSW1NK0pUcG9DNGZtelJ4
anRfBWUxTkKvWW12U3FmWW5Nc3pRanNyUk41dW42OE15eHYvbG1QYitLSVPbNU4vaUF6K0dZVWpq
U0c4bVgzUmZsTDVEbnYwS1NNeHRwMT1PL09BYi9NT0FPb3U4Z2R1dTNPuVlMNGJkTDBVMG1RVExL
ak00cGhxcXZwbWZEVVZHeXY1QUpxdGVzQ1VnSEtIY1hlU0V3cXpvUE1pUTVuK0czSW5QZ1F2dDJZ
cDJkeVh0cH13Z3MyVDNZR1hEeEh6UjhyTW9vS3VkmNlWcnM4b3ZoK2FLbE5zSCs5bk5DZ1d6QWR1
M051VndBa0RBTH1IU0pYbktObm5WM0gzV3M1bEFFd3RvUmdSbVBNTENjYjNMOXBYTTRRSUJFUT09
IiBfBmNBbGdvpSJSU0EiIFNQs0FsZ289I1JTQSIgU1B1YktleT0iTu1JQkNBS0NBuUVBbEaOfk2
YktES2lhU25EWnp2OHdhbXhxL1pOM3JHeVpBVVZ0Z2NwV0F0bjJPazNZd2wyT2psV1VBQ1JSNFA1
WDFNbThiMUxWYUF4TW9IQndSL0g4YVFrSURmRmRtUUxhZEVKMnRtenR2RwDW3R4NUNNT01leGdY
aTFJR2JhSnQ5YXMIQXlnTTBkSWUyZlZGUfPBeWgzWW1QNnZ2L09tUCTaekFqWThIaHY4VEJ2eDFx
S0I1RktKT2RzYTBDBNTZjeldvS1c2TWFfiYldEcTJ0Z21OW1FqVmtLb1RjTndKYmE3U1pIOV1Zdmpv
VC9HU1JoL2VOLZntSnBRsWpFWj1IaFRoOGdpUnBUY1ZZMXdKZ1ZPVU1ZR2dzVndKZ1Q3S1AraWZD
NTZ6NKRkVX1DYkQ1R1NiaHdBNHFNL3RHbGdkZjgybkZLL3BrNkl6RG9McE9idW5VVUN3SUJFUT09
IiBtAwdBbGdvpSJSU0EiLz48ZzpFbmMgRUM9IkdWQ3N5b1lwUVFtTWtaQnNuQj1liR216R1VXMIvP
UmJkMGNhOTFiSG9xbEJPQ0EYyORSakZFMHkxMEGxSWo3WjJSeUxYZFJLb3JZTEF6Rxc9IiBJVj0i
bkdmVn1BtmhRdjZpak5EZfZoz1FnVHB4L05BPSIvPjxnOkF1dGggU2lnPSJ0e1YyNHFaYtJaSkd6
MmFaN2tnSmdkdjhyRkdNVUzsdEY3Y3ZQ1Z1TDNqNnoyeWVvkZkErYTVGTtVpNDU3a1FXTS9wSmtZ
M212VU82V0VOR05pbDc0Z0s3VThqRU44RUdvUHpvTDE2dEZSNDJhVtd5RmdYYTBnTWN3ZEgzZ25i
K0NHTk9aemNSSU9sU1ZRYUxLRUdZV29NVG0xZ01hYtdQT0JSbz1WWk8ySHhoM2dtV3g3dUhmBkZ1
Vmtrc2VaVHhQWUwZcitOYzV6TDNUamtCL3JOKy9aejRBSjZNd09XQWcweVpMZysyN3hQdGQ3S1Y5
M1Nhn2d0akhzRjgvsJFrSkU5Z29aM3UvUE01Z1AxVS9KwmlmNGg3MHHERMG81L1FRZ1U1d2JpCUtO
bu9YNVJINK5nRXFo3RzdUZMYmRGUVRcv1Z6UVZGdHRZTkZEaHRrWmw4Q1E9PSIvPjwvZzpTRT48
L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg==" xsi:type="binary"/>
</Registration>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The base64 encoded /Envelope/Body/Registration/Payload/@data contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="Registration">
    <g:SE
      EncryptedKey="NMv/qyHGfDtwx66u5+dz+SJF4mHD2G3w0VgwU6HcXTlY4ZvxbV9u37mUVmkq2kt9rzOTehjxsdqInbm
uhWVYIKUhSvnIOqRLGmHdMrl5eBE3yFQzn+E4pUD+LTPOf1D1PC9DY0DiSOIyr/mj0oASo+0g9oxofydf1Rwb2AoF+jpX
eFUwesJfKzk7/vyPGMF541P/YKJ7s+k9koW8epws/szdASoN6Hert9PK0JLaOsLY4DOetETtHsWk7/aHppqThPzVSkzobS
120WLR55o1s5I+qsVyb1pRHHUGXjwLzdhTapocCuPmQ5BqYcqaf7nngwiUcPFks3f6eA9t0BCRZ2ZKg==">
    <g:Cert EPKAlgo="RSA"
      EPubKey="MIIBCACAQEAALZ8Y6We5hQYQY09U8g9tIeTEFLg+0VUssJEU18CM8DrWZESZImM+JTpOC4fmzRxjtEme1NI/
YivSqfYnMsZQjsrRN5un68Myxv/liPb+KIZA5N/iAz+GYUjjSG8mX3Rf1L5Dnv0JSMxtp19O/OAb/MOAOou8gduu3QOYL
4bdL0U0mQTLKjM4phqqvVmfdUVGyv5AJqtesBUgHKHcXesEwqzoPMiQ5n+G3InPgQvt2Yp2dyXtpywgs2T3YGXDxHzR8r
MooKud2yVrs0vh+aKlNsH+9nNCfWzAde3NeVwAkDALyHSJXnKNnnV3H3Ws51AEwtoRgRmPMLCcb3L9pXM4QIBEQ=="
      EncAlgo="RSA" SPKAlgo="RSA"
      SPubKey="MIIBCACAQEAavkwk/4bKDKiaSnDzv8wamxq/ZN3rGyZAUvtgcpWAtn2Ok3Yw12Oj1VUABRR4P5X1Mm8b1LV
aAxMoHBwR/H8aQkIDfFdmQLadEJ2tmztvEgVWtx5CMOMuxgXi1IGbaJt9as5AygM0dIe2zVFPZayh3YmP6vv/OmP+ZzAj
Y8HhV8TBvx1qKB5FKJ0dsa0C56czWoKW6MabbWDq2tgMNZQjVvKoTcNwJba7RZH9YYvjOT/GSRh/eN/3mJpQIjEZ9HhTh
8giRpTbVY1wJgVOUMYGgsVwJgT7JP+ifC56z6DdUyCbD5GSbhwA4qm/tGlgdf82nFK/pk6IzDoLpObunUUCwIBEQ=="
      SigAlgo="RSA"/>
    <g:Enc
      EC="GpCsynYpQQmKzBsnB9bGizFUW2UiRbd0ca91bHoqlBOCA2CDRjFE0y10H1Ij7Z2RyLXdRkorYLAzEw="
      IV="nGfVyANhQv6ijNDdVhfQgTpx/NA="/>
```

