[MS-GRVSSTPS]:

Simple Symmetric Transport Protocol (SSTP) Security Protocol

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability
6/27/2008	1.0	Major	Revised and edited the technical content
1/16/2009	1.01	Editorial	Revised and edited the technical content
7/13/2009	1.02	Major	Revised and edited the technical content
8/28/2009	1.03	Editorial	Revised and edited the technical content
11/6/2009	1.04	Editorial	Revised and edited the technical content
2/19/2010	2.0	Minor	Updated the technical content
3/31/2010	2.01	Editorial	Revised and edited the technical content
4/30/2010	2.02	Editorial	Revised and edited the technical content
6/7/2010	2.03	Editorial	Revised and edited the technical content
6/29/2010	2.04	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.05	Minor	Clarified the meaning of the technical content.
9/27/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
3/18/2011	3.0	Major	Significantly changed the technical content.
6/10/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
4/11/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	3.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2013	3.0	None	No changes to the meaning, language, or formatting of the technical content.
7/30/2013	3.0	None	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	3.0	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	3.0	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
4/30/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
6/23/2016	3.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies a security protocol used for client registration and authentication within the Simple Symmetrical Transmission Protocol (SSTP). SSTP Security is a sub protocol of the SSTP protocol.

SSTP Security is a block-oriented application-layer binary protocol designed so that a client and a relay server are mutually authenticated before a client retrieves data from a relay server. It provides a mechanism for a client and a relay server to securely exchange secret keys that are then used to authenticate each other through a simple challenge/response message sequence.

This SSTP Security protocol is embedded in the SSTP protocol – it relies on SSTP commands to transport its protocol messages. This protocol depends on and works only within SSTP.

Protocol data for SSTP Security is encoded as authentication tokens in several predefined SSTP commands: **Connect**, **ConnectResponse**, **ConnectAuthenticate**, **Register**, **RegisterResponse**, **Attach**, **AttachResponse** and **AttachAuthenticate**. Refer to [MS-GRVSSTP] for a complete specification of these SSTP commands.

SSTP Security is a protocol used only between a client and a relay server.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

account: A collection of data and settings for a SharePoint Workspace or Groove identity that represents a user. This includes shared spaces, messages, and preferences that are associated with a user's identity. An account can reside on one or more devices.

account key: A secret key that is shared between a relay server and a client account for **account** authentication (2). A protocol client generates this key when it creates a new account, and then registers this key on a relay server through a registration sequence. The relay server uses this key to authenticate the account and enable the protocol client to retrieve **identity-targeted messages** from the server.

account URL: A unique identifier for an **account**, as described in [RFC3986].

American National Standards Institute (ANSI) character set: A character set defined by a code page approved by the American National Standards Institute (ANSI). The term "ANSI" as used to signify Windows code pages is a historical reference and a misnomer that persists in the Windows community. The source of this misnomer stems from the fact that the Windows code page 1252 was originally based on an ANSI draft, which became International Organization for Standardization (ISO) Standard 8859-1 [ISO/IEC-8859-1]. In Windows, the ANSI character set can be any of the following code pages: 1252, 1250, 1251, 1253, 1254, 1255, 1256, 1257, 1258, 874, 932, 936, 949, or 950. For example, "ANSI application" is usually a reference to a non-**Unicode** or code-page-based application. Therefore, "ANSI character set" is often misused to refer to one of the character sets defined by a Windows code page that can be used as an active system code page; for example, character sets defined by code page 1252 or character sets defined by code page 950. Windows is now based on **Unicode**, so the use of ANSI character sets is strongly discouraged unless they are used to interoperate with legacy applications or legacy data.

ASN.1: Abstract Syntax Notation One. ASN.1 is used to describe Kerberos datagrams as a sequence of components, sent in messages. ASN.1 is described in the following specifications: [ITUX660] for general procedures; [ITUX680] for syntax specification, and [ITUX690] for the

Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and **Distinguished Encoding Rules (DER)** encoding rules.

certificate: A certificate is a collection of attributes (1) and extensions that can be stored persistently. The set of attributes in a certificate can vary depending on the intended usage of the certificate. A certificate securely binds a public key to the entity that holds the corresponding private key. A certificate is commonly used for authentication (2) and secure exchange of information on open networks, such as the Internet, extranets, and intranets. Certificates are digitally signed by the issuing certification authority (CA) and can be issued for a user, a computer, or a service. The most widely accepted format for certificates is defined by the ITU-T X.509 version 3 international standards. For more information about attributes and extensions, see [RFC3280] and [X509] sections 7 and 8.

challenge: A piece of data used to authenticate a user. Typically a challenge takes the form of a **nonce**.

connection: A link between two devices that uses the **Simple Symmetric Transport Protocol (SSTP)**. Each connection can support one or more SSTP sessions.

device: A client or server computer that uses a **device URL** to identify itself as an endpoint (5) for synchronizing account data.

device key: A secret key that is shared between a relay server and a client device for device authentication (2).

device URL: A unique identifier for a client device, as described in [RFC3986].

device-targeted message: A message with an intended destination of a specific resource handler, identity, and client device combination. A device-targeted message is sent over a session addressed by a tuple of resource URL, identity URL, and client device URL.

Distinguished Encoding Rules (DER): A method for encoding a data object based on Basic Encoding Rules (BER) encoding but with additional constraints. DER is used to encode X.509 **certificates** that need to be digitally signed or to have their signatures verified.

ElGamal encryption: A public-key encryption scheme, as described in [CRYPTO].

identity: A digital persona that is associated with two key pairs, one for encrypting data and another for signing data.

identity URL: A string of characters that uniquely identifies an identity and conforms to the syntax of a URI, as described in [RFC3986].

identity-targeted message: A message that is destined for a specific resource handler and identity combination, regardless of the client device. The message address includes a resource URL, identity URL, and client device URL, where the client device URL is empty.

keyed-hash Message Authentication Code: A symmetric keyed hashing algorithm used to verify the integrity of data to help ensure it has not been modified while in storage or transit.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

management server: A server application that is used to manage SharePoint Workspace and Groove identities and services.

Modified Alleged Rivest Cipher 4 (MARC4) algorithm: A variable, key-length, symmetric encryption algorithm that discards the first 256 bytes of a keystream.

network address translation (NAT): The process of converting between IP addresses used within an intranet, or other private network, and Internet IP addresses.

nonce: A number that is used only once. This is typically implemented as a random number large enough that the probability of number reuse is extremely small. A nonce is used in authentication protocols to prevent replay attacks. For more information, see [RFC2617].

object identifier (OID): In the context of a directory service, a number identifying an object class or attribute (2). Object identifiers are issued by the ITU and form a hierarchy. An OID is represented as a dotted decimal string (for example, "1.2.3.4"). For more information on OIDs, see [X660] and [RFC3280] Appendix A. OIDs are used to uniquely identify certificate templates available to the certification authority (CA). Within a **certificate**, OIDs are used to identify standard extensions, as described in [RFC3280] section 4.2.1.x, as well as non-standard extensions.

private key: One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data that has been encrypted with the corresponding public key. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

public key: One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a digital certificate. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

RC4: A variable key-length symmetric encryption algorithm. For more information, see [SCHNEIER] section 17.1.

relay server: A server application that provides message transmission services for **Simple Symmetric Transport Protocol (SSTP)** messages.

relay URL: A string of characters that uniquely identifies a relay server and conforms to the syntax of a URI, as described in [RFC3986].

secret key: A symmetric encryption key shared by two entities, such as between a user and the domain controller (DC), with a long lifetime. A password is a common example of a secret key. When used in a context that implies Kerberos only, a principal's secret key.

session: A unidirectional communication channel for a stream of messages that are addressed to one or more destinations. A destination is specified by a resource URL, an identity URL, and a device URL. More than one session can be multiplexed over a single connection.

SHA-1 hash: A hashing algorithm as specified in [FIPS180-2] that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

Simple Symmetric Transport Protocol (SSTP): A protocol that enables two applications to engage in bi-directional, asynchronous communication. SSTP supports multiple application endpoints (5) over a single network connection between client nodes.

symmetric key: A **secret key** used with a cryptographic symmetric algorithm. The key needs to be known to all communicating parties. For an introduction to this concept, see [CRYPTO] section 1.5.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

X.509: An ITU-T standard for public key infrastructure subsequently adapted by the IETF, as specified in [RFC3280].

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

[CRYPTO] Menezes, A., Vanstone, S., and Oorschot, P., "Handbook of Applied Cryptography", 1997, http://www.cacr.math.uwaterloo.ca/hac/

[MS-GRVDYNM] Microsoft Corporation, "Groove Dynamics Protocol".

[MS-GRVHENC] Microsoft Corporation, "HTTP Encapsulation of Simple Symmetric Transport Protocol (SSTP)".

[MS-GRVSPCM] Microsoft Corporation, "Client to Management Server Groove SOAP Protocol".

[MS-GRVSPMR] Microsoft Corporation, "Management Server to Relay Server Groove SOAP Protocol".

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

[PKCS1] RSA Laboratories, "PKCS #1: RSA Cryptography Standard", PKCS #1, Version 2.1, June 2002, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-rsa-cryptography-standard.htm

[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

[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

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

1.2.2 Informative References

[RFC3641] Legg, S., "Generic String Encoding Rules (GSER) for ASN.1 Types", RFC 3641, October 2003, http://www.rfc-editor.org/rfc/rfc3641.txt

1.3 Protocol Overview (Synopsis)

SSTP Security is a security protocol that is used for client registration and authentication with a **relay server** over SSTP [MS-GRVSSTP]. **Simple Symmetric Transport Protocol (SSTP)** is an application layer protocol that provides a full-duplex **connection** between two applications. It supports bidirectional, asynchronous communications for multiple endpoints within the applications. SSTP is used both for communications between two clients and for communications between a client and a server.

When a client sends application data to another client, it encrypts the data using a cryptographic key so that only the intended target can decrypt the data. When two clients communicate directly with each other, they authenticate each other at the application layer, and applications provide the security guarantees of message confidentiality and integrity [MS-GRVDYNM].

When clients cannot communicate with each other directly (for example, a client is offline or is unreachable because of firewalls or **network address translation (NAT) devices**), they route data to a relay server. A relay server provides the message transmission services and then is responsible for securely delivering data to targets. When a target client connects to a relay server, the relay server authenticates the client before forwarding any stored data. The data forwarding occurs only after a successful client authentication. SSTP Security is an application layer protocol designed for client registration and authentication between a client and a relay server.

SSTP Security is a sub protocol of the SSTP protocol and enables a relay server to authenticate connecting clients before forwarding application data to target clients. Protocol messages of SSTP Security are embedded in SSTP and are transported through a selected set of SSTP commands.

A client uses the SSTP Security protocol to register **secret keys** with its relay server for new devices and **accounts**. These secret keys are then used by the relay server to authenticate client devices and accounts. A relay server forwards **device-targeted messages** to a client only after it has successfully authenticated the client's device, and similarly, a relay server forwards **identity-targeted messages** to a client only after it has successfully authenticated a client account associated with the target **identity**. A successful authentication is valid only for the duration of the current SSTP connection. A client has to be authenticated for each and every new connection to its assigned relay server.

The SSTP Security protocol uses a simple **challenge**/response sequence to authenticate client devices and accounts. It is tied to the SSTP protocol because the SSTP protocol not only serves as the transport for the security protocol, but also provides mechanisms to carry security processing response codes and other critical information in order for the security protocol to function properly.

1.3.1 Client/Server Model

This protocol follows a typical client/server model where a client connects to a relay to send or retrieve messages, and a relay is a server that transmits and delivers messages to client targets. A client registers its **device key** and **account key** with a relay server for each new device and new account. These keys are treated as secrets that are shared only between a relay and a client entity (either device or account). When a connecting client sends a challenge to a relay server for authentication, the relay server looks up a secret key of the client, verifies the content of the challenge using the secret key, and then responds to the client with its own challenge or an error code. Only after a successful sequence of challenges and responses will the relay open an outbound SSTP **session**, and deliver any stored message to the authenticated client.

The SSTP Security protocol also provides an option for a client to pre-authenticate itself to a relay server. A **management server** is responsible for creating identities and assigning relays. When the management server creates an identity, it also generates a random identifier and passes it as a pre-authentication token to the assigned relay server to uniquely identify the user. When a user configures its account for an identity, it also receives the pre-authentication token from the management server. A client includes this pre-authentication token to identify itself when registering to its assigned relay server. The relay server verifies the token against the one that it has received earlier from the management manager. By verifying this pre-authentication token, the relay server ensures that the connecting client has a legitimate identity that has been created by the management server.

The following diagram shows a simplified view of interactions among a client, a relay server and a management server. Refer to the following protocols for details on communications among a client, a management server and a relay server:

- [MS-GRVSPMR] specifies a SOAP protocol used for communications between a management server and a relay server.
- [MS-GRVSPCM] specifies a protocol used for communications between a client and a management server.

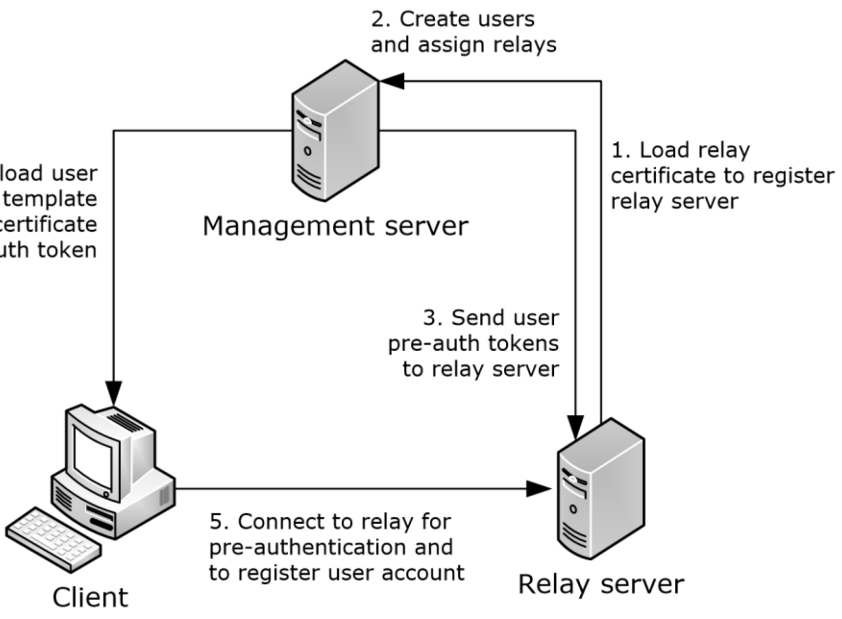
2. Create users and assign relays
1. Load relay certificate to register relay server
4. Download user identity template with relay certificate and pre-auth token
Management server
3. Send user pre-auth tokens to relay server
Client
5. Connect to relay for pre-authentication and to register user account
Relay server

Figure 1: Interactions between client, relay server, and management server

1.3.1.1 Client Role

The client uses a configuration code received from a management server to configure its identity and obtain relay assignments before it can connect to its assigned relay server for the first time. The management server also sends to the client the relay server's **certificate** and a pre-authentication token to be used during registrations to identify the client to the relay server.