```

    <g:Auth
      Sig="tzV24qZa2ZJGz2aZ7kgJgdv8XFGMUK3tF7cv6CVuL3j6z2yedfA+a5FM5i457kQWM/pJkY3mvUO6WENGNI174gK7
      U8jEN8EGoPzoLl6tFR42aU7yFgXa0gMcwDH3gnb+CGNOZzcRIOlSVQaLKEGYWoMTmlgIaa7POBRo9VZO2Hxh3gmWx7uHf
      nFuVkkseZTxjYL6r+Nc5zL3TjkB/rN+/Zz4AJ6MwOWAg0yZLg+27xPtd7JV93Sa7gtjHsF8/J1kJE9goZ3u/PM5fP1U/J
      Zif4h70q+0o5/QQfU5wbOqKNmOX5RH6NgEqhktsuFLbdfQTBVVzQVfttYNFDhtkZl8BQ==" />
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<Payload/>

```

- The server receives the **Registration** request. If the request is successful, the server returns a successful registration response.

```

<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <Registration>
      <Payload
        data="PD94bWwgdmVyc2lvdj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnNpb249JzEuMCC/PjxnOmZyYWdtZW50
        IHhtbG5zOmc9InVyb2pncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydGVyPSJUZXN0c2VydGVyL2
        dtcy5kbGwiIEl1dGhvdzD0iUmVnaXN0cmF0aW9uIj48ZzpTRT48ZzpFbmMgRUM9IkNuRjhvd0oyTC8waGRZZStI
        M01NRjE3NURvcTZBNlYrakI0c2FseXVwYXxvUlZhNS8rYnUzODdiazNDUXl6NWcxZlMyR0Q2MW91akpiTWpwT3
        VHdzQxWnRIL3RmMEJ0clFIR0oxNUNBWTlVakl4VmdCZjJubWkzeGcwZ3FqWkZUUXhkcWdiMzFaUUtZQitTcXJC
        eUNlMHNvanllaGvkaZJ5SDE0c3MzSWR0UVC0R0ROQUdyellrN0FjOUU0ZENvSjU3aXJmUlpY0E9PSIgSVY9In
        RXY1NCWjJUAe41c0FrMkhzMlhLb3ZlchNBQT0iLz48ZzpBdXR0IE1BQz0iSWpFL1I4SXRZZ2V0bjclZWNVekor
        Z0hjWxhvPSIvPjwvZzpTRT48L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg==" xsi:type="binary"/>
      </Registration>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/Registration/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="Registration">
    <g:SE>
      <g:Enc
        EC="CnF8owJ2L/0hdYe+H3MMF175Dog6A7V+jB4salyupaloRVa5/+bu387bk3CQyz5g1fS2GD61ouJbMjpOuGw41ztH
        /tf0BtrQHGGJ15CAY9UjIxVgBf2nmi3xg0wqjZFTQxdqgb3lZQKsB+SqrByCu0sojyehedk2yHl4ss3IdtQW4GDNAGrzYk
        7Ac9E4dCoJ57irfRZicA==" IV="tWcSBZ2ThN5sAk2Hs2XKovupsAA=" />
      <g:Auth MAC="IjE/R8ItYgetn75ecUzJ+gHcYxo=" />
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

- The client receives the **Registration** response. If the registration is successful, it can send other service requests.

When the value of the **epoch** attribute in a service response is 0, the client builds or rebuilds the server's user database. Before building a server's user database, the client sets the server to the inactive (quiescent) state. Once the server is in the inactive state, the client adds users to the server's user database. After the completion of adding users to the server's user database, the client sets the server to the active state.

- The client sends a **RelayQuiescent** request to the server to set the server to the inactive state.

The base64 encoded /Envelope/Body/RelayQuiescent/Payload/@data contains:

65 / 88

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<RelayQuiescent>
  <relay status="1"/>
</RelayQuiescent>
```

The value of the /RelayQuiescent/relay/@status is set to 1 to set the server to the inactive state.

- The server receives the **RelayQuiescent** request. If the request is successful, the server enters the inactive state and returns a successful **RelayQuiescent** response.

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <RelayQuiescent>
      <Payload
        data="PD94bWwgdGVyc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnpb249JzEuMCC/PjxnOmZyYWdtZW50
        IHhtbG5zOmc9InVybJpncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydGVyPSJUZXN0c2VydGVyL2
        dtcy5kbGwiIEl1dGhvdz0iUmVsYXlRdWllc2NlbnQiPjxnOlNFPjxnOkVuYyBFQz0ic3JKR1VlK0tKUlZxbmFS
        SkFUCGJZJzJE0VlRVcllWS1lxZzJNenVlNHRRSXpMUzY3WDlZUWRkb0Jpc1FBYXQxOEF6Yy9SazFQak5PK1k4bV
        B6YnpQb0QyV0xRd3dsTThTVUp5K2lBPT0iIElWPSJGOWwzU0FoTFNCR3FnRWRmK1VZVUxvQWZLRTQ9Ii8+PGc6
        QXV0aCBNQUM9IkpVb3lpUjNKRlFEMXRxdFhqMWZLYU1TS3hUdz0iLz48L2c6U0U+PC9QYXlsb2FkPjwvZzpmcm
        FnbWVudD4=" xsi:type="binary"/>
      </RelayQuiescent>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The base64 encoded /Envelope/Body/RelayQuiescent/Payload/@data contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="RelayQuiescent">
    <g:SE>
      <g:Enc
        EC="srJGUH+KJRVqnaRJATpbYf14VTUrYVKYqg2Mzue4tQIzLS67X9YQddoBisQAat18Azc/Rk1PjNO+Y8mPzbzPoD2WL
        QwwlM8SUJy+iA==" IV="F9l3SAhLSBGqgEdf+UYULUAFKE4="/>
      <g:Auth MAC="JUoyiR3JFQDltqtXj1fKaMSKxTw="/>
    </g:SE>
  </Payload>
</g:fragment>
```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<RelayQuiescent epoch="3"/>
```

- If the client receives the successful **RelayQuiescent** response, it can start to add users to the server's user database using the **userAdd** request. The **userAdd** operation can be used multiple times for a large user database.

```

<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <userAdd>
      <Version xsi:type="xsd:int">1</Version>
      <Payload
data="PD94bWwgdMvYc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnpb249JzEuMCc/PjxnOmZyYWdtZW50
IHhtbG5zOmc9InVybypncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydMvYPSJUZXR0c2VydMvYL2
dtcy5kbGwiIE1ldGhvZD0idXNlcFkZCI+PgC6U0U+PGC6RW5jIEVDPSJKVUhJY0JFMUZOTk1YUnZERGlNbJn2
dHB0MkJP0EMwR25nb1kvTDV0WVRZdngyRzI2QkxMbVNLQTJSb0loT0creVF0MWM4SG9xNTJ0UzE3TW5rK2ExSW
ZQV0grUXNPd3VmRzBtRlQ1bFFHNYthZjEvQ3RjNmZXV1NvU2J5OHNNxdTBGc2YyczRSZlE5SUV5bzhCTno5UDBR
bzIzbnRNMDFXV1kwYzI4R1daVTN0T1J2NUVob1RCcFNQbzA4K3c5VWVsWD1zUUNoUDJYmjhSFFYa3VHNDR2U1
ZQQV1Db2puZU5vTVhuK0N2aGV5My9kVmo2cFFzY1pGUHdlL2F6VlFvU09mUW1VYnNKczR4V2tPMFN6YnFHWXRr
VnZyMEDjRlNiVVFvFbEVteG5Db3lhdXpQM2IvdE0raXNlJMGQRmYsveW0wR3BKL3Nyczc5bVf4KzN6Nm0TnZtZE
o0cE45S1VlY2Y2OFpYM2xUMTdQd2M3cDF5WE00bHlxZVNrbnVyOW9Yc2U2Tm1ZPSIgSVY9Ik94dy9WenlCTExZ
dzR3cU1Mdm1SzZWZklMOD0iLz48ZzpbBdXR0IE1BQz0iSmVsaDV3TzRtbFI3R3ZmcEdBNVfVYXN3NERvPSIvPj
wvZzptRT48L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg==" xsi:type="binary"/>
      </userAdd>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/userAdd/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="userAdd">
    <g:SE>
      <g:Enc
EC="JUHIcBE1FNNMXRvDDiMn3vtpt2B08C0GngnY/L5tYTYvx2G26BLLmsKA2RoIhOG+yQt1c8Hoq52tS17Mnk+a1IfPW
H+QsOwufG0mFT5lQG7+af1/Ctc6fWWSoSby8sWu0Fs2s4RgQ9IEyo8BNz9P0Qo23ndM01WWY0c28GWZU3tORv5EhnTBp
SPo08+w9UelX9sQChP2X28aHGXkuG44vRVPAYCojneNoMXn+Cvhey3/dVj6pQscZFPwu/azVQoSOQfQiUbsJs4xWk00Szb
qGYtkVvr0GcFSbTUElEmxnCoyauzP3b/tM+isI0d+3+/ym0GpJ/srs79mQx+3z734NvmdJ4pN9KUecf68ZX3lT17Pwc7p
lyXJ4lyqeSQnur9oXse6NmY=" IV="Oxw/VzyBLLYw4wqILvmuK6VfML8="/>
      <g:Auth MAC="Jelh5wO4mlR7GvfpGA5Qoasw4Do="/>
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<userAdd rowCount="4">
  <user userId="0C68C31C-7EB1-43DA-8B20-B9CF1F604EB0"/>
  <user userId="05A8DAC9-28EA-457B-8A2E-E1939A72D9B1"/>
  <user userId="1AD36E9A-DDC5-4368-89F5-4C9F9FC1150D"/>
  <user userId="695ACC95-1E8A-46C2-ACF0-A6AC3E63E024"/>
</userAdd>

```

- The server receives the **userAdd** request. If the request is successful, the server adds the users to its user database and returns a successful **userAdd** response.

```

<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

```

```

xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Body>
  <userAdd>
    <Payload
      data="PD94bWwgdmVyc2lvcj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnNpb249JzEuMCC/PjxnOmZyYWdtZW50
      IHhtbG5zOmc9InVybjpncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydMvYPSJUZXXN0c2VydMvYL2
      dtcy5kbGwiIE1ldGhvZD0idXN1ckFkZCI+PGc6U0U+PGc6RW5jIEVDPStJFSjJyQnE5cGltL1dBZ1h6SEhCeH1x
      TXNoU21qWgtSTFTbUx1WVNpV0hLWWh6M1NzTlJkbDJOeXUvZ2dIVFQ4NVRxd0JaTtg5eTnueHFrTlRXTVZNMI
      ZuOXR4UiIgSVY9Im5DcjRuWk92WE1PYlJlcHI1dkEvK2xuYVh3ST0iLz48ZzpbBdXR0IE1BQz0icmVNdXI1bXlT
      c3pCanlqbW5QdkZqWwXIVZHU4PSIvPjwvZzptRT48L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg=="
      xsi:type="binary"/>
    </userAdd>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/userAdd/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="userAdd">
    <g:SE>
      <g:Enc
        EC="EJ2rBq9pim/WAgXzHHBxyqMshSmjXkqI1SmLuYciWHKYhz2SsNRdl2Nyu/ggHTT85TqwBZM89y3nxqkNTqMVM3Vn9
        txR" IV="nCr4nZOVXMOBRupr5vA/+lnaXwI="/>
      <g:Auth MAC="reMur5mySszBjyjmnPvFjYr/du8="/>
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<userAdd epoch="4"/>

```

- After the client has finished the **userAdd** request(s), it sends a **RelayQuiescent** to the server to set the server to the active state.