A client creates asymmetric encryption and signature key pairs for the device that it resides on, and for each account on the device. The client exchanges public parts of these key pairs with a relay server during the device and account registration.

A client also maintains status about registrations and authentications of its accounts and device so that it is able to register just once for each combination of **device URL**, **account URL** and **relay URL**, and to authenticate once for each new connection to its assigned relay server. It is important for this protocol that the client saves the key pairs as well as secret keys. Redundant registration is permitted, but in general, is to be avoided for performance reasons. A client authenticates its device and each account only once for every SSTP connection to a relay server.

1.3.1.2 Server Role

A relay server maintains application data, user pre-authentication tokens, certificates and keys for devices and accounts. A server keeps track of authentication status on a per connection basis for each device and account, so that it delivers messages to clients only for correctly authenticated devices and accounts. For performance and security reasons, a relay server saves certificates and keys for devices and accounts in a persistent storage so that a client does not have to re-transmit the information each time it connects to the relay server, and the relay server can use stored secret keys to authenticate reconnecting clients.

1.3.2 Messages

Messages for SSTP Security are encoded as binary blocks and denoted as authentication tokens in a selected set of SSTP commands that have been defined to transport security messages. Each security message starts with a header followed by message data. The header contains the protocol's major version and minor version numbers, and a numerical message identifier that is unique under the scope of the transporting SSTP command. The major and minor version numbers of the SSTP Security protocol are separate from those of the SSTP protocol. The message identifier maps to a specific format and enables the message data to be parsed correctly.

Messages in the SSTP Security protocol are paired up with the transporting SSTP commands, and can be classified into three categories: device authentication messages, account authentication messages and registration messages. See section 3 for details on which SSTP commands and security messages belong to each of the three categories.

1.3.3 Typical Message Sequences

The following subsections provide an overview of the typical message exchange sequences between a client and a server when the SSTP Security protocol is used for client registration and authentications.

1.3.3.1 Registrations

The following diagram shows a typical message exchange sequence for registering a client with a relay server. The client and server first exchange SSTP commands to establish an SSTP connection. Then the client sends a security message embedded in an SSTP command to register with the relay server. The security message contains the following:

- Secret key encrypted using the encryption **public key** from the relay server certificate
- Challenge encrypted using the secret key
- Other cryptographic information such as message signature for integrity protection

The server verifies that the message is intended for itself, decrypts the secret key using its encryption **private key**, and uses the secret key to decrypt the challenge, and then responds with a security message that includes the client's challenge and a counter challenge that the server has encrypted using the secret key. The client verifies the response message from the server, and then can authenticate itself to the relay server.

In the subsequent diagrams, security messages are shown as being embedded in SSTP commands by [] brackets.

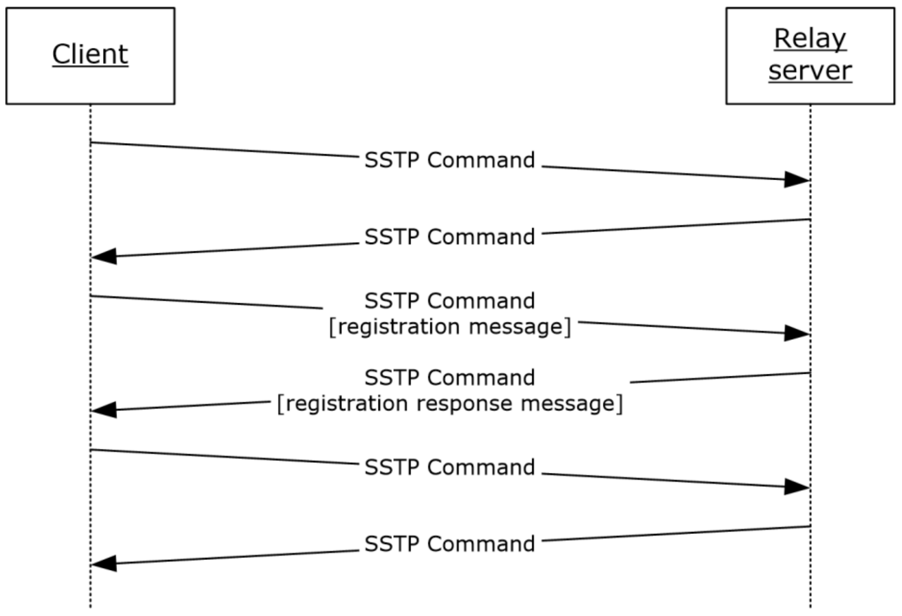
Client
Relay server
SSTP Command
SSTP Command
SSTP Command [registration message]
SSTP Command [registration response message]
SSTP Command
SSTP Command

Figure 2: Client registration message sequence

1.3.3.2 Authentications

The following diagram shows a typical message exchange sequence for a mutual authentication between a client and a server. The client has previously registered with the relay server by exchanging a secret key between the client and the server. The client first sends a challenge encrypted using the secret key to authenticate itself to the relay server. The server decrypts the challenge using the secret key, and then responds with a security message that includes the client’s challenge and a counter challenge that the server has encrypted using the secret key. The client verifies the response against its initial challenge, decrypts the server’s challenge using the shared secret key, and then sends a message with the decrypted server challenge to complete the authentication sequence.

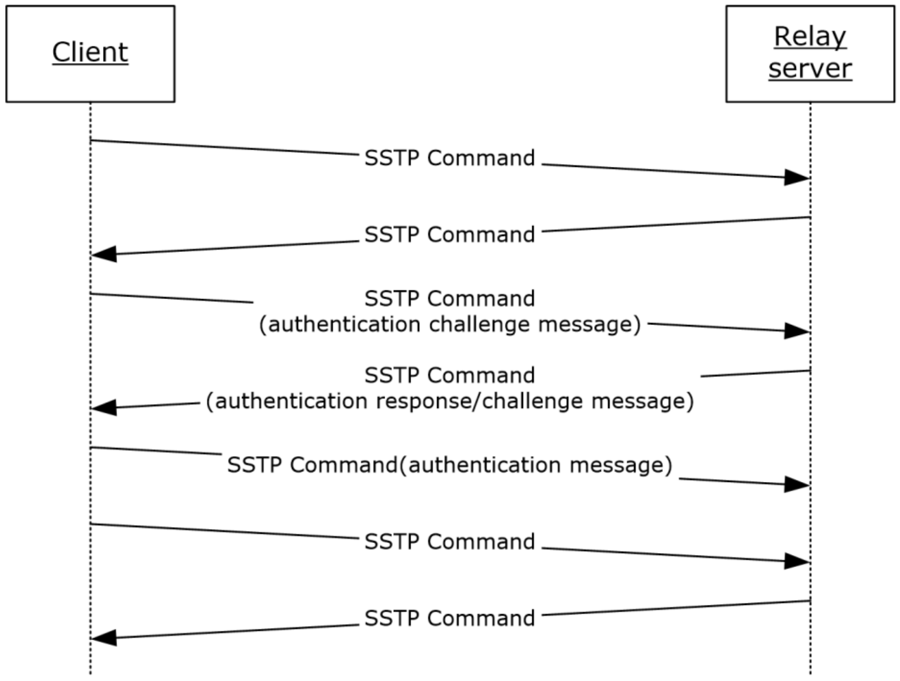
Client
Relay server
SSTP Command
SSTP Command
SSTP Command (authentication challenge message)
SSTP Command (authentication response/challenge message)
SSTP Command(authentication message)
SSTP Command
SSTP Command

Figure 3: Client authentication message sequence

1.4 Relationship to Other Protocols

This protocol is a sub-protocol to the SSTP protocol, which operates over TCP or HTTP. Protocol data for SSTP Security is embedded as authentication tokens in a selected set of SSTP commands. This protocol relies on the SSTP protocol to transport protocol messages between a client and a relay server, and can only function within the SSTP protocol.

The following diagram shows the protocol stack in relation to other protocols:

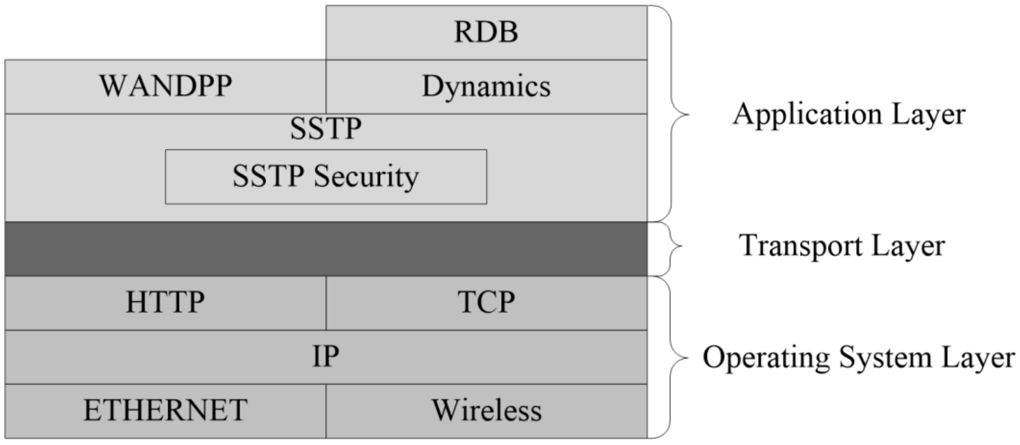

Figure 4: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

This protocol is designed to work only with the SSTP protocol. It cannot be implemented independently of the SSTP protocol. Therefore, to use this security protocol, both client and server have to implement the SSTP protocol.

A relay server is required to have a self-signed **X.509** certificate with several extensions (see section 3.1.1.1 for details), and a client obtains the certificate of its assigned relay server before using this protocol.

Before using this protocol, a client generates encryption and signature key pairs for the client device and for each account on the device (see section 3.1.1.3 for details).

1.6 Applicability Statement

This SSTP Security protocol is designed to augment the SSTP protocol with mechanisms to register client entities and then to authenticate these entities. Cryptographic keys are exchanged during the registrations, and then are used to produce challenges and responses during the authentications. A client and a server can implement these mechanisms to ensure that the server is communicating with an intended entity before delivering application data to a client.

An application that uses SSTP is responsible for protecting the integrity, confidentiality, and authenticity of the application data that is sent through a relay server. The authentication mechanisms specified by this protocol provide added security protection when clients communicate through a relay server. The device authentication prevents a potential attacker from receiving messages targeted to other devices, while the account authentication prevents a potential attacker from receiving messages targeted to other users. This protocol provides no mechanism to authenticate clients that send application data to a relay server.

1.7 Versioning and Capability Negotiation

This protocol provides no versioning and negotiation capability. However, the SSTP protocol supports features to negotiate peer capabilities. See [MS-GRVSSTP] for details on SSTP peer capability negotiation.

This protocol has no localization dependent behavior.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The SSTP Security protocol is a sub-protocol embedded within the SSTP protocol. Packets of the SSTP Security protocol are sent as a binary authentication token in one of several selected SSTP commands. The SSTP protocol is transported over TCP/IP using port 2492 or 443, and can also be encapsulated in HTTP using port 80. See [MS-GRVSSTP] and [MS-GRVHENC] for details on the SSTP protocol, and the HTTP encapsulation protocol.

2.2 Message Syntax

SSTP Security messages consist of a combination of fixed-length and variable-length fields. Although there are no limits on the lengths of the individual variable-length fields in the messages, the length of some variable-length variables is determined by cryptographic algorithms used in the protocol, and there is an absolute limit on the lengths of SSTP Security messages. SSTP Security messages cannot be larger than 6,144 (0x1800) bytes.

The format of all SSTP Security messages starts with a three-byte message header. The message header contains one byte for the SSTP Security protocol's major version number, one byte for the SSTP Security protocol's minor version number, and one byte for a numerical message identifier. Each message identifier identifies one of the security messages defined by this protocol.

The following two tables list the message identifiers defined by this protocol. Security messages are classified into two groups: device layer messages and account layer messages. The following table defines message identifiers for all device layer messages.

Device layer messages identifier	Value	Section
SecConnectMsgId	0x01	2.2.1
SecConnectResponseMsgId	0x02	2.2.2
SecConnectAuthenticateMsgId	0x03	2.2.5
SecDeviceAccountRegisterMsgId	0x04	2.2.12
SecDeviceAccountRegisterResponseMsgId	0x05	2.2.13
SecConnectResponseDeviceRegistrationNeededMsgId	0x0A	2.2.3
SecConnectResponseAuthenticationFailedMsgId	0x0C	2.2.4

The following table defines message IDs for all account layer messages.

Account layer message identifier	Value	Section
SecAttachMsgId	0x01	2.2.6
SecAttachResponseMsgId	0x02	2.2.7
SecAttachAuthenticateMsgId	0x03	2.2.11
SecAccountRegisterMsgId	0x04	2.2.14
SecAccountOnNewDeviceMsgId	0x05	2.2.15
SecIdentityRegisterMsgId	0x06	2.2.17
SecAccountRegisterResponseMsgId	0x08	2.2.16

Account layer message identifier	Value	Section
SecAttachResponseAccountRegistrationNeededMsgId	0x0A	2.2.8
SecAttachResponseNewDeviceRegistrationNeededMsgId	0x0B	2.2.9
SecAttachResponseAuthenticationFailedMsgId	0x0C	2.2.10

All SSTP security messages are embedded as an authentication token in one of the following SSTP commands [MS-GRVSSTP]:

- **Connect**
- **ConnectResponse**
- **ConnectAuthenticate**
- **Attach**
- **AttachResponse**
- **AttachAuthenticate**
- **Register**
- **RegisterResponse**

The SSTP commands that carry a security authentication token provide two fields in their command formats: one is **AuthenticationTokenLength**, which specifies the length of an authentication token, and the other is **AuthenticationToken** where the binary data of the authentication token is stored.

In the SSTP Security protocol, the byte order for all data MUST be **little-endian**.

The version numbers in a message header specify the protocol's version numbers of the sending device. Different major versions imply incompatible protocol packet formats. Devices with different major version numbers cannot communicate using the SSTP Security protocol. The major version number MUST be 1. Devices with different minor version numbers MAY communicate with each other using the SSTP Security protocol. The minor version number MUST be 3 or 4.<1>

In the messages specified in this section, a **nonce** generated by either a client or a server MUST be 24 bytes in length.

The terminating NULL character for an American National Standards Institute (ANSI) character set string is one byte and for a **Unicode** string it is two bytes.

When calculating a **SHA-1 hash**, a Unicode string is treated as an array of bytes without the terminating NULL character, but for an ANSI string the terminating NULL character is included.

The following sections describe the SSTP Security message formats.

2.2.1 SecConnect

The **SecConnect** message is used by a client to send a challenge to a relay server for the device authentication. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **Connect** command. The **AuthenticationTokenLength** field of the SSTP **Connect** command specifies the size in bytes of the binary byte sequence.

SSTP **Connect** MUST be the first SSTP command that a client sends when it connects to a relay server. If the client is connecting to its assigned relay server, it MUST include the **SecConnect** message as the authentication token in the SSTP **Connect** command to authenticate itself to the relay server. The relay server MUST NOT send device-targeted messages until the connecting device has been successfully authenticated.

The **SecConnect** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								IVLength							
...								IV (variable)																							
...																															
HMACLength																HMAC (variable)															
...																															
EncryptedDeviceNonceLength																EncryptedDeviceNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecConnectMsgId" for the **SecConnect** message.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field. The IV length MUST be the same as the key length for **RC4**, which is 24 bytes in length.

IV (variable): This field is a randomly generated block of binary data that the client has used as an initialization vector (IV) in encrypting a device nonce saved in the **EncryptedDeviceNonce** field.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a **keyed-hash Message Authentication Code** that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecConnectMsgId**
    2. **device URL**
    3. **ServerCertificateFingerprint**
    4. **DeviceNonce**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret device key.

Where **SecConnectMsgId** is the 1-byte message identifier as in the message header of the **SecConnect** message, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, **ServerCertificateFingerprint** is the server certificate fingerprint as calculated in section 3.1.1.2, and **DeviceNonce** is a random number of 24 bytes in length, which the client has generated for this message.

EncryptedDeviceNonceLength (2 bytes): This field specifies the length in bytes of the next field: **EncryptedDeviceNonce**. EncryptedDeviceNonceLength MUST be the same as the key length for RC4, which is 24 bytes in length.

EncryptedDeviceNonce (variable): This field contains a device nonce that the client has newly generated and encrypted using the **Modified Alleged Rivest Cipher 4 (MARC4) algorithm** cipher as specified in section 3.1.1.4, with the IV and the secret device key shared between the device and the relay server. The client sends this encrypted nonce to challenge the relay server about its knowledge of the secret device key. The secret device key is a RC4 key, which is 24 bytes in length.

2.2.2 SecConnectResponse

The **SecConnectResponse** message is sent by a relay server in response to a **SecConnect** message that the server received from the client. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **ConnectResponse** command. The **AuthenticationTokenLength** field of the SSTP **ConnectResponse** command specifies the size in bytes of the binary byte sequence.

The relay server uses the **SecConnectResponse** message to respond to the previous **SecConnect** message only if the server already has the secret device key and has successfully verified the device nonce and the HMAC from the **SecConnect** message. If the server does not have registration information for the connecting device, it MUST send a **SecConnectResponseDeviceRegistrationNeeded** message as a response to the **SecConnect** message.

The **SecConnectResponse** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								IVLength							
...								IV (variable)																							
...																															
HMACLength																HMAC (variable)															
...																															
DeviceNonceLength																DeviceNonce (variable)															
...																															
EncryptedRelayNonceLength																EncryptedRelayNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecConnectResponseMsgId" for the **SecConnectResponse** message.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field.

IV (variable): This field is a randomly generated block of binary data that has been used as an initialization vector (IV) in encrypting a relay nonce saved in the **EncryptedRelayNonce** field.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA1 algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecConnectReponseMsgId**
    2. **device URL**
    3. **ServerCertificateFingerprint**
    4. **RelayNonce**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret device key.

Where **SecConnectResponseMsgId** is the 1-byte message identifier as in the **SecConnectResponse** message header, **device URL** is a null-terminated ANSI string that uniquely identifies the client device, **ServerCertificateFingerprint** is the server certificate fingerprint as calculated in section 3.1.1.2, and **RelayNonce** is a random number of 24 bytes in length, which the server has newly generated.

DeviceNonceLength (2 bytes): This field specifies the length in bytes of the **DeviceNonce** field.

DeviceNonce (variable): This field contains the decrypted device nonce from the **SecConnect** message that the server has previously received from the client. The server MUST use the MARC4 cipher as specified in section 3.1.1.4, to decrypt the nonce.

EncryptedRelayNonceLength (2 bytes): This field specifies the length in bytes of the **EncryptedRelayNonce** field.

EncryptedRelayNonce (variable): This field contains a relay nonce that the server has newly generated and encrypted using the MARC4 cipher as specified in section 3.1.1.4, with the IV and the secret device key. The server sends this encrypted nonce to challenge the client about its knowledge of the secret device key.

2.2.3 SecConnectResponseDeviceRegistrationNeeded

The **SecConnectResponseDeviceRegistrationNeeded** message is sent by a relay server in response to the **SecConnect** message when the server has found out that it has no information about the connecting device or the secret device key. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **ConnectResponse** command. The **AuthenticationTokenLength** field of the SSTP **ConnectResponse** command specifies the size in bytes of the binary byte sequence.

The **SecConnectResponseDeviceRegistrationNeeded** message fields are shown in the following table.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
MajorVersionNumber								MinorVersionNumber								MessageID															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to **SecConnectResponseDeviceRegistrationNeededMsgId** for the **SecConnectResponseDeviceRegistrationNeeded** message.

2.2.4 SecConnectResponseAuthenticationFailed

The **SecConnectResponseAuthenticationFailed** message is sent by a relay server in response to a **SecConnect** message when the server can not verify the HMAC or decrypt the device nonce contained in the **SecConnect** message. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **ConnectResponse** command. The **AuthenticationTokenLength** field of the SSTP **ConnectResponse** command specifies the size in bytes of the binary byte sequence.

The **SecConnectResponseAuthenticationFailed** message fields are shown in the following table.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
MajorVersionNumber								MinorVersionNumber								MessageID															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecConnectResponseAuthenticationFailedMsgId" for the **SecConnectResponseAuthenticationFailed** message.

2.2.5 SecConnectAuthenticate

The **SecConnectAuthenticate** message is sent by a client to a relay server as a response to the **SecConnectResponse** message that the relay server has previously sent to the client. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **ConnectAuthenticate** command. The **AuthenticationTokenLength** field of the SSTP **ConnectAuthenticate** command specifies the size in bytes of the binary byte sequence.

The **SecConnectAuthenticate** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								RelayNonceLength							
...								RelayNonce (variable)																							
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecConnectAuthenticateMsgId" for the **SecConnectAuthenticate** message.

RelayNonceLength (2 bytes): This field specifies the length in bytes of the **RelayNonce** field.

RelayNonce (variable): This field contains the relay nonce that the client device has decrypted from the previous **SecConnectResponse** message sent by the relay server, using the MARC4 cipher as specified in section 3.1.1.4.

2.2.6 SecAttach

The **SecAttach** message is sent as a challenge by a client on behalf of an account to a relay server for the account authentication. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **Attach** command. The **AuthenticationTokenLength** field of the SSTP **Attach** command specifies the size in bytes of the binary byte sequence.

After the client has established a connection to its assigned relay server, it sends an SSTP **Attach** command to the relay server for each account on the client's device authenticated by the relay server.

The **SecAttach** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								IVLength							
...								IV (variable)																							
...																															
HMACLength																HMAC (variable)															
...																															
EncryptedAccountNonceLength																EncryptedAccountNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachMsgId" for the **SecAttach** message.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field.

IV (variable): This field is a randomly generated block of binary data that the client has used as an initialization vector (IV) in encrypting a device nonce saved in the **EncryptedAccountNonce** field.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA1 algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecAttachMsgId**
    2. **account URL**
    3. **relay URL**
    4. **device URL**
    5. **AccountNonce**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret account key.

Where **SecAttachMsgId** is the 1-byte message identifier as in the message header of the **SecAttach** message, **account URL** is a null-terminated ANSI string that uniquely identifies the account, **relay URL** is a null-terminated ANSI string that uniquely identifies the relay server, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, and **AccountNonce** is a random number of 24 bytes in length, which the client has newly generated for the account identified by the **account URL**. The **account URL** and **device URL** are both generated by the client. The **relay URL** is generated by the management server.

EncryptedAccountNonceLength (2 bytes): This field specifies the length in bytes of the **EncryptedAccountNonce** field.

EncryptedAccountNonce (variable): This field contains an account nonce that the client has newly generated, and encrypted using the MARC4 cipher as specified in section 3.1.1.4, with the IV and the secret account key. The client sends this encrypted nonce to challenge the relay server about its knowledge of the secret account key.

2.2.7 SecAttachResponse

The **SecAttachResponse** message is sent by a relay server in response to a **SecAttach** message that the relay server has previously received from the client. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **AttachResponse** command. The **AuthenticationTokenLength** field of the SSTP **AttachResponse** command specifies the size in bytes of the binary byte sequence.

The relay server sends the **SecAttachResponse** message as a response to the previous **SecAttach** message only if the server already has the secret account key and has successfully verified the account nonce and the HMAC from the **SecAttach** message. If the server has no knowledge about the account or the secret account key, it MUST send a **SecAttachResponseAccountRegistrationNeeded** message as a response to the **SecAttach** message. If the server has the secret account key, but no knowledge about the connecting device or the secret device key, then it MUST send a **SecAttachResponseNewDeviceRegistrationNeeded** message as a response to the **SecAttach** message. Otherwise, if the HMAC verification fails or any other processing errors are encountered, the server MUST send a **SecAttachResponseAuthenticationFailed** message.

The **SecAttachResponse** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								IVLength							
...								IV (variable)																							
...																															
HMACLength																HMAC (variable)															
...																															
AccountNonceLength																AccountNonce (variable)															
...																															
EncryptedRelayNonceLength																EncryptedRelayNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachResponseMsgId" for the **SecAttachResponse** message.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field.

IV (variable): This field is a randomly generated block of binary data that the relay server has used as an initialization vector (IV) in encrypting a relay nonce saved in the **EncryptedRelayNonce** field.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecAttachResponseMsgId**
    2. **account URL**
    3. **relay URL**
    4. **device URL**
    5. **RelayNonce**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret account key.

Where **SecAttachResponseMsgId** is the 1-byte message identifier as in the **SecAttachResponse** message header, **account URL** is a null-terminated ANSI string that uniquely identifies the client account, **relay URL** is a null-terminated ANSI string that uniquely identifies the relay server, **device URL** is a null-terminated ANSI string that uniquely identifies the client device, and **RelayNonce** is a random number of 24 bytes in length, which the relay server has newly generated as a challenge to the account identified by the **account URL**.

AccountNonceLength (2 bytes): This field specifies the length in bytes of the **AccountNonce** field.

AccountNonce (variable): This field contains the account nonce that the server has decrypted from the **SecAttach** message that the server has previously received from the client, using the MARC4 cipher as specified in section 3.1.1.4.

EncryptedRelayNonceLength (2 bytes): This field specifies the length in bytes of the **EncryptedRelayNonce** field.

EncryptedRelayNonce (variable): This field contains a relay nonce that the server has newly generated and encrypted using the MARC4 cipher as specified in section 3.1.1.4, with the IV and the secret account key. The server sends this encrypted nonce to challenge the client about its knowledge of the secret account key.

2.2.8 SecAttachResponseAccountRegistrationNeeded

The **SecAttachResponseAccountRegistrationNeeded** message is sent by a relay server in response to the **SecAttach** message when the server has no information about the connecting client's account or the secret account key. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **AttachResponse** command. The **AuthenticationTokenLength** field of the SSTP **AttachResponse** command specifies the size in bytes of the binary byte sequence.

The **SecAttachResponseAccountRegistrationNeeded** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachResponseAccountRegistrationNeededMsgId" for the **SecAttachResponseAccountRegistrationNeeded** message.

2.2.9 SecAttachResponseNewDeviceRegistrationNeeded

The **SecAttachResponseNewDeviceRegistrationNeeded** message is sent by a relay server in response to the **SecAttach** message when the server has found out that it has the secret account key, but no information about the connecting client's device or the secret device key. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **AttachResponse** command. The **AuthenticationTokenLength** field of the SSTP **AttachResponse** command specifies the size in bytes of the binary byte sequence.

The **SecAttachResponseNewDeviceRegistrationNeeded** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachResponseNewDeviceRegistrationNeededMsgId" for the **SecAttachResponseNewDeviceRegistrationNeeded** message.