```

<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <RelayQuiescent>
      <Version xsi:type="xsd:int">1</Version>
      <Payload
        data="PD94bWwgdmVyc2lvcj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnNpb249JzEuMCC/PjxnOmZyYWdtZW50
        IHhtbG5zOmc9InVybjpncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydMvYPSJUZXXN0c2VydMvYL2
        dtcy5kbGwiIE1ldGhvZD0iUmVsYXlRdW1lc2NlbnQiPjxnOlNFPjxnOkVuYyBFQz0icUxUM29RbDlobW5xv21L
        UXVxe1RDQ1Oc29KMnhVQUJzQ3ZPRUVMb2ZJY1RGRHFyQTNWQmtrWWJ6Y2tldlQ0d29VUFQd1kzakRDMkdQUT
        JUUWRMaHi5Witsdm1uMFU3SStqOFF3VW8vNTE3eGo0WXgqZFFcnVuVWNXUnJONXB1WW1YOG5zPSIgSVY9Imd2

```

```

a0E2dk5KeU5mN3ZFwndpM2Z2VmgvSVZTZz0iLz48ZzpBdXR0IE1BQz0iQWdRWWM5Vlc3MjBpVHZVcnpzeHRrbW
13VXhZPSIvPjwvZzpTRT48L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg==" xsi:type="binary"/>
</RelayQuiescent>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/RelayQuiescent/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="RelayQuiescent">
    <g:SE>
      <g:Enc
EC="qLT3oQl9hmnqWmKQuqzTCCYNsoJ2xUABsCvOEELofIbTFDqra3VBkQYbzckevT4woUPQPwY3jDC2GPQ2TQdLhr9Z+
lvmn0U7I+j8QwUo/517xj4Yx1C1ErUnUcWRrN5pbYmX8ns=" IV="gvkA6vNJyNf7vEZwi3fvVh/IVSg="/>
      <g:Auth MAC="AgQYc9WW720iTvUrzsxtkmmwUxY="/>
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<RelayQuiescent>
  <relay status="0"/>
</RelayQuiescent>

```

The value of the /RelayQuiescent/relay/@status is set to 0 to set the server to the active state.

- The server receives the **RelayQuiescent** (end quiescent) request. If the request is successful, the server enters the active state and returns a successful **RelayQuiescent** response.

```

<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <RelayQuiescent>
      <Payload
data="PD94bWwgdMvYc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnpb249JzEuMCc/PjxnOmZyYWdtZW50
IHhtbG5zOmc9InVybjpncm9vdmUubmV0Ij44UGF5bG9hZCBNYW5hZ2VtZW50U2VydMvYPSJUZXXN0c2VydmVYl2
dtey5kbGwiIE1ldGhvZD0iUmVsYXlRdWl1c2NlbnQiPjxnOlNFPjxnOkVuYyBFQz0iN091M21JU0NrRkZHSjdj
bW50VjN1V4K1M3QnV2UWRSYjIqVmFoVjdxMUTwRlBjdDBUbjRJaTBoRV1HSFQzOWh2dXhDYUtqb3Q2UVZmTz
JJUnV6Wi8zaWNTTFRJRkZkTGt1A2WkRBPT0iIE1WPSJlcHlaT3FIM3o0UUNmTHdqUkd4R2o1QTBaNWs9Ii8+PGc6
QXV0aCBNQUM9ImM4dFdLMVpMS2d5VlUwZ0Vwa2xjZnZHNtZGND0iLz48L2c6U0U+PC9QYXlsb2FkPjwvZzpmcm
FnbWVudD4=" xsi:type="binary"/>
      </RelayQuiescent>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/RelayQuiescent/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>

```

```
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="RelayQuiescent">
    <g:SE>
      <g:Enc
EC="70u3mISckFFGHwcn0Z3EIKUx+S7BuvQdRb9jVahV7q1KpFPct0Tn4Ii0hEYGHt39hvuxCaKjot6QVfO2IRuzZ/3ic
SMTiFBNP6ZDA==" IV="epyZOqH3z4QCfLwjRGxGj5A0Z5k="/>
      <g:Auth MAC="JUoyiR3JFQD1tqtXj1fKaMSKxTw="/>
    </g:SE>
  </Payload>
</g:fragment>
```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<RelayQuiescent epoch="5"/>
```

4.4 Enable/Disable User Relay Service

The **accountModify** operation enables or disables relay services for users. The following are examples for enabling and disabling user relay services.

4.4.1 Disable User Relay Services

The following is an example for disabling relay services for users:

- The client sends an **accountModify** request to the server for disabling relay services for the specified users.

```
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <accountModify>
      <Version xsi:type="xsd:int">1</Version>
      <Payload
data="PD94bWwgdmVyc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZ1cnNpb249JzEuMCc/PjxnOmZyYWdtZW50
IHhtbG5zOmc9InVybjpncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydmlvPSJU
ZXN0c2VydmlvY2dtcy5kbGwiIE1ldGhvZD0iYWNjb3VudE1vZGlmeSI+PGc6U0U+PGc6RW5jIEVD
PSJnZTdQR24yVXJPeXpDcUF0d3ZGN0F1YnhoY3Y5dUNNdlpVRTlNV2Rudk5TaWhROUVDeEZRa29Z
SWVhaDIwOWhFemw5YjFpYj1URDRQMmt6QzRHZEkwN1FUbGQrVDJqNDBGUml5TyszbHlaQnVEVDU4
bnBGVFB3MWhRakZXMkZzQitYU0Y4bXh0RTVYZnJZUXU2RmVWUmX6TTNxi9ndTRwVXJkNzV5T250
TkhpMFRJSlFhamJmRj1kb3Q5a0wrykF3RksyK1FlLzJORjBWU1NYc2s4PSIgSVY9IkRpeENQK1I3
c1hVZ3FpSm90aWhldklhR3pWTT0iLz48ZzpbBdXRoIE1BQz0iVFFxbj1xd2hIdktnOWhlMVV4eWlh
dXg2OXA0PSIvPjwvZzptRT48L1BheWxvYWQ+PC9nOmZyYWdtZW50Pg==" xsi:type="binary"/>
      </accountModify>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The base64 encoded /Envelope/Body/accountModify/Payload/@data contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
```

```

    <Payload ManagementServer="Testserver/gms.dll" Method="accountModify">
      <g:SE>
        <g:Enc
EC="ge7PGn2UrOyzCqAtwvF7Aubxhcv9uCMwZUE9MWdnvNSihQ9ECxFQkoYIeah209hEz19b1ib9TD4P2kzC4GdI07QT1
d+T2j40FRb9O+3lyZBuDT58npFTPw1hQjFW2FsB+XSF8mxtE5XfrYQu6FeVRLzM3qB/gu4pUrd75yOnNNHi0TIJQajbFf
9dot9kL+bAwFK2+Qu/2NF0VSSXsk8=" IV="DixCP+R7sXUgqiJoNihuvIaGzVM="/>
        <g:Auth MAC="TQqn9qwhHvKg9he1Uxyiaux69p4="/>
      </g:SE>
    </Payload>
  </g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<accountModify rowCount="4">
  <user lockout="1" userId="0C68C31C-7EB1-43DA-8B20-B9CF1F604EB0"/>
  <user lockout="1" userId="05A8DAC9-28EA-457B-8A2E-E1939A72D9B1"/>
  <user lockout="1" userId="1AD36E9A-DDC5-4368-89F5-4C9F9FC1150D"/>
  <user lockout="1" userId="695ACC95-1E8A-46C2-ACF0-A6AC3E63E024"/>
</accountModify>

```

The value of the /accountModify/user/@lockout is set to 1 to disable relay services for the user.