2.2.10 SecAttachResponseAuthenticationFailed

The **SecAttachResponseAuthenticationFailed** message is sent by a relay server in response to the **SecAttach** message when the server fails to process the **SecAttach** message for reasons other than a new account or an account on a new device. The possible causes of failure include HMAC verification failure, a locked account, and system resource failures. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **AttachResponse** command. The **AuthenticationTokenLength** field of the SSTP **AttachResponse** command specifies the size in bytes of the binary byte sequence.

The **SecAttachResponseAuthenticationFailed** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachResponseAuthenticationFailedMsgId" for the **SecAttachResponseAuthenticationFailed** message.

2.2.11 SecAttachAuthenticate

The **SecAttachAuthenticate** message is sent by a client account to a relay server as a response to the **SecAttachResponse** message that the relay server previously sent to the sending client. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **AttachAuthenticate** command. The **AuthenticationTokenLength** field of the SSTP **AttachAuthenticate** command specifies the size in bytes of the binary byte sequence.

The **SecAttachAuthenticate** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								RelayAccountNonce Length							
...								RelayAccountNonce (variable)																							
...																															
RelayDeviceNonceLength																RelayDeviceNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAttachAuthenticateMsgId" for the **SecAttachAuthenticate** message.

RelayAccountNonceLength (2 bytes): This field specifies the length in bytes of the **RelayAccountNonce** field.

RelayAccountNonce (variable): This field contains the relay nonce that the sending client has decrypted from the previous **SecAttachResponse** message from the relay server, using the MARC4 cipher as specified in section 3.1.1.4.

RelayDeviceNonceLength (2 bytes): This field specifies the length in bytes of the **RelayDeviceNonce** field.

RelayDeviceNonce (variable): This field contains the relay nonce that the sending client has decrypted from the **SecConnectResponse** message or the **SecDeviceAccountRegisterResponse** message that the client received from the relay server, using the MARC4 cipher as specified in section 3.1.1.4.

2.2.12 SecDeviceAccountRegister

The **SecDeviceAccountRegister** message is sent by a client to register a new device or a new account on the client device with the relay server. When a client registers a device or an account with a relay server, the client exchanges a secret key with the relay server. The secret key is shared only between the client and the client's relay servers. The message is encoded as a binary byte sequence in the **AuthenticationToken** field of the SSTP **Register** command. The

AuthenticationTokenLength field of the SSTP **Register** command specifies the size in bytes of the binary byte sequence.

When a client registers an account, it MUST register the connecting device too, even if the device MAY have already registered with the relay server.

The **SecDeviceAccountRegister** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								Timestamp							
...																								AccountURL (variable)							
...																															
FingerprintLength																Fingerprint (variable)															
...																															
EncryptedRelayDeviceKeyLength																EncryptedRelayDeviceKey (variable)															
...																															
AccountLayerMessageLength																AccountLayerMessage (variable)															
...																															
Reserved1																Reserved2								SignatureLength							
...								Signature (variable)																							
...																															
DevicePublicKeysObjectLength																DevicePublicKeysObject (variable)															
...																															
IVLength																IV (variable)															
...																															
EncryptedDeviceNonceLength																EncryptedDeviceNonce (variable)															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecDeviceAccountRegisterMsgId" for the **SecDeviceAccountRegister** message.

Timestamp (4 bytes): This field is a 4-byte unsigned integer that represents the local time in seconds on the sending client's computer since midnight, January 1, 1970.

AccountURL (variable): This field contains a null-terminated ANSI string that uniquely identifies the account that is to register with the relay server.

FingerprintLength (2 bytes): This field specifies the length in bytes of the **Fingerprint** field.

Fingerprint (variable): This field contains the fingerprint of the relay server certificate. See section 3.1.1.2 for details on how to calculate the server certificate fingerprint.

EncryptedRelayDeviceKeyLength (2 bytes): This field specifies the length in bytes of the **EncryptedRelayDeviceKey** field.

EncryptedRelayDeviceKey (variable): This field contains the secret device key that the client has generated and encrypted with the relay server's encryption public key using **ElGamal encryption**. The client registers the device key with the relay server.

AccountLayerMessageLength (2 bytes): This field specifies the length in bytes of the **AccountLayerMessage** field.

AccountLayerMessage (variable): This field contains an account layer message that MUST be either the **SecAccountRegister** message (see section 2.2.14) or the **SecAccountOnNewDevice** message (see section 2.2.15).

Reserved1 (2 bytes): This field contains a 2-byte unsigned integer. The client MUST set the first byte to 0x01 and the second byte to 0x00. The relay server MUST ignore this field.

Reserved2 (1 byte): This field is a 1-byte field. The client MUST set this field to 0x00 and the relay server MUST ignore this field.

SignatureLength (2 bytes): This field specifies the length in bytes of the **Signature** field.

Signature (variable): This field contains the signature of the device registration message. The client MUST have generated the encryption and signature key pairs for the connecting device. It calculates the signature as follows:

1. Apply SHA1 [RFC3174] to the concatenation of the following elements in order:
    1. **SecDeviceAccountRegisterMsgId**
    2. **account URL**
    3. **device URL**
    4. **ServerCertificateFingerprint**
    5. **EncryptedDeviceNonce**
    6. **EncryptedRelayDeviceKey**
    7. **Timestamp**
    8. **DevicePublicKeysObject**
2. Apply SHA1 [RFC3174] to the hash computed in step 1, to produce the final hash value.

3. Sign the final hash value produced in step 2 with the connecting device's signature private key using RSA algorithm as defined in [PKCS1].

Where **SecDeviceAccountRegisterMsgId** is the 1-byte message identifier in the message header of the **SecDeviceAccountRegister** message, **account URL** is the null-terminated ANSI string in the **AccountURL** field, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, **ServerCertificateFingerprint** is the server certificate fingerprint from the **Fingerprint** field, as calculated in section 3.1.1.2, **EncryptedDeviceNonce** is the encrypted nonce in the **EncryptedDeviceNonce** field, **EncryptedRelayDeviceKey** is the encrypted device key in the **EncryptedRelayDeviceKey** field, **Timestamp** is the 4-byte local time in the **Timestamp** field, and **DevicePublicKeysObject** is the public keys object in the **DevicePublicKeysObject** field.

DevicePublicKeysObjectLength (2 bytes): This field specifies the length, in bytes, of the **DevicePublicKeysObject** field.

DevicePublicKeysObject (variable): This field contains the information about the public halves of the encryption and signature key pairs that the client has generated for the connecting device. See section 3.1.1.3 for details on how to generate a public keys object.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field.

IV (variable): This field is a randomly generated block of binary data that the sending client has used as an initialization vector (IV) in encrypting the device nonce saved in the **EncryptedDeviceNonce** field.

EncryptedDeviceNonceLength (2 bytes): This field specifies the length, in bytes, of the **EncryptedDeviceNonce** field.

EncryptedDeviceNonce (variable): This field contains a device nonce that the client has newly generated and encrypted using the MARC4 cipher as specified in section 3.1.1.4, with the IV and the secret device key.

2.2.13 SecDeviceAccountRegisterResponse

The **SecDeviceAccountResponse** message is sent by a relay server in response to the **SecDeviceAccountRegister** message that the relay server has received from the client. The relay server sends the **SecDeviceAccountResponse** message only if the server has successfully processed the **SecDeviceAccountRegister** message.

The message is encoded as a binary byte sequence in the **RegistrationToken** field of the SSTP **RegisterResponse** command. The **RegistrationTokenLength** field of the SSTP **RegisterResponse** command specifies the size in bytes of the binary byte sequence.

The **SecDeviceAccountResponse** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								AccountLayerMessage Length							
...								AccountLayerMessage (variable)																							
...																															
Reserved								IVLength																IV (variable)							

...	
HMACLength	HMAC (variable)
...	
DeviceNonceLength	DeviceNonce (variable)
...	
EncryptedRelayNonceLength	EncryptedRelayNonce (variable)
...	

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecDeviceAccountRegisterResponseMsgId" for the **SecDeviceAccountRegisterResponse** message.

AccountLayerMessageLength (2 bytes): This field specifies the length in bytes of the next field: **AccountLayerMessage**.

AccountLayerMessage (variable): This field MUST contain the **SecAccountRegisterResponse** message as specified in section 2.2.16.

Reserved (1 byte): The relay server MUST set the **Reserved** field to 0x00 and the client MUST ignore this field.

IVLength (2 bytes): This field specifies the length in bytes of the **IV** field.

IV (variable): This field is a randomly generated block of binary data that the relay server has used as an initialization vector (IV) in encrypting the relay nonce that is saved in the **EncryptedRelayNonce** field.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:

    1. **SecDeviceAccountRegisterResponseMsgId**
    2. 0x00, 1 byte
    3. **account URL**
    4. **device URL**
    5. **ServerCertificateFingerprint**

6. **RelayNonce**

2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in the step 1 with the secret device key.

Where **SecDeviceAccountRegisterResponseMsgId** is the 1-byte message identifier as in the SecConnectResponse message header, **account URL** is a null-terminated ANSI string that uniquely identifies the account, **device URL** is a null-terminated ANSI string that uniquely identifies the client device, **ServerCertificateFingerprint** is the server certificate fingerprint as calculated in section 3.1.1.2, and **RelayNonce** is a random number of 24 bytes in length, which the server has newly generated.

DeviceNonceLength (2 bytes): This field specifies the length in bytes of the **DeviceNonce** field.

DeviceNonce (variable): This field contains the device nonce that the server has decrypted from the **SecDeviceAccountRegister** message that the server has received from the client, using the MARC4 cipher as specified in section 3.1.1.4.

EncryptedRelayNonceLength (2 bytes): This field specifies the length in bytes of the **EncryptedRelayNonce** field.

EncryptedRelayNonce (variable): This field contains a relay nonce that the server has newly generated and encrypted using the MARC4 cipher as specified in section 3.1.1.4, with the IV and the secret device key. The server sends this encrypted nonce to challenge the client about its knowledge of the secret device key.

2.2.14 SecAccountRegister

The **SecAccountRegister** message is an account layer message that a client sends to a relay server as part of the **SecDeviceAccountRegister** message. The **SecAccountRegister** message contains information for the account registration, and is encoded as a binary block in the **AccountLayerMessage** field of the **SecDeviceAccountRegister** message. The field **AccountLayerMessageLength** in the **SecDeviceAccountRegister** message specifies the length in bytes of the **SecAccountRegister** message. A client sends the **SecAccountRegister** message along with the **SecDeviceAccountRegister** message to register a new account and the connecting device with a relay server.

The **SecAccountRegister** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								EncryptedRelayAccountKey Length							
...								EncryptedRelayAccountKey (variable)																							
...																															
SignatureLength																Signature (variable)															
...																															
AccountPublicKeysObjectLength																AccountPublicKeysObject (variable)															

...		
Reserved1	Reserved2	UserPreAuthToken
...		

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAccountRegisterMsgId" for the **SecAccountRegister** message.

EncryptedRelayAccountKeyLength (2 bytes): This field specifies the length in bytes of the **EncryptedRelayAccountKey** field.

EncryptedRelayAccountKey (variable): This field contains the secret account key that the client has generated and encrypted using the relay server's encryption public key. The client registers the account key with the relay server.

SignatureLength (2 bytes): This field specifies the length in bytes of the **Signature** field.

Signature (variable): This field contains the signature of the account registration message. The client MUST have generated the encryption and signature key pairs for the account. It calculates the signature as follows:

1. Apply SHA1 [RFC3174] to the concatenation of the following element in order:
    1. **SecAccountRegisterMsgId**
    2. **account URL**
    3. **device URL**
    4. **ServerCertificateFingerprint**
    5. **Timestamp**
    6. **EncryptedRelayAccountKey**
    7. **AccountPublicKeysObject**
2. Apply SHA1 [RFC3174] to the hash computed in step 1, to produce the final hash value.
3. Sign the final hash value produced in step 2 with the account's signature private key using RSA algorithm as defined in [PKCS1].

Where **SecAccountRegisterMsgId** is the 1-byte message identifier as in the message header of the **SecDeviceAccountRegister** message, **account URL** is the null-terminated ANSI string from the **AccountURL** field of the **SecDeviceAccountRegister** message, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, **ServerCertificateFingerprint** is the server certificate fingerprint from the **Fingerprint** field of the **SecDeviceAccountRegister** message, as calculated in section 3.1.1.2, **Timestamp** is the 4 byte local time from the **Timestamp** field of the **SecDeviceAccountRegister** message, **EncryptedDeviceNonce** is the encrypted nonce from the **EncryptedDeviceNonce** field, **EncryptedRelayAccountKey** is the encrypted account key from the **EncryptedRelayAccountKey** field of the **SecAccountRegister** message, and **AccountPublicKeysObject** is the public keys object in the **AccountPublicKeysObject** field.

AccountPublicKeysObjectLength (2 bytes): This field specifies the length in bytes of the **AccountPublicKeysObject** field.

AccountPublicKeysObject (variable): This field contains the information about the public halves of the encryption and signature key pairs that the client has generated for the client account. See section 3.1.1.3 for the format of a public keys object.

Reserved1 (2 bytes): This field contains a 2-byte unsigned integer. The client MUST set the first byte to 0x01 and the second byte to 0x00.

Reserved2 (1 byte): This field is a 1-byte field. The client MUST set this field to 0x00. The relay server MUST ignore this field.

UserPreAuthToken (variable): This field contains a null-terminated ANSI string that a management server has created and sent to the client when the client configures its identity. This string uniquely identifies a user and is used as a pre-authentication token when a client registers its account with a relay server. The relay server SHOULD receive this pre-authentication token from the management server, and SHOULD validate the value of the **UserPreAuthToken** field against the pre-authentication token received earlier from the management server. If the value of the **UserPreAuthToken** field does not match the pre-authentication token that the relay server received from the management server, the relay server MUST send a **Close** command to close the current **Attach** session with the **ReasonId** field in the **Close** command set to **ReasonIdUserAuthenticationFailed**.

2.2.15 SecAccountOnNewDevice

The **SecAccountOnNewDevice** message is an account layer message that a client sends to a relay server as part of the **SecDeviceAccountRegister** message. The client embeds the **SecAccountOnNewDevice** message in the **SecDeviceAccountRegister** message as an account layer message when it finds out that the account has already registered with the relay server, but the connecting client device has not.

The **SecDeviceAccountRegister** message is encoded as a binary block in the **AccountLayerMessage** field of the **SecDeviceAccountRegister** message. The field **AccountLayerMessageLength** in the **SecDeviceAccountRegister** message specifies the length in bytes of the **SecAccountOnNewDevice** message.

The **SecAccountOnNewDevice** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								HMACLength							
...								HMAC (variable)																							
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAccountOnNewDeviceMsgId" for the **SecAccountOnNewDevice** message.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecAccountOnNewDeviceMsgId**
    2. **account URL**
    3. **device URL**
    4. **ServerCertificateFingerprint**
    5. **Timestamp**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret account key.

Where **SecAccountOnNewDeviceMsgId** is the 1-byte message identifier that is specified in the **SecAccountOnNewDevice MessageID** field, **account URL** is a null-terminated ANSI string that uniquely identifies the account, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, **ServerCertificateFingerprint** is the server certificate fingerprint as calculated in section 3.1.1.2, and **Timestamp** is the 4 byte local time from the **Timestamp** field of the **SecDeviceAccountRegister** message.

2.2.16 SecAccountRegisterResponse

The **SecAccountRegisterResponse** message is an account layer message that a relay server sends to a client as part of the **SecDeviceAccountRegisterResponse** message in response to the **SecDeviceAccountRegister** message that the relay server received from the connecting client. The relay server sends the **SecAccountRegisterResponse** message only if the server has successfully processed the **SecDeviceAccountRegister** and **SecAccountRegister** messages. If the server encounters any error while processing the messages, it MUST send an SSTP command such as the **ConnectClose** or **Close** command to the client as a response. See section 3.3 for details on server error handling.

This **SecAccountRegisterResponse** message is encoded as a binary block in the **AccountLayerMessage** field of the **SecDeviceAccountRegisterResponse** message. The field **AccountLayerMessageLength** in the **SecDeviceAccountRegisterResponse** message specifies the length in bytes of the **SecAccountRegisterResponse** message.

The **SecAccountRegisterResponse** message fields are shown in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MajorVersionNumber								MinorVersionNumber								MessageID								Reserved							
Timestamp																															
HMACLength																HMAC															
...																															

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending relay server.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending relay server.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecAccountRegisterResponseMsgId" for the **SecAccountRegisterResponse** message.

Reserved (1 byte): The relay server MUST set this field to 0x00, and the client MUST ignore this field.

Timestamp (4 bytes): This field is a 4-byte unsigned integer that represents the local time, in seconds, on the relay server's computer since midnight, January 1, 1970.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This field contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:
    1. **SecAccountRegisterResponseMsgId**
    2. 0x00, 1 byte
    3. **account URL**
    4. **device URL**
    5. **ServerCertificateFingerprint**
    6. **Timestamp**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret account key.

Where **SecAccountRegisterResponseMsgId** is the 1-byte message identifier in the **SecAccountRegisterResponse** message header, **0x00** is one byte, **account URL** is a null-terminated ANSI string that uniquely identifies the account, **device URL** is a null-terminated ANSI string that uniquely identifies the connecting client device, **ServerCertificateFingerprint** is the server certificate fingerprint as calculated in section 3.1.1.2, and **Timestamp** is the 4 byte local time from the **Timestamp** field of the **SecAccountRegisterResponse** message.

2.2.17 SecIdentityRegister

The **SecIdentityRegister** message is sent by a client to a relay server to register a list of identities on the client's account after a successful account authentication. The message is encoded as a binary byte sequence in the **RegistrationToken** field of the SSTP **Register** command. The **RegistrationTokenLength** field of the SSTP **Register** command specifies the size in bytes of the binary byte sequence.

The **SecIdentityRegister** message fields are shown in the following table.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
MajorVersionNumber								MinorVersionNumber								MessageID								Timestamp							

...		AccountURL (variable)
...		
HMACLength	HMAC (variable)	
...		
Reserved	IdentityListsLength	IdentityLists (variable)
...		
RelayURL (variable)		
...		

MajorVersionNumber (1 byte): This field specifies the SSTP Security protocol major version number of the sending client.

MinorVersionNumber (1 byte): This field specifies the SSTP Security protocol minor version number of the sending client.

MessageID (1 byte): This field is a numerical message identifier, and MUST be set to "SecIdentityRegisterMsgId" for the **SecIdentityRegister** message.

Timestamp (4 bytes): This field is a 4-byte unsigned integer that represents the local time on the sending client's computer in seconds since midnight, January 1, 1970.

AccountURL (variable): This field contains a null-terminated ANSI string that uniquely identifies the account for the identities that are to register with the relay server.

HMACLength (2 bytes): This field specifies the length in bytes of the **HMAC** field.

HMAC (variable): This contains a keyed hash message authentication code that is calculated as follows:

1. Apply the SHA-1 hash algorithm [RFC3174] to the concatenation of the following elements in order:
   1. **SecIdentityRegisterMsgId**
   2. **account URL**
   3. **relay URL**
   4. **device URL**
   5. **Timestamp**
2. Apply the HMAC-SHA1 algorithm [RFC4634] to the hash value produced in step 1 with the secret account key.

Where **SecIdentityRegisterMsgId** is the 1-byte message identifier in the **SecIdentityRegister** message header, **account URL** is the null-terminated ANSI string from the **AccountURL** field, **relay URL** is the null-terminated ANSI string from the **RelayURL** field, **device URL** is a null-terminated

ANSI string that uniquely identifies the client device, **Timestamp** is the 4-byte local time from the **Timestamp** field.

Reserved (1 byte): The client MUST set this field to 0x00, and the relay server MUST ignore this field.

IdentityListsLength (2 bytes): This field specifies the length in bytes of the **IdentityLists** field.

IdentityLists (variable): This field contains two lists of **identity URLs**. The first is a list of identities to be added to the relay server, and the second is a list of identities to be removed from the relay server. The field MUST be formatted as shown in the following table.

										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
IdentitiesToAddCount								IdentitiesToRemove Count								IdentityURL 1 (variable)															
...																															
IdentityURL 2 (variable)																															
...																															
IdentityURL 3 (variable)																															
...																															
IdentityURL 4 (variable)																															
...																															
IdentityURL n (variable)																															
...																															

IdentitiesToAddCount (1 byte): This field is a 1-byte unsigned integer that specifies the number of identity URLs to be added to the relay server.

IdentitiesToRemoveCount (1 byte): This field is a 1-byte unsigned integer that specifies the number of identity URLs to be removed from the relay server.

IdentityURL 1 (variable): First null-terminated ANSI identity URL, with the URLs to be added specified first, and the URLs to be removed specified second.

IdentityURL 2 (variable): Second null-terminated ANSI identity URL, with the URLs to be added specified first, and the URLs to be removed specified second.

IdentityURL 3 (variable): Third null-terminated ANSI identity URL, with the URLs to be added specified first, and the URLs to be removed specified second.

IdentityURL 4 (variable): Fourth null-terminated ANSI identity URL, with the URLs to be added specified first, and the URLs to be removed specified second.

IdentityURL n (variable): Last null-terminated ANSI identity URL, with the URLs to be added specified first, and the URLs to be removed specified second.

RelayURL (variable): This field contains a null-terminated ANSI string that uniquely identifies the relay server with which the list of identities is to be registered.

3 Protocol Details

3.1 Common Details

This section specifies details that are common to both protocol server and protocol client behavior.

While the SSTP protocol is used for communications between clients, and between clients and relay servers, the SSTP Security sub-protocol has been designed specifically for registration and mutual authentication between a client and a relay server.

3.1.1 Common Security Parameter Formats and Processing

3.1.1.1 Relay Server Certificate

Each relay server MUST have an X.509 v3 certificate [RFC3280] that binds the relay server's URL with the server's encryption and signature public key pairs. Each client MUST have obtained the X.509 certificate of its assigned relay server before connecting to the server for the first time. Each X.509 server certificate MUST have the following three extensions:

1. Encryption Key: This extension specifies the server's Diffie-Hellman encryption public key. The **object identifier (OID)** for this extension MUST be "2.16.840.1.114227.1.1.1." The value MUST be the Diffie-Hellman public key used for ElGamal encryption<2>. The details of the DHPublicKeyForElgamalEncryption syntax are specified following this list.

2. Encryption Key Algorithm: This extension specifies the name of the encryption public key algorithm. The OID for this extension MUST be "2.16.840.1.114227.1.1.2", and the value MUST be "DH", which is a Unicode string without the terminating NULL character.

3. Encryption Algorithm: This extension specifies the name of the encryption algorithm. The OID for this extension MUST be "2.16.840.1.114227.1.1.3", and the value MUST be "ELGAMAL", which is a Unicode string without the terminating NULL character.

```
DHPublicKeyForElgamalEncryption::= SEQUENCE {
      p    INTEGER, -- prime, p
      q    INTEGER OPTIONAL, -- factor of p-1, only present when p = j*q+1,
-- where j is not 2
      g    INTEGER, -- generator, g
      y    INTEGER -- public key (g^x mod p, where x is the private key)
}
```

The relay server certificate MUST also be self-signed using SHA1-RSA [PKCS1].

3.1.1.2 Relay Server Certificate Fingerprint

The server certificate fingerprint is calculated by applying the SHA-1 hash function [RFC3174] to the concatenation of the following elements in order from the server certificate:

1. EncryptionPublicKeyAlgorithmName

2. EncryptionAlgorithmName

3. EncryptionPublicKey

where **EncryptionPublicKeyAlgorithmName** is the null-terminated Unicode string for the encryption public key algorithm name, **EncryptionAlgorithmName** is the null-terminated Unicode string for the encryption algorithm name, and **EncryptionPublicKey** is the DER-encoded encryption public key. The values of these 3 elements MUST be from the relay server's X.509 certificate (See section 3.1.1.1). Although the first two fields are null-terminated Unicode strings, the null-terminators are not included when concatenating the fields.

3.1.1.3 Client Public Keys Object Format

A client MUST generate encryption and signature key pairs for the client device and each account on the device. The signature key pairs MUST be generated using the RSA public key algorithm [PKCS1], and the encryption key pair MUST be generated using either the ElGamal public key algorithm [CRYPTO] with Crypto++ padding or the RSA public key algorithm [PKCS1].<3>

Crypto++ ElGamal encryption is done as follows:

1. Inputs:
   - LenM: Number of bytes in the modulus of the underlying Diffie-Hellman group.
   - LenP: Number of bytes of the plaintext, which is the input for encryption.
2. Allocate a block of (LenM minus 1) bytes.
3. Generate (LenM minus 2 minus LenP) bytes of randomness, and copy them starting at offset 0 of the allocated block.
4. Copy the plaintext after the (LenM minus 2 minus LenP) random bytes in the allocated block. This will occupy almost all the remaining space in the block, except for the last byte.
5. Set the last byte of the allocated block at offset (LenM minus 2) to LenP.
6. Apply ElGamal encryption to the resulting block, interpreting the block as if it was in Big Endian format. The result of encryption is encoded into Big Endian format as well.

Crypto++ ElGamal decryption is done as follows:

1. Treat the encrypted input as Big Endian format, and apply ElGamal decryption to get the decrypted block, which also will be in Big Endian format.
2. Get the first byte of the decrypted block, which is LenP because it is in Big Endian format and therefore the order of bytes is reversed.
3. Get the next LenP bytes of the decrypted block, which is the original plaintext in Big Endian format.
4. Convert the original plaintext from Big Endian to Little Endian.

When registering a device or an account, the client MUST send the server a public keys object that contains information about the public halves of the encryption and signature key pairs that the client has generated for the device or account. A public keys object MUST be formatted as follows, with fields as specified in the following table.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Signature Algorithm Name (variable)																															
...																															

Encryption Algorithm Name (variable)
...
Signature Key Algorithm Name (variable)
...
Encryption Key Algorithm Name (variable)
...
Signature Public Key Length
Signature Public Key (variable)
...
Encryption Public Key Length
Encryption Public Key (variable)
...

Signature Algorithm Name (variable): Null-terminated ANSI string that MUST be set to "RSA". This field specifies the name of the signature algorithm.

Encryption Algorithm Name (variable): Null-terminated ANSI string that MUST be set to "ELGAMAL" or "RSA". This field specifies the name of the encryption algorithm.

Signature Key Algorithm Name (variable): Null-terminated ANSI string that MUST be set to "RSA". This field specifies the name of the type of signature key used in the signature algorithm.

Encryption Key Algorithm Name (variable): Null-terminated ANSI string that MUST be set to "DH" or "RSA". This field specifies the name of the type of encryption key used in the encryption algorithm.

Signature Public Key Length (4 bytes): Length of the **Signature Public Key** field.

Signature Public Key (variable): DER-encoded signature public key.

Encryption Public Key Length (4 bytes): Length of the **Encryption Public Key** field.

Encryption Public Key (variable): DER-encoded encryption public key.

3.1.1.4 MARC4

This security protocol uses a variation of the RC4 algorithm called Modified Alleged RC4, or MARC4 to encrypt or decrypt the nonce exchanged between the client and the relay server during the registration and during authentication.

Original RC4 algorithm is described in [SCHNEIER].

MARC4 used in this protocol differs from RC4 in the following areas:

- Both encryption and decryption MUST use a random initialization vector (IV), and the IV MUST be the same size as the secret device or account key.
- A new random IV MUST be generated every time the data is encrypted. The matching IV MUST be used every time the data is decrypted.
- The IV MUST be XOR-ed with the secret device or account key, and the result MUST be used as the initial 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 that it is used in RC4.

3.1.2 Common Data Model

Clients and relay servers both MUST implement SSTP connections and sessions to support the SSTP Security protocol.

3.1.2.1 Connections

As SSTP Security messages are embedded in SSTP commands, both client and relay server MUST support and implement the SSTP protocol as specified in [MS-GRVSSTP] before considering using this security protocol for registrations and authentications. Because the SSTP Security is entirely dependent on SSTP, implementing the SSTP protocol is a prerequisite to implementing this protocol.