The server receives the **accountModify** request. If the request is successful, the server disables relay services for the specified users and returns a successful **accountModify** response.

```

<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <accountModify>
      <Payload
data="PD94bWwgdMvYc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZlc3R0N3ZyVUxNQLJhM2tVK1NuMlFpbStoZnNpc01x
z0mc9InVybjpncm9vdmUubmV0Ij48UGF5bG9hZCByYW5hZ2VtZW50U2VydMvYPSJU
ZXN0c2VydMvYl2dscy5kbGwiIE1ldGhvZD0iYWYjY2VudE1vZGlmZSI+PGc6U0U+PGc6RW5jIEVD
PSJvWmdVcy9WahBSdEd3TTlwUktiR0o3MnFlc3R0N3ZyVUxNQLJhM2tVK1NuMlFpbStoZnNpc01x
QzZnaEd2Uv1JS1JWYzRBQXJ2QzZJdDkprRU0JqdmpsaUpwNVg5a08wYnhqUW4vIiBJVj0iU0ZlejKx
dDJrS1VtOW15aWk3VjNkR3djZmJrPSIvPjxnOkFlDGggTUFDPSIvN1ZndnBwNDA5MHRMU2V5WWYw
MFpwWENPNXM9Ii8+PC9n01NFPjwvUGF5bG9hZD48L2c6ZnJhZ211bnQ+" xsi:type="binary"/>
      </accountModify>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/accountModify/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="accountModify">
    <g:SE>
      <g:Enc
EC="oZgUs/VhpRtGwM9pRKbGJ72qestt7vrULMBRa3kU+Sn2Qim+hfsisMqC6ghGvQYIJRVc4AARvc2CjJQSBjvjliJp5
X9k00bxjQn/" IV="SFez91t2kKUm9iyii7V3dGwcfbk="/>
      <g:Auth MAC="/7VMvpp4090tLSeYf00ZpXC05s="/>
    </g:SE>
  </Payload>
</g:fragment>

```

```
</Payload>
</g:fragment>
```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<accountModify epoch="1"/>
```

4.4.2 Enable User Relay Services

The following is an example for enabling relay services for users:

- The client sends an **accountModify** request to the server for enabling relay services for the specified users.

```
<SOAP-ENV:Envelope>
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<SOAP-ENV:Body>
  <accountModify>
    <Version xsi:type="xsd:int">1</Version>
    <Payload
data="PD94bWwgdmVyc2l1bWJ0nMS4wJz8+PD9ncm9vdmUubmV0IHZlcnNpb249JzEuMCC/PjxnOmZyYWdtZW50
IHhtbG5zOmc9InVybWJpcncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydmVyPSJU
ZXN0c2VydmVyL2dtcy5kbGwiIElldGhvZD0iYWNjb3VudElvZGlmeSI+PGc6U0U+PGc6RW5jIEVD
PSJ0SFBA4UdWVmFlL0tmZGdtTGx4cVdqWkF0SWZlZDZkUzFvb2t1UEhxt2VKt0JmDGpONzlgYkVr
U2RuKzhUSDRNuitJak9kOC9LNGFrSlNwVldakZdpZE45RitlRUM0WUQxRWw6QmZVSnZOObkxEZWdC
ODcwTkZ4NfNiRfIlRlI2dDdSnFmOGkr04vMDNJTVpNRnNRY24ydxHdHw6URVp0VzRMDU5bkFn
SGNyRw4ZjBKYnNrYnNCMlFrTzlwalFRN3EyUU1DWGZ0N0tvWjZ3cG50R0huaVFUURkVWFVmuFox
NWl1eTB0Z1ExemtubXZpaTl4bm1HTC9zTzZ2Z2pML2p6Y0svMEhweDdBQm5RVlJ1dTR2dVkyM2hk
cnhmNVNhTVZtVCTdWd25rL1lPRm9tYkozN1hyNk1LMWJDbKE0UXZTUDNECDdoWTRpelZBUmlnMytw
L1ZMOGRhMw5JUVBjZjFQYyaEwzSHFLdVl6U1V3WTFwTHF4NlQwVXk4TEhJdDEzSnA0Q1dhbHVWVL2Nx
RlNzaU1hUTVG9yjdSQVn2TVp5K3dPTDRVQWwqQjI4SkdhZ1NMTFdrOHRlZCt0SFZKZkdWUUV5Rmha
M0RsR0g2c0J0eXRHMTdVlE9IiBJVj0iITnVPd3FmcGJDU3Z1NXJHMzV3ZlZWJmY2xFTUwPSivPjxn
OkF1dGggTUFDPStJoVkvJfOFlwCdF0TmExc0JSOU5uV1ZXdGhNY1U9Ii8+PC9nO1NFPjwvUGF5bG9h
ZD48L2c6ZnJhZ211bnQ+" xsi:type="binary"/>
  </accountModify>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The base64 encoded /Envelope/Body/accountModify/Payload/@data contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="accountModify">
    <g:SE>
      <g:Enc
EC="tHXPpGVVau/KfdgmLlxqWjZAtIfuuFdSlookKPHqOeJOBLtjN79fEbKsDn+8TH4MR+IjOd8/K4akJSpVWZ+7idN9F
+eEC4YD1EhzBfUjVnNLDegB870NFx4SbDR5GR6t7CJqf8i+sN/03IMZMFsQcn2uwGZOTEZtW4L059nAgHcrEgxf0Jbskb
sB2QkO9pKQQ7q2QMCXft7KoZ6wpntGHniQTQDdYQLPZ15b5y0tfQ1zknmvii9xniGL/sO6vgjL/jzcK/0Hpx7ABnQVRuu
4vuY23hdxrf5SaMvMT+Vwnk/YOFombJ37Xr6IKlBcNa4QvSP3Dp7hY4izVARiM3+p/VL8daInIQPIDt2hL3HqKuYzRUWY
1pLqx6T0Uy8LHIt13Jp4CwaluV/cqFSYiMaQ5Fb7RASvMZy+wOL4AljB28JGagSLLWQ8ted+hHVJfGpYeyPhZ3D1GH6s
BnYtG17UVQ=" IV="NoUwqgfbC5vu5rG35webfclEA50=">
```



```

    <g:Auth MAC="hVEc8YVp1tNalsBR9NnVVWthMcU="/>
  </g:SE>
</Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<accountModify rowCount="2">
  <user logout="0" userId="0C68C31C-7EB1-43DA-8B20-B9CF1F604EB0"/>
  <user logout="0" userId="05A8DAC9-28EA-457B-8A2E-E1939A72D9B1"/>
</accountModify>

```

The value of the /accountModify/user/@logout is set to 0 to enable relay services for the user.

- The server receives the **accountModify** request. If the request is successful, the server enables relay services for the specified users and returns a successful **accountModify** response.