Security messages are transmitted over an SSTP connection that is established after a client and server successfully exchange the SSTP **Connect**/**ConnectResponse** commands. The client maintains just one SSTP connection for one relay server, and MUST send a **Connect** command as the very first command to initiate a connection with a relay server.

3.1.2.2 Sessions

The **Attach** and **Register** commands provide the **EventId** field that is used to correlate the commands with subsequent commands involved in the account authentication and in the registration. Subsequent commands such as **AttachResponse**, **AttachAuthenticate**, and **RegisterResponse** MUST include the same **EventId** as in the initial **Attach** or **Register** command. This **EventId** creates an SSTP session between the client and the server.

When a client sends an **Attach** command to a relay server, an **Attach** session is created between the client and the server. The number of commands that are exchanged within this session depends on whether or not the client has registered its account and device with the relay server. If the client has already registered its account and device, then the client and the server just need to exchange **Attach**/**AttachResponse**/**AttachAuthenticate** commands to authenticate its account within the session. Otherwise, if the client has not yet registered its account and device, the server instructs the client to register its new account or its account on a new device. In this case, the client and the server exchange the following commands within the same **Attach** session: **Attach**, **AttachResponse**, **AttachAuthenticate**, **Register**, **RegisterResponse** and **Close**. These commands MUST carry the same **EventId** so that they all belong to the same **Attach** session.

The client can also create a **Register** session with the relay server when the client and the relay server exchange just the **Register**/**RegisterResponse** commands, for example, when the client registers identities to the relay server.

3.1.3 Message Mappings

The SSTP Security protocol supports two types of authentication: device authentication and account authentication. It also supports three types of registration: account-device registration, account-on-new-device registration, and identity registration.

This section shows mappings between SSTP commands and security messages. The mapping is important as this information is used to parse security messages correctly.

3.1.3.1 Device Authentication Messages

A client MUST authenticate its device to a server before the client can retrieve device-targeted messages from the server. The SSTP protocol provides these three commands to transport security messages for device authentications: **Connect**, **ConnectResponse** and **ConnectAuthenticate**. The following table shows what security messages are mapped to each of the 3 SSTP commands.

SSTP Command	SSTP Security Message	Direction
Connect	SecConnect	Client to server
ConnectResponse	SecConnectResponse, SecConnectResponseDeviceRegistrationNeeded, SecConnectResponseAuthenticationFailed	Server to client
ConnectAuthenticate	SecConnectAuthenticate	Client to server

3.1.3.2 Account Authentication Messages

A client MUST authenticate its account to a server before the client can retrieve identity-targeted messages from the server. The SSTP protocol provides these three commands to transport security messages for account authentications: **Attach**, **AttachResponse** and **AttachAuthenticate**. The following table shows what security messages are mapped to each of these SSTP commands.

SSTP Command	SSTP Security Message	Direction
Attach	SecAttach	Client to server
AttachResponse	SecAttachResponse, SecAttachResponseAccountRegistrationNeeded, SecAttachResponseNewDeviceRegistrationNeeded, SecAttachResponseAuthenticationFailed	Server to client
AttachAuthenticate	SecAttachAuthenticate	Client to server

3.1.3.3 Registration Messages

A relay server MUST have both the client account and device to be registered to have complete authorization to include the device as valid for receiving the account's data. Therefore the device registration always piggybacks the account registration, and a client sends just one message that includes all information to register both account and device.

A client generates a secret key for each device and each account, and then shares the key with its relay server. For each account, there is a device key (shared among all accounts on the same device) for the device that the client application runs on, and there is also an account key for the account itself. A client registers these keys with the server first, and then uses the keys to authenticate its device and account to the server. A client registers its identities to the relay server after its account is authenticated.

For the device and account registration, a client exchanges secret keys (device key and account key) with a relay server once for each triplet {account URL, device URL, and relay URL}. The registration establishes a relationship and an agreement among these three entities. There are two scenarios for registration: one is for account and device registration for a new account and the other is for device registration for an existing account on a new device.

The SSTP protocol provides these two commands to transport security messages for a client to register keys and identities to a server: **Register** and **RegisterResponse**. The following table shows what security messages are mapped to each of these commands.

SSTP Command	SSTP Security Message	Direction
Register	SecDeviceAccountRegister, SecIdentityRegister	Client to server
RegisterResponse	SecDeviceAccountRegisterResponse	Server to client

The **SecDeviceAccountRegister** and **SecDeviceAccountRegisterResponse** messages are exchanged between a client and a server to register both device and account. They both contain an account layer message with information used for account registration. The following table shows what account layer messages are mapped to each of these security messages.

Security Message	Account Layer Message	Direction
SecDeviceAccountRegister	SecAccountRegister, SecAccountOnNewDevice	Client to server
SecDeviceAccountRegisterResponse	SecAccountRegisterResponse	Server to client

3.2 Client 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.

In addition to connections and sessions, a client needs to maintain the following conceptual objects to participate in this protocol.

Account Object: one for each account that the client has created. An account object contains the account URL, the URLs of the devices that the account runs on, the secret account key, the encryption and signature key pairs for the account. The account object also keeps a list of identity URLs active on the account, and a list of identity URLs inactive on the account. Whenever a new identity is added, the client adds the identity URL<4> to the list of active identity URLs in the account object, and whenever an identity is removed, the client adds the identity URL to the list of inactive identity URLs in the account object.

Identity Object: one for each identity that has been added to an account. An identity object contains the identity URL, the associated account URL, and the pre-authentication token received from the management server.

Device Object: one for each operating system user on a given computer. A device object contains the device URL, the secret device key, the encryption and signature key pairs for the device.

A client persists these objects so that the information contained in the object can live beyond the duration of the current client session.

A client uses the SSTP protocol to establish a multiplexed, asynchronous communication connection with a relay server. The abstract data model as described in [MS-GRVSSTP] is followed when implementing this security protocol. Within each connection, the client maintains a state about the status of the device authentication. For each account on the device, the client also maintains a state about the account's authentication status.

The following diagrams, titled Client state transition for a new connection request and Client state transition on an established connection, show how a client transitions its state when it issues a new SSTP connection request to an assigned relay, or when it attempts to authenticate an account on an established SSTP connection. A state transition can take place when the client sends or receives a message from the relay server.

The states that a client can be in are described following:

- **Connecting**: The client has sent a **Connect** command and is waiting for an SSTP connection to be established to a relay server.
- **Connected**: The client has established an SSTP connection to the relay server after receiving the **Ok** status in the **ResponseId** field of **ConnectResponse** command.
- **ConnectAuthenticated**: The client has received the **Ok** status in the **ResponseId** field of **ConnectResponse** command, and has sent the **ConnectAuthenticate** command.
- **AuthenticationFailed**: The client has received either the **ConnectResponseAuthenticationFailed** status in the ResponseId field of **ConnectResponse** command, or the **AttachResponseAuthenticationFailed** status in the **ResponseId** field of **AttachResponse** command.
- **AttachChallenging:** The client has sent an **Attach** command to start the account authentication with the relay server.
- **AttachAuthenticated**: The client has successfully authenticated its account and is ready to send or receive data from the relay server.
- **DeviceAccountRegistering**: The client has sent the **SecDeviceAccountRegister** and **SecAccountRegister** messages, and is waiting for a server response.
- **AccountOnNewDeviceRegistering**: The client has sent the **SecDeviceAccountRegister** and **SecAccountOnNewDevice** messages, and is waiting for a server response.
- **DeviceAccountRegistered**: The client has received the **SecDeviceAccountRegisterResponse** message from the relay server, and has successfully processed the message.
- **RegistrationFailed**: The client has received the **Close** command to close the **Attach/Register** session after having sent the **SecDeviceAccountRegister** message to the relay server.

- **Sending/Receiving**: The client is sending or receiving data from the relay server using the SSTP commands. This is not covered by this protocol.
- **Sending**: The client is only sending data to the relay server using the SSTP commands. This is not covered by this protocol.
- **Disconnected:** The client or relay server has closed the transport layer and will not be sending or receiving any more messages.

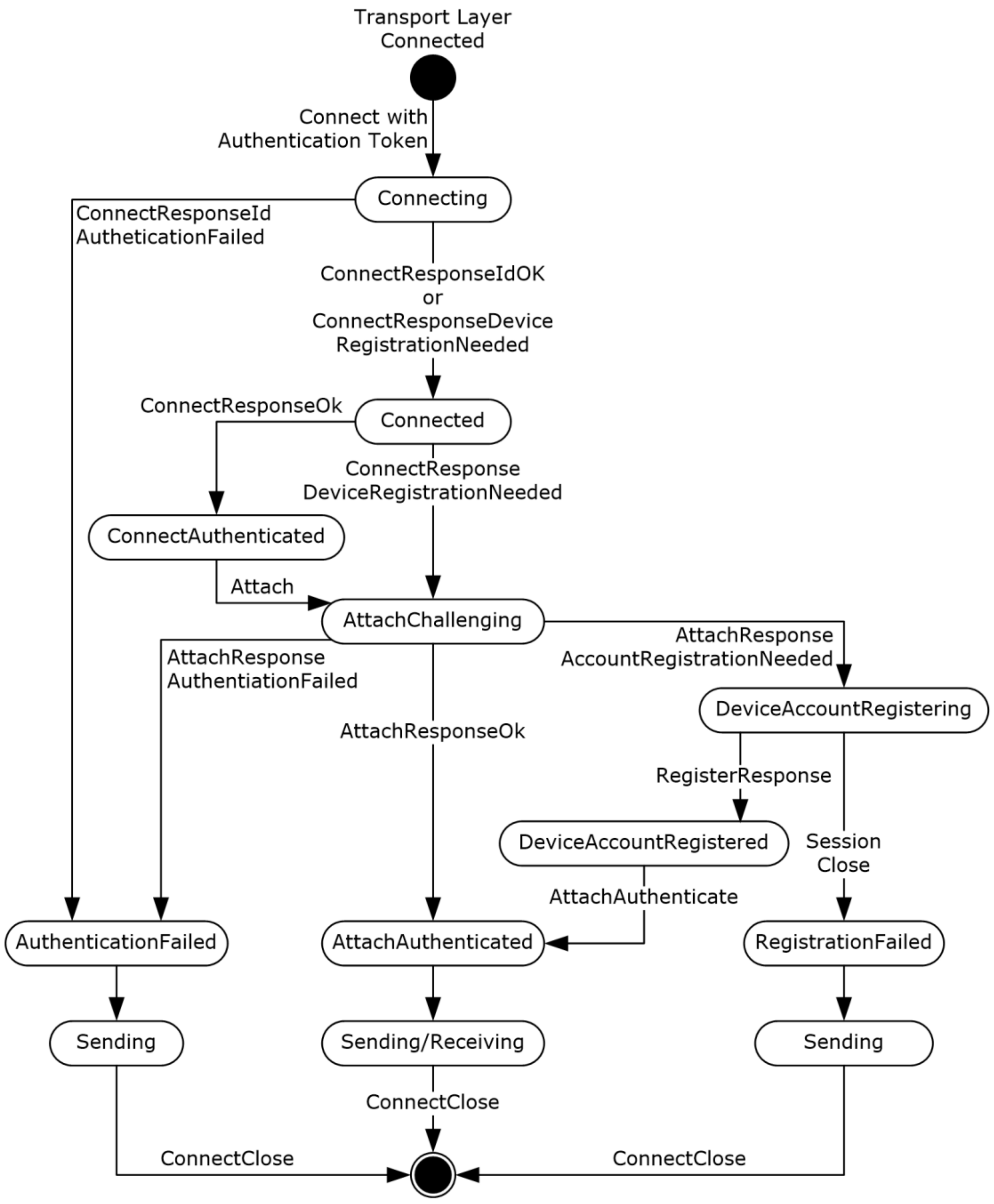

Figure 5: Client state transition for a new connection request

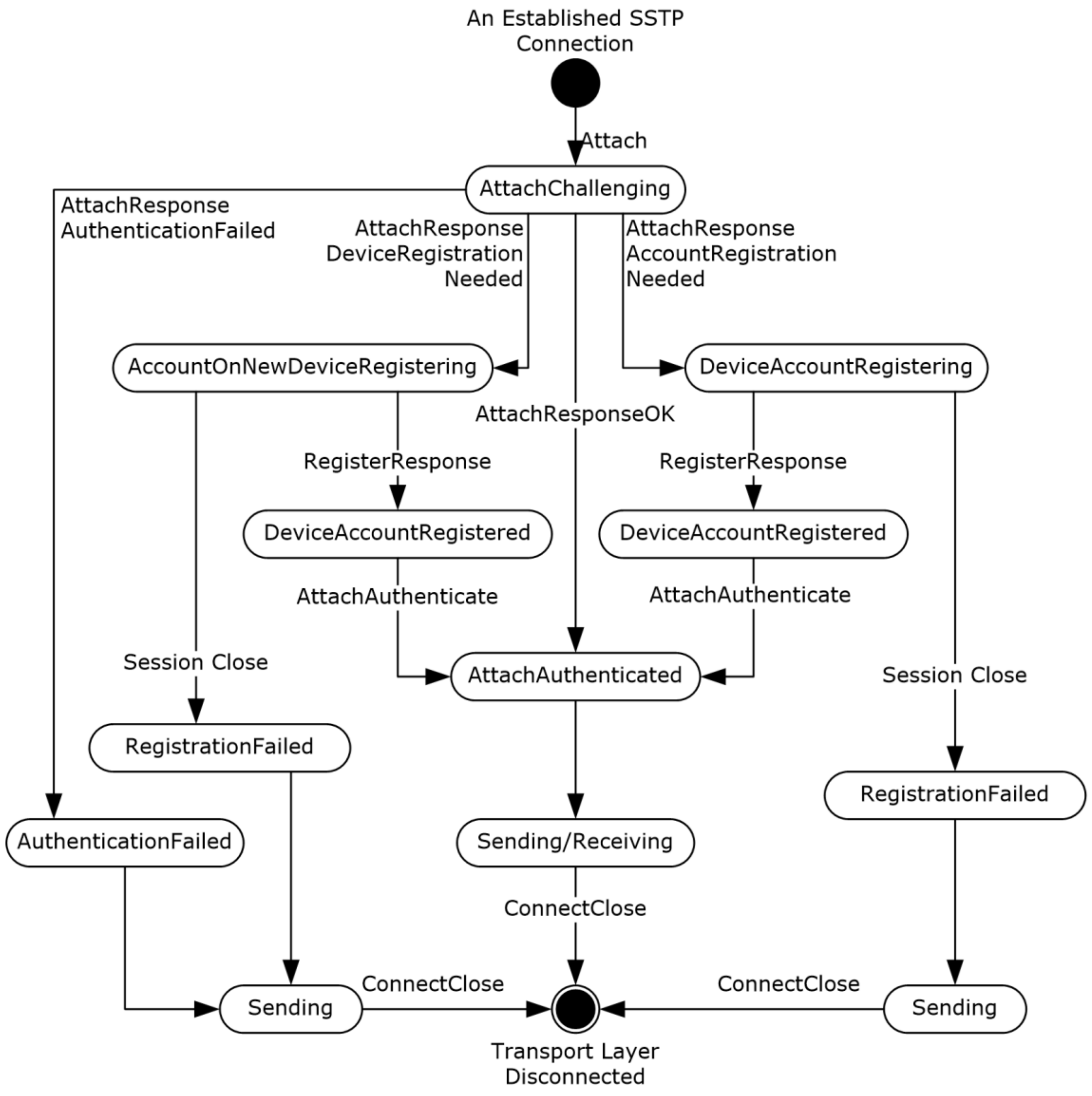

Figure 6: Client state transition on an established connection

3.2.2 Timers

The relay server has no timer for this security protocol, but has timers at the SSTP level. See [MS-GRVSSTP] for information.

3.2.3 Initialization

The client MUST connect on an SSTP port (2492, 443 or 80) to send/receive SSTP commands and security messages. If the client is connecting to its assigned relay server, then it MUST include an authentication token in the initial SSTP **Connect** command to authenticate the device to the relay server. See section 2.2.1 and [MS-GRVSSTP] for more details.

The client MUST provide its device URL, which is used in many messages of this protocol, in the **SourceDeviceURLs** field of the initial SSTP **Connect** command.

Before connecting to the relay server, the client MUST have already received the server's X.509 v3 certificate, and have already generated the encryption and signature key pairs for the client device and account.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Higher-Layer Device Authentication Initiation

If the client is establishing a new SSTP connection to its assigned relay server, and plans to retrieve data targeted to the device, then it MUST send an authentication challenge in the form of a **SecConnect** message to the relay server in the SSTP **Connect** command to authenticate the device. A device authentication MUST occur when a client establishes a new SSTP connection to a relay server. If a device is only sending data to the relay server, it can connect to the relay server without an authentication token.

The client generates a random nonce for the device and encrypts the nonce as a challenge to the relay server using the MARC4 cipher as specified in section 3.1.1.4, with the secret device key and a new random IV. The client constructs the **SecConnect** message as specified in section 2.2.1 and then sends an SSTP **Connect** command to start the device authentication sequence.

After sending the **SecConnect** message to the relay server, the client transitions into the **Connecting** state.

3.2.4.2 Higher-Layer Account Authentication Initiation

The account authentication occurs within an already established SSTP connection. The client generates a random nonce for the account and encrypts the nonce using the MARC4 cipher as specified in section 3.1.1.4 with the secret account key and a new random IV as a challenge to the relay server. The client constructs the **SecAttach** message as specified in section 2.2.6 and then sends an SSTP **Attach** command to start the account authentication sequence.

After sending the **SecAttach** message to the relay server, the client transitions into the **AttachChallenging** state.

3.2.4.3 Higher-Layer Account and Device Registration Initiation

3.2.4.3.1 New Account and Device Registration

For a new account, the client sends to a relay server the **SecAccountRegister** account-layer message within the **SecDeviceAccountRegister** message. The client retrieves the secret account key and the public key information from the current account object, and then constructs the **SecAccountRegister** message as specified in section 2.2.14. The value of the **UserPreAuthToken** in the **SecAccountRegister** message SHOULD come from one of identity objects associated with the current account object<5>. The client retrieves the secret device key and the public key information from the current device object, and then constructs the **SecDeviceAccountRegister** message as specified in section 2.2.12. The client sends an SSTP **Register** command to start the new account registration sequence.

After sending the **SecDeviceAccountRegister** message with the **SecAccountRegister** message to the relay server, the client transitions into the **DeviceAccountRegistering** state.

3.2.4.3.2 Account on New Device Registration

For an account that is on a new device, a client sends to a relay server the **SecAccountOnNewDevice** account-layer message within the **SecDeviceAccountRegister** message. The client calculates an HMAC and constructs the **SecAccountOnNewDevice** message as specified in section 2.2.15. The client constructs the **SecDeviceAccountRegister** message with the secret device key and the public key information from the current device object just as in the case of the new account registration. The client sends an SSTP **Register** command to start the account on new device registration sequence.

After sending the **SecDeviceAccountRegister** message with the **SecAccountOnNewDevice** message to the relay server, the client transitions into the **AccountOnNewDeviceRegistering** state.

3.2.4.4 Higher-Layer Identity Registration Initiation

After the client has its account successfully authenticated, it MUST register each identity belonging to that account. This is done by sending the **SecIdentityRegister** message encoded in the **AuthenticationToken** field of the SSTP **Register** command. The client retrieves the lists of active and inactive identity URLs from the current account object, and then constructs the **SecIdentityRegister** message as specified in section 2.2.17. The client MUST add every identity from the active URL list to the relay server, and MUST remove every identity from the inactive URL list from the relay server. The client sends an SSTP **Register** command to register identities.

3.2.5 Message Processing Events and Sequencing Rules

The client MUST treat any error in parsing a security message as a protocol violation, and then close the SSTP connection.

The client SHOULD follow the state machine as defined in section 3.2.1 to keep track of state transitions and MUST treat out of order messages as protocol violations.

3.2.5.1 SecConnectResponse Message

Upon receipt of this message from the relay server, the client's device transitions into the **Connected** state and does the following:

- Parses the message and extracts all fields as defined in section 2.2.2.
- Verifies the equivalence of the device nonce with the one previously sent in the **SecConnect** message.
- Decrypts the relay nonce using the MARC4 cipher as specified in section 3.1.1.4 with the secret device key and the IV.
- Calculates a HMAC using the message identifier, device URL, server certificate fingerprint and the relay nonce as specified in section 2.2.2, and verifies the HMAC in the **SecConnectResponse** message.
- If either verification fails, the client closes the connection by sending the **ConnectClose** command with the **ReasonId** field set to **DeviceAuthenticationFailed**.
- If verification is successful, the client can then proceed by sending the **ConnectAuthenticate** message (see section 2.2.5 for how to construct a **SecConnectAuthenticate** message). The client MUST save the decrypted relay nonce in the **SecConnectResponse** message for later use in the **SecAttachAuthenticate** message. After sending the **ConnectAuthenticate** message, the client transitions into the **ConnectAuthenticated** state. The client then MUST proceed to authenticate its account. See section 3.2.4.2.

3.2.5.2 SecConnectResponseDeviceRegistrationNeeded Message

Upon receipt of this message, the client transitions into the **Connected** state, and MUST go to register the device by first sending the **SecAttach** message and then the **SecDeviceAccountRegister** message to the relay server. See sections 3.2.4.2 and 3.2.4.3.

3.2.5.3 SecConnectResponseAuthenticationFailed Message

Upon receipt of this message, the client transitions into the **AuthenticationFailed** state, and SHOULD handle this failure case by sending a **ConnectClose** command to the relay server to close the connection<6>.

3.2.5.4 SecAttachResponseAuthenticationFailed Message

Upon receipt of this message, the client transitions into the **AuthenticatedFailed** state, and SHOULD handle this failure case by sending a **ConnectClose** command to the relay server to close the connection<7>.

3.2.5.5 SecAttachResponse Message

Upon receipt of this message from the relay server, the client does the following:

- Parses the message and extract all fields as defined in section 2.2.7.
- Verifies the equivalence of the account nonce with the one previously sent in the **SecAttach** message.
- Decrypts the relay nonce using the MARC4 cipher as specified in section 3.1.1.4 with the secret account key and the IV.
- Calculates a HMAC using the message identifier, account URL, relay URL, device URL and the relay nonce, as specified in section 2.2.7, and verifies the HMAC in the **SecAttachResponse** message.

  If either verification fails, the client transitions to the **AuthenticationFailed** state and SHOULD<8> close the **Attach** session by sending the **Close** command.

  If verification is successful, the client transitions to the **AttachAuthenticated** state and then proceeds by sending the **AttachAuthenticate** message. The client MUST include two relay nonces in the **AttachAuthenticate** message: the first one is the relay nonce that the client decrypted from the **SecAttachResponse** message, and the other is the relay nonce that the client has decrypted and saved from the **SecConnectResponse** message or the **SecDeviceAccountRegisterResponse** message. The client then MUST proceed with the identity registration. See section 3.2.4.4.

3.2.5.6 SecAttachResponseNewDeviceRegistrationNeeded Message

Upon receipt of this message, the client transitions into the **AccountOnNewDeviceRegistering state** and MUST proceed with the account on new device registration. See section 3.2.4.3.2 for details on "Account on New Device Registration."

3.2.5.7 SecAttachResponseAccountRegistrationNeeded Message

Upon receipt of this message, the client transitions into the **DeviceAccountRegistering** state and MUST proceed with account and device registration. See section 3.2.4.3.1 for details on "New Account and Device Registration."

3.2.5.8 SecDeviceAccountRegisterResponse Message

Upon receipt of the message, the client transitions into the **DeviceAccountRegistered** state, and does the following:

- Parses the message and extract all fields as defined in section 2.2.13.

- Calculates a HMAC using the message identifier, 0x00, account URL, device URL, server certificate fingerprint and the timestamp, as specified in section 2.2.16, and verifies the HMAC in the **SecAccountRegisterResponse** message.
- Verifies the equivalence of the device nonce received in the **SecDeviceAccountRegisterResponse** message with the one previously sent in the **SecDeviceAccountRegister** message.
- Decrypts the relay nonce in the **SecDeviceAccountRegisterResponse** message using the MARC4 cipher as specified in section 3.1.1.4, with the device key and the IV, and then saves the nonce for later use in the **SecAttachAuthenticate** message.
- Calculates a HMAC using the message identifier, account URL, device URL, server certificate fingerprint and the relay nonce, as specified in section 2.2.13, and verifies the HMAC in the **SecDeviceAccountRegisterResponse** message.

  If any of the preceding verifications fails, the client transitions to the RegistrationFailed state, and SHOULD<9> send a Close command to close the Register session.

  If all the verifications succeed, the client transitions to the DeviceAccountRegistered state, indicating that both device and account have been registered with the relay server.

3.2.5.9 RegisterResponse with no Authentication Token

If the relay server has registered the identities successfully, it responds with the SSTP **RegisterResponse** command with no authentication token. The client takes no action in this case.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

If the underlying TCP connection goes down, the client SHOULD close all sessions that were created when the connection was alive.

3.3 Server 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.

A relay server provides storage for application data queues that are uniquely identified by a tuple of **account URL**, **device URL**, and **ResourceURL**. The server stores application data sent by clients through inbound sessions to one of the queues, and then creates outbound sessions to forward application data from queues to target clients that have been authenticated. The server distinguishes between device-targeted messages and identity-targeted messages so that it forwards device-targeted messages only to authenticated devices and identity-targeted messages only to identities whose accounts have been authenticated.

To facilitate the implementation of the SSTP Security protocol, the server maintains the following types of metadata records, and creates mapping relationships among them:

Device Record: one for each device that maintains information relevant to the device. It contains the device URL, the secret device key, the device public keys, the list of account URLs on the device, and the list of application data queues that are targeted to the device. The server creates a device record each time it sees a new device, and then populates the record as relevant information becomes available. For example, when the server receives registration information from a device, it stores the secret device key and the public keys object into the device record. The server can index all device records based on device URLs, so that given a device URL, it can quickly look up the device record.

Account Record: one for each account that maintains information relevant to the account. It contains the account URL, the secret account key, the account public keys, the list of devices URLs associated with the account, and the list of identities on the account. The server creates an account record each time it sees a new account, and then populates the record as relevant information becomes available. For example, when the server receives registration information from an account, it stores the secret account key and the account public keys into the account record. The server can index all account records based on account URLs, so that given an account URL, it can quickly look up the account record.

Identity Record: one for each identity that maintains information relevant to the identity. It contains the identity URL, the identity's account URL, and the list of application data queues targeted to the identity. The server creates an identity record each time it sees a new identity, and then populates the record as relevant information becomes available. For example, when the server receives the identity registration information from an account, it stores the account URL into the device record. The server can index all identity records based on identity URLs, so that given an identity URL, it can quickly look up the identity record.

User Record: one for each user created by a management server that maintains information for the user. It contains the user identifier that is used as a pre-authentication token, and the account URL associated with the user identifier. The server creates a user record when it receives the information from the management server. The relay server can index all user records based on the user identifier, so that given a user identifier, it can quickly look up the user record.

The relay server persists these metadata records so that a client only needs to register a device and an account once for each triplet of **account URL**, **device URL**, and **relay URL**.

The relay server authenticates each and every new connection from a client as the authentications are only valid for the duration of an SSTP connection. When a client connects to its assigned relay server, the relay server authenticates the connecting device and every account on that device. When a device is authenticated successfully, the relay server can look up the corresponding device record, and then for each device-targeted queue on the device, opens an outbound session to forward any queued message to the connecting device. When an account on the device is authenticated successfully, the relay server can look up the corresponding account record, and then for each identity on the account, it finds the identity record and then opens an outbound session for each identity-targeted queue to forward any queued message to the identity.

3.3.2 Timers

The server has no timer for this security protocol, but has timers at the SSTP level. See [MS-GRVSSTP] for information.