```

<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
  ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
  <SOAP-ENV:Body>
    <accountModify>
      <Payload
        data="PD94bWwgdMvYc2lrbj0nMS4wJz8+PD9ncm9vdmUubmV0IHZ1cnNpb249JzEuMCC/PjxnOmZyYWdtZW50
        IHhtbG5zOmc9InVybjpncm9vdmUubmV0Ij48UGF5bG9hZCBNYW5hZ2VtZW50U2VydMvYPSJU
        ZXN0c2VydMvYl2dtcy5kbGwiIE1ldGhvZD0iYWNjb3VudElvZG1meSI+PGc6U0U+PGc6RW5jIEVD
        PSJ0anpndjFmUHErVzZtQk1PWFl4RU42SDRPWjEwSkhjSElVMk9jMXNabUV6Nm9CdjN3d3YleTFM
        aEVBMl1TcU4yWlNKTEJFb0VwNGpOcEJlWENFY0h5RDVLUlZwMXhPajQ4cGRDIiBjVj0iaW9sQjF1
        UUI4anFwSlJyVzBRQVo4cFRhUitNPSIvPjxnOkF1dGggTUFDPSJpRzAvRDlid2Y5TDFSDldrb3FM
        SEFaM0xUNDg9Ii8+PC9nOlNFPjwvUGF5bG9hZD48L2c6ZnJhZ2l1bnQ+" xsi:type="binary"/>
      </accountModify>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The base64 encoded /Envelope/Body/accountModify/Payload/@data contains:

```

<?xml version='1.0'?><?groove.net version='1.0'?>
<g:fragment xmlns:g="urn:groove.net">
  <Payload ManagementServer="Testserver/gms.dll" Method="accountModify">
    <g:SE>
      <g:Enc
        EC="tjzgv1fPq+W6mBMOXYxEN6H4OZ10JHcHIU2Oc1sZmEz6oBv3wvw5y1LhEA2YSqN2ZSJLBEoEp4jNpBuXCEcHyD5KR
        Vp1xOj48pdC" IV="iolBluQB8jqPJRrW0QAZ8pTaR+M="/>
      <g:Auth MAC="iG0/D9bwf9L1RwWkoqLHAZ3LT48="/>
    </g:SE>
  </Payload>
</g:fragment>

```

The encrypted /fragment/Payload/SE/Enc/@EC contains:

```
<?xml version='1.0'?><?groove.net version='1.0'?>  
<accountModify epoch="2"/>
```

5 Security

5.1 Security Considerations for Implementers

5.1.1 Use of semi-weak algorithms

The current protocol uses SHA1 and HMAC-SHA1 when computing message digest and keyed message digest.

While there are no known practical attacks against SHA1, there is evidence of weakness.

5.1.2 Use of weak algorithms

The current protocol uses MARC4 (RC4-drop(256)) for symmetric key encryption.

This encryption mechanism is considered somewhat weak at this point, because of problems with its key scheduling algorithm.

5.1.3 Insufficient encryption of protocol messages

The current protocol does not encrypt the message header.

This allows a passive attacker to read all the unencrypted data in the message header.

5.1.4 Use of the same key for encryption and MAC

The current protocol uses the same secret key for both encryption and integrity protection.

This opens the door for related key attacks.

5.1.5 Lack of nonce or sequence number to prevent replay attacks

The current protocol does not include a nonce or sequence number in each message to prevent replay attacks.

This allows an active attacker to replay previously captured messages.

5.1.6 Out-of-band Certificate Exchange

The protocol security depends on an out-of-band exchange of certificates of the two entities before any transaction can happen. Even though this is outside the scope of the protocol, a security consideration of this exchange is necessary to ensure that the parties in the exchange can authenticate each other. [<3>](#)

5.2 Index of Security Parameters

Security Parameter	Section
Management Server Encryption Public Key	1.3.1.2 , 2.2.3.4.1 , 3.2.3.1.2 , 3.2.4.1.1 , 3.2.4.1.3.2 ,

Security Parameter	Section
	3.2.4.1.3.9 , 3.3.1 , 3.3.3 , 3.3.4.1.1.2
Management Server Encryption Private Key	1.3.1.2 , 3.3.1 , 3.3.3
Management Server Signature Public Key	1.3.1.2 , 1.3.3 , 2.2.3.4.1 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.8 , 3.2.4.1.3.9 , 3.3.1 , 3.3.3 , 3.3.4.1.1.2
Management Server Signature Private Key	1.3.1.2 , 1.3.3 , 3.3.1 , 3.3.3 , 3.3.4.1.1.7
Relay Server Encryption Public Key	1.3.1.1 , 1.3.3 , 3.2.1 , 3.2.3 , 3.2.3.1.1 , 3.2.3.1.2 , 3.2.4.1.3.3 , 3.3.4.1.1.3
Relay Server Encryption Private Key	1.3.1.1 , 1.3.3 , 3.2.1 , 3.2.3
Relay Server Signature Public Key	1.3.1.1 , 3.2.1 , 3.2.3 , 3.2.3.1.1
Relay Server Signature Private Key	1.3.1.1 , 3.2.1 , 3.2.3
Secret key shared between the Management Server and the	1.3.2.1 ,

Security Parameter	Section
Relay Server	1.3.3 , 2.2.3.4.1 , 3.1.2 , 3.1.2.2.1 , 3.1.2.2.5 , 3.1.2.3.1 , 3.1.2.3.6 , 3.2.1 , 3.2.4.1 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.3 , 3.2.4.1.3.6 , 3.2.4.2.1 , 3.2.4.2.2 , 3.2.4.2.4 , 3.3.1 , 3.3.4.1.1.1 , 3.3.4.1.1.3 , 3.3.4.1.1.6 , 3.3.4.1.1.11 , 3.3.4.1.1.12 , 3.3.4.2.2 , 3.3.4.2.5
Secret key encryption algorithm	3.1.2.1 , 3.1.2.2.5 , 3.1.2.3.6 , 3.1.2.3.8 , 3.2.4.1.3.6 , 3.3.4.1.1.1 , 3.3.4.1.1.6 , 5.1.2
Public key encryption algorithm	1.3.1.1 , 1.3.1.2 , 2.2.3.4.1 , 3.2.3.1.2 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.3 , 3.2.4.1.3.9 , 3.3.4.1.1.2 , 3.3.4.1.1.3
Signature algorithm	1.3.1.1 , 1.3.1.2 , 2.2.3.4.1 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.8 , 3.2.4.1.3.9 , 3.3.4.1.1.2 , 3.3.4.1.1.7

Security Parameter	Section
Hash algorithm	3.1.2.2.4 , 3.1.2.3.7 , 3.2.4.1.3.7 , 3.3.4.1.1.5 , 5.1.1
HMAC algorithm	3.1.2.2.6 , 3.1.2.3.8 , 5.1.1
Initialization vector	2.2.2.1.2 , 2.2.2.3 , 2.2.2.3.6 , 2.2.3.4.1 , 3.1.1.3 , 3.1.2.1 , 3.1.2.2.5 , 3.1.2.2.7 , 3.1.2.3.2 , 3.1.2.3.6 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.6 , 3.3.4.1.1.6
Message signature	1.3.3 , 2.2.3.4.1 , 3.2.4.1.1 , 3.2.4.1.3.2 , 3.2.4.1.3.8 , 3.3.4.1.1.7
Message HMAC	1.3.3 , 3.1.2.2.6 , 3.1.2.3.8

6 Appendix A: Message Schemas

6.1 Request Message Schemas

```
<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import/>
  <xs:element name="Envelope">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Body">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:element ref="accountModify"/>
                <xs:element ref="Registration"/>
                <xs:element ref="RelayDefault"/>
                <xs:element ref="RelayQuiescent"/>
                <xs:element ref="userAdd"/>
                <xs:element ref="userPurge"/>
              </xs:choice>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>
```

The referenced child elements of the **Body** element are specified in the following schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/1999/XMLSchema-instance"/>

  <xs:element name="accountModify" type="ServiceRequestType"/>
  <xs:element name="Registration" type="ServiceRequestType"/>
  <xs:element name="RelayDefault" type="ServiceRequestType"/>
  <xs:element name="RelayQuiescent" type="ServiceRequestType"/>
  <xs:element name="userAdd" type="ServiceRequestType"/>
  <xs:element name="userPurge" type="ServiceRequestType"/>

  <xs:complexType name="ServiceRequestType">
    <xs:sequence>
      <xs:element name="Version" type="NumericType"/>
      <xs:element name="Payload" type="PayloadType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="NumericType">
```

```

<xs:simpleContent>
  <xs:extension base="xsd:int">
    <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      ref="xsi:type" user="required" fixed="xsd:int"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>

<xs:complexType name="PayloadType">
  <xs:attribute name="data" type="xs:base64Binary" use="required"/>
  <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    ref="xsi:type" user="required" fixed="binary"/>
</xs:complexType>
</xs:schema>

```

The referenced "xsi:type" is specified in the following schema:

```

<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="type" type="xs:string"/>
</xs:schema>

```

6.2 Response Message Schemas

```

<<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import/>
  <xs:element name="Envelope">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Body">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:element ref="accountModify"/>
                <xs:element ref="Registration"/>
                <xs:element ref="RelayDefault"/>
                <xs:element ref="RelayQuiescent"/>
                <xs:element ref="userAdd"/>
                <xs:element ref="userPurge"/>
              </xs:choice>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>

```


The referenced child elements of the **Body** element are specified in the following schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://www.w3.org/1999/XMLSchema-instance"/>

  <xs:element name="accountModify" type="ServiceResponseType"/>
  <xs:element name="Registration" type="ServiceResponseType"/>
  <xs:element name="RelayDefault" type="ServiceResponseType"/>
  <xs:element name="RelayQuiescent" type="ServiceResponseType"/>
  <xs:element name="userAdd" type="ServiceResponseType"/>
  <xs:element name="userPurge" type="ServiceResponseType"/>

  <xs:complexType name="ServiceResponseType">
    <xs:sequence>
      <xs:element name="Payload" type="PayloadType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="NumericType">
    <xs:simpleContent>
      <xs:extension base="xsd:int">
        <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
          ref="xsi:type" user="required" fixed="xsd:int"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="PayloadType">
    <xs:attribute name="data" type="xs:base64Binary" use="required"/>
    <xs:attribute xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      ref="xsi:type" user="required" fixed="binary"/>
  </xs:complexType>

</xs:schema>
```

The referenced "xsi:type" is specified in the following schema:

```
<xs:schema xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="type" type="xs:string"/>
</xs:schema>
```

The following specifies the service fault response message schema:

```
<xs:schema xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Envelope">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Body">
          <xs:complexType>
            <xs:sequence>
```

```

        <xs:element name="Fault" type="ServiceFaultResponseType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
<xs:attribute ref="SOAP-ENV:encodingStyle" use="required"/>
</xs:complexType>
</xs:element>

<xs:complexType name="ServiceFaultResponseType">
  <xs:sequence>
    <xs:element name="faultCode" type="xs:int"/>
    <xs:element name="faultString" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

  <xs:attribute name="encodingStyle" type="xs:string"/>
</xs:schema>

```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® Office 2010 suites
- Microsoft® Office Groove® 2007
- Microsoft® Office Groove® Server 2007
- Microsoft® Groove® Server 2010
- Microsoft® SharePoint® Workspace 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 2.2.2.3.2:](#) The Office Groove Server 2007 and Groove Server 2010 products provide a Manager component and a Relay component. The Manager component implements the management server functions of the protocol specified in this document, and is the client for the protocol. The Relay component implements the relay server functions of the protocol specified in this document, and is the server for the protocol