3.3.3 Initialization

The server MUST listen on an SSTP port (2492, 443 or 80) to receive SSTP commands and security messages, and MUST initialize any newly created record to be blank for all fields.

The SSTP **Connect** command contains a field that provides a unique device URL for the connecting device. The server MUST save the device URL and use it throughout the registration and authentication exchange sequences.

The server MUST also have an X.509 v3 certificate with extensions as specified in section 3.1.1.1. The certificate binds the server's relay URL with the signature and encryption public keys from the certificate.

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Message Processing Events and Sequencing Rules

This section describes how a relay server processes and responds to each security message that it has received from a client. The server identities the type of security messages according to the type of SSTP commands that a security message is embedded in, and expects to receive only these six types of security messages from clients: **SecConnect**, **SecConnectAuthenticate**, **SecAttach**, **SecAttachAuthenticate**, **SecDeviceAccountRegister** and **SecIdentityRegister**. The server MUST treat an authentication token that cannot be parsed correctly as a protocol error and respond by sending a response message, closing the session, or closing the connection. The server MUST also check the version numbers from the message header. If the major version specified in the header is different from the server's current major version, the server SHOULD reject the message by closing the session or the connection.

The relay server MUST treat any error in parsing a security message as a protocol violation, and then close the session or the SSTP connection.

The server MAY<10> treat a security message received out of order as a protocol violation and then close the session or the SSTP connection.

3.3.5.1 SecConnect

The relay server MUST receive the **SecConnect** message in an SSTP **Connect** command, which MUST be the first command that the server receives for a new SSTP connection.

The **SecConnect** message contains an HMAC and an encrypted device nonce. See section 2.2.1 for the **SecConnect** message format. The client sends the encrypted nonce as a challenge to the relay server, and the HMAC to protect the message integrity.

Upon receiving the **SecConnect** message, the server does the following:

1. Parses the message and extract all fields as defined in section 2.2.1.
2. If the server finds incorrect version numbers in the **SecConnect** security message header, or encounters any error in parsing the message, it SHOULD<11> embed a **SecConnectResponseAuthenticationFailed** security message in a **ConnectResponse** command, and set the **ResponseID** field in the **ConnectResponse** to **AuthenticationFailed** (see [MS-GRVSSTP] for more details), and then send the **ConnectResponse** to the client.
3. Retrieves the device URL for the connecting device, and then uses it to locate the device record. If the server cannot find the device record, or found the record without the secret device key, then the server knows that the device has not registered. In this case, the server sends an SSTP **ConnectResponse** command to the client by embedding a **SecConnectResponseDeviceRegistrationNeeded** security message within the command and setting the **ResponseID** field in the **ConnectResponse** command to **Ok**.
4. Ensures that the device has at least one account on the device. If the server sees that the device record has an empty list of account URLs, it sends an SSTP **ConnectResponse** command to the client by embedding a **SecConnectResponseAuthenticationFailed** security message within the command.

5. Retrieves the device key from the device record, decrypts the nonce in the **SecConnect** message using the secret device key and the IV with the MARC4 cipher as specified in section 3.1.1.4.

6. Calculates a HMAC with the message identifier, device URL, server certificate fingerprint and the device nonce as specified in section 2.2.1, and verifies the HMAC.

   If the HMAC verification fails, the server sends an SSTP **ConnectResponse** command to the client by embedding a **SecConnectResponseAuthenticationFailed** security message within the command and setting the **ResponseID** field in the **ConnectResponse** command to **AuthenticationFailed**.

   If the **SecConnect** message is processed successfully, generates a new random nonce and a new random IV, encrypts the nonce as a challenge to the client using the secret device key and the IV with the MARC4 cipher as specified in section 3.1.1.4, and then creates a **SecConnectResponse** message as specified in section 2.2.2, and sends it along with a **ConnectResponse** command to the client, with the **ResponseID** field in the **ConnectResponse** command set to **OK**.

3.3.5.2 SecConnectAuthenticate

The relay server MUST receive the **SecConnectAuthenticate** message in an SSTP **ConnectAuthenticate** command after the server has previously sent a **SecConnectResponse** message to the client. If the server receives the **SecConnectAuthenticate** message out of order, it MUST send a **ConnectClose** command with the **ReasonId** field set to **ProtocolError** to terminate the SSTP connection.

The **SecConnectAuthenticate** message contains the decrypted relay nonce that the server has generated and sent as a challenge to the client in the **SecConnectResponse** message. Upon receiving the **SecConnectAuthenticate** message, the server does the following:

1. Parses the message and extract all fields as defined in section 2.2.5.

   If the server finds incorrect version numbers in the **SecConnectAuthenticate** message header, or encounters any error in parsing the message, it sends a **ConnectClose** command to the client, with the **ReasonId** field set to **StaleConnectAuthenticate** (see [MS-GRVSSTP] for more details).

2. Verifies the nonce contained in the message against the one it generated earlier. If they are the same, then the server has successfully authenticated the device. The server now creates an outbound session for every device-targeted queue in the device record, and then forwards any queued message to the device.

   If the nonce verification fails, the server sends to the client an SSTP **ConnectClose** command with the **ReasonId** field set to **StaleConnectAuthenticate** to close the SSTP connection.

3.3.5.3 SecAttach

The relay server MUST receive the **SecAttach** message in an SSTP **Attach** command. The client sends the message to authenticate its account. The **SecAttach** message contains an HMAC and an encrypted account nonce. See section 2.2.6 for the **SecAttach** message format. The client sends the encrypted nonce as a challenge to the relay server, and the HMAC to protect the message integrity.

Upon receiving the **SecAttach** message, the server does the following:

1. Parses the message and extract all fields as defined in section 2.2.6.

2. If the server finds incorrect version numbers in the **SecAttach** security message header, or encounters any error in parsing the message, it embeds a **SecAttachResponseAuthenticationFailed** security message in a **AttachResponse** command

with the **ResponseID** field set to **AttachRejected** (see [MS-GRVSSTP] for more details), and then sends the **AttachResponse** to the client.

3. Retrieves the account URL from the SSTP **Attach** command, and uses it to locate the account record. If the server cannot find the account record, or found the record with no secret account key, then the server knows that the account has not registered. In this case, the server sends an SSTP **AttachResponse** command to the client by embedding a **SecAttachResponseAccountRegistrationNeeded** security message within the command and setting the **ResponseID** field in the **AttachResponse** command to **Ok**.

4. If the account key is found from the account record, verifies that the account is on the connecting device by checking if the account record contains the device URL. If the device is not in the account record, then this account is on a new device and is connecting to the server for the first time from the device. In this case, the server sends an **AttachResponse** command to the client by embedding a **SecAttachResponseNewDeviceRegistrationNeeded** message with the **ResponseID** field in the **AttachResponse** command set to **AwaitingRegistrater**.

5. Retrieves the account key from the account record, decrypts the nonce in the **SecAttach** message using the secret account key and the IV with the MARC4 cipher as specified in section 3.1.1.4.

6. Calculates a HMAC with the message identifier, account URL, relay URL, device URL, and the account nonce as specified in section 2.2.6, and verifies the HMAC.

   If the HMAC verification fails, the server sends an SSTP **AttachResponse** command to the client by embedding a **SecAttachResponseAuthenticationFailed** security message within the command and setting the **ResponseID** field in the **AttachResponse** command to **AccountUnknown**.

7. If the **SecAttach** message is processed successfully, generates a new random nonce and a new random IV, encrypts the nonce as a challenge to the client using the account key and the new IV with the MARC4 cipher as specified in section 3.1.1.4, and then creates a **SecAttachResponse** message as specified in section 2.2.7 and sends it along with a **AttachResponse** command to the client, with the **ResponseID** field in the **AttachResponse** command set to **Ok**.

3.3.5.4 SecAttachAuthenticate

The relay server MUST receive the **SecAttachAuthenticate** message in an SSTP **AttachAuthenticate** command after the server has sent a **SecAttachResponse** message earlier to the client. If the server receives the **SecAttachAuthenticate** message out of order, it MUST send a **Close** command to close the attach session indicated by the **EventId** field of the **AttachAuthenticate** command.

The **SecAttachAuthenticate** message contains the decrypted relay nonce that the server has generated and sent as a challenge to the client in the **SecAttachResponse** message. It also contains the decrypted relay nonce that the server had generated and sent during the device authentication or registration. Upon receiving the **SecAttachAuthenticate** message, the server does the following:

1. Parses the message and extract all fields as defined in section 2.2.11.

   If the server finds incorrect version numbers in the **SecAttachAuthenticate** security message header, or encounters any error in parsing the message, it sends a **Close** command with the **ReasonId** field set to **StaleAttachAuthenticate** (see [MS-GRVSSTP] for more details), to close the **Attach** session.

2. Verifies the account nonce against the one that the server generated from the **SecAttachResponse** message, and the device nonce against the one that the server generated from the **SecConnectResponse** message or the **SecDeviceAccountRegisterResponse** message. If they both match<12>, then the server has successfully authenticated the account. The server now locates identity records for all identity URLs in the account record, creates an

outbound session for every identity-targeted queue in each identity record, and then forwards any queued message to the corresponding identity on the connecting device.

If the nonce verifications fail, the server sends to the client an SSTP **AttachResponse** command by embedding the **SecAttachResponseAuthenticationFailed** message, with the **ResponseID** field in the **AttachResponse command** set to **AttachRejected**.

3.3.5.5 SecDeviceAccountRegister

The **SecDeviceAccountRegister** message contains information for device registration, and an account layer message. The account layer message can be either **SecAccountRegister** or **SecAccountOnNewDeviceRegister**, depending whether the client wants to register a new account or an account on a new device. The server identifies the type of the account layer messages by checking the message identifier in the message header. If the relay server cannot parse the account layer message, it SHOULD<13> send an SSTP **Close** command to close the session designated by the **EventID** field in the **Register** command.

The following sections describe how the relay server processes the **SecDeviceAccountRegister** message according to the type of the account layer messages in the **SecDeviceAccountRegister** message.

3.3.5.5.1 New Account Registration

The **SecDeviceAccountRegister** message with a **SecAccountRegister** message is for registering a new account and the connecting device.

The **SecAccountRegister** message contains the account registration information that includes the encrypted secret account key, the account's public keys object, an optional pre-authentication token, and a signature of the message signed with the account's signature private key using RSA algorithm as defined in [PKCS1]. See section 2.2.14 for the **SecAccountRegister** message format.

The **SecDeviceAccountRegister** message contains the device registration information in addition to the account layer message. The message provides an account URL, the server certificate fingerprint, the encrypted secret device key, the device's public keys object, a device nonce, and a signature of the message signed with the device's signature private key using RSA algorithm as defined in [PKCS1]. See section 2.2.12 for the **SecDeviceAccountRegister** message format.

Upon receipt of this message, the relay server does the following:

1. Parses the message and extract all fields as defined in section 2.2.12.
2. Calculate the server certificate fingerprint as specified in section 3.1.1.2, and verifies the server certificate fingerprint to make sure that the message is intended for itself.
3. Retrieves the device URL to locate the device record or creates a new record if none is found, and then uses the Event identifier from the **Register** command to look up the matching session if one exists, or create a new session if no existing one is found. The server creates an account record using the account URL from the message.
4. If the pre-authentication token is present, uses it to locate the matching user record. If no user record is found or the account URL from the matching user record does not match the account URL in the account record, the server sends an SSTP **Close** command to the client to close the session, with the **ReasonId** field in the **Close** command set to **UserAuthenticationFailed**.
5. Validates public keys objects for the device and account to be sure that the server supports the signature and encryption algorithms specified in the objects.

6. Calculates two SHA-1 hashes as specified in sections 2.2.12 and section 2.2.14, and verifies the two signatures to be sure that the message came from the device and account with corresponding signature public key, using RSA algorithm, as defined in [PKCS1].

7. Decrypts the secret account and device keys from the message using the server's encryption private key, stores the device public keys object and the device key in the device record, and then stores the account public keys object and the account key in the account record.

8. If the account has already registered with the relay server (for example, by another instance of the account), verifies that the account public keys object and the account key match the ones stored in the account record.

9. If the connecting device has already registered with the relay server (for example, by a different account sharing the same device), verifies that the device public keys object and the device key match the ones stored in the device record.

   If any of the preceding verification fails, the server sends an SSTP **Close** command to the client to close the session identified by the **EventId** field from the **Register** command, with the **ReasonId** field in the **Close** command set to **DeviceAuthenticationFailed**.

   If the relay server encounters no error while processing the **SecDeviceAccountRegister** message, sends an SSTP **RegisterResponse** command to the client by embedding a **SecDeviceAccountRegisterResponse** message with a **SecAccountRegisterResponse** message as the account layer message. See sections 2.2.13 and 2.2.16 on the **SecDeviceAccountRegisterResponse** and **SecAccountRegisterResponse** message formats.

   If the server responded earlier to the **SecAttach** message with a **SecAttachResponseAccountRegistrationNeeded** message, it MUST now send an SSTP **AttachResponse** command to the client to continue with the client on the account authentication process by embedding a **SecAttachResponse** message, with the **ResponseID** field in the **AttachResponse** command set to **Ok**.

3.3.5.5.2 Account- on-New-Device Registration

The **SecDeviceAccountRegister** message with a **SecAccountOnNewDevice** message is for registering an account on a new device.

The **SecAccountOnNewDevice** message contains an HMAC. See section 2.2.15 for the **SecAccountOnNewDevice** message format.

The **SecDeviceAccountRegister** message contains the device registration information in addition to the account layer message. The message provides an account URL, the server certificate fingerprint, the encrypted secret device key, the device's public keys object, a device nonce, and a signature of the message signed with the device's signature private key using RSA algorithm as defined in [PKCS1]. See section 2.2.12 for the **SecDeviceAccountRegister** message format.

Upon receipt of this message, the relay server does the following:

1. Parses the message and extract all fields as defined in section 2.2.12.

2. Calculate the server certificate fingerprint as specified in section 3.1.1.2, and verifies the server certificate fingerprint to make sure that the message is intended for itself. See section 3.1.1.2 for how to calculate the server certificate fingerprint.

3. Retrieves the device URL