[<2> Section 3.2.3:](#) The Office Groove Server 2007 Manager and Groove Server 2010 Manager generate a registry file containing its certificate and identity information to support interoperability. The registry file uses the following template:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office Server\Groove\Groove
Relay\Parameters\ManagementServers\[[- hostname -]]\gms.dll]
"GUID"="[[- management server GUID -]]"
"Organization"="[[- management server organization name -]]"
"ServerURL"="http://[[- hostname -]]/gms.dll"
"Certificate"=hex:[[- certificate in hex -]]
```

[[- hostname -]]: The host name for the management server.

[[-management server GUID -]]: The unique ID that identifies this management server.

[[- management server organization name -]]: the organization name for the management server.

[[- certificate in hex -]]: The management server certificate in HEX format

[<3> Section 5.1.6:](#) In the Office Groove Server 2007 and Groove Server 2010 implementations, the requirement for a secure exchange mechanism for the certificate is done through the Management Server web administration user interface. The server administrator is responsible for downloading the client's certificate and uploading the relay's server identification XML file (which contains the

relay server's SOAP certificate). The administrator uses the Management Server HTTPS web interface to ensure that the correct management interface is being used.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

A

Abstract data model
 [client](#) 54
 [server](#) 42
accountModify
 [client](#) 61
 [server](#) 54
[accountModify request message](#) 24
[accountModify response message](#) 25
Adding users
 [overview](#) 11
[Applicability](#) 13
Application message processing rules
 [client](#) 58
 [server](#) 51

B

[Build relay server's user database example](#) 65

C

[Capability negotiation](#) 13
[Change tracking](#) 85
Changing server settings
 [overview](#) 11
Changing server states
 [overview](#) 11
Client
 [abstract data model](#) 54
 [initialization](#) 55
 [message processing](#) 55
 [other local events](#) 61
 [sequencing rules](#) 55
 [timer events](#) 61
 [timers](#) 55
Client bootstrapping
 [overview](#) 11
Client/server registration
 [overview](#) 11
Common attributes
 [messages](#) 22
Common elements
 [messages](#) 17
[Common Schema message](#) 14
[Common security processing rules](#) 38
Common types
 [messages](#) 20

D

Data model - abstract
 [client](#) 54
 [server](#) 42
Details
 [accountModify request message](#) 24
 [accountModify response message](#) 25
 [payload data template](#) 37

[Registration request message](#) 26
[Registration response message](#) 28
[RelayDefault request message](#) 29
[RelayDefault response message](#) 30
[RelayQuiescent request message](#) 31
[RelayQuiescent response message](#) 31
[request message template](#) 36
[response message template](#) 36
[userAdd request message](#) 32
[userAdd response message](#) 33
[userPurge request message](#) 33
[userPurge response message](#) 34

Details – client

[accountModify](#) 61
[application message processing rules](#) 58
[registration](#) 55
[RelayDefault](#) 60
[RelayQuiescent](#) 60
[userAdd](#) 60
[userPurge](#) 60

Details – security processing rules

[MARC4](#) 38
[open secured Payload element](#) 41
[secure and serialize message](#) 38

Details – server

[accountModify](#) 54
[application message processing rules](#) 51
[registration](#) 47
[RelayDefault](#) 52
[RelayQuiescent](#) 53
[userAdd](#) 53
[userPurge](#) 53

[Disable user relay services example](#) 70

E

[Enable user relay services example](#) 72
[Enable/disable user relay service example](#) 70
Enabling/disabling user relay services
 [overview](#) 12

Examples

[build relay server's user database](#) 65
[disable user relay services](#) 70
[enable user relay services](#) 72
[enable/disable user relay service](#) 70
[registration](#) 62
[serialization](#) 62

F

[Fields - vendor-extensible](#) 13

G

[Glossary](#) 8

I

[Index of security parameters](#) 75

[Informative references](#) 10
Initialization
 [client](#) 55
 [server](#) 44
[Introduction](#) 8

M

[MARC4 security processing rules](#) 38
[Message Definitions message](#) 24
Message processing
 [client](#) 55
 [server](#) 47
[Message syntax](#) 14
Message templates
 [payload data](#) 37
 [request](#) 36
 [response](#) 36
Messages
 [accountModify request](#) 24
 [accountModify response](#) 25
 [common attributes](#) 22
 [common elements](#) 17
 [Common Schema](#) 14
 [common types](#) 20
 [Message Definitions](#) 24
 [message syntax](#) 14
 [Namespaces](#) 14
 [Registration request](#) 26
 [Registration response](#) 28
 [RelayDefault request](#) 29
 [RelayDefault response](#) 30
 [RelayQuiescent request](#) 31
 [RelayQuiescent response](#) 31
 [transport](#) 14
 [userAdd request](#) 32
 [userAdd response](#) 33
 [userPurge request](#) 33
 [userPurge response](#) 34

N

[Namespaces message](#) 14
[Normative references](#) 9

O

[Open secured Payload element security processing rules](#) 41
Other local events
 [client](#) 61
 [server](#) 54
Out-of-band certificate exchange
 [overview](#) 11
[Overview \(synopsis\)](#) 10

P

[Parameters - security index](#) 75
[Payload data template](#) 37
[Preconditions](#) 13
[Prerequisites](#) 13

[Product behavior](#) 83
Protocol security
 [overview](#) 12
Purging user data
 [overview](#) 12

R

References
 [informative](#) 10
 [normative](#) 9
Registration
 [client](#) 55
 [server](#) 47
[Registration example](#) 62
[Registration request message](#) 26
[Registration response message](#) 28
[Relationship to other protocols](#) 12
RelayDefault
 [client](#) 60
 [server](#) 52
[RelayDefault request message](#) 29
[RelayDefault response message](#) 30
RelayQuiescent
 [client](#) 60
 [server](#) 53
[RelayQuiescent request message](#) 31
[RelayQuiescent response message](#) 31
[Request message schemas](#) 79
[Request message template](#) 36
[Response message schemas](#) 80
[Response message template](#) 36

S

Schemas
 [request message](#) 79
 [response message](#) 80
[Secure and serialize message security processing rules](#) 38
Security
 [parameter index](#) 75
Security processing rules
 [common](#) 38
Sequencing rules
 [client](#) 55
 [server](#) 47
[Serialization example](#) 62
Server
 [abstract data model](#) 42
 [initialization](#) 44
 [message processing](#) 47
 [other local events](#) 54
 [sequencing rules](#) 47
 [timer events](#) 54
 [timers](#) 44
Server bootstrapping
 [overview](#) 10
[Standards assignments](#) 13

T

Timer events
 [client](#) 61
 [server](#) 54
Timers
 [client](#) 55
 [server](#) 44
[Tracking changes](#) 85
[Transport](#) 14

U

userAdd
 [client](#) 60
 [server](#) 53
 [userAdd request message](#) 32
 [userAdd response message](#) 33
userPurge
 [client](#) 60
 [server](#) 53
 [userPurge request message](#) 33
 [userPurge response message](#) 34

V

[Vendor-extensible fields](#) 13
[Versioning](#) 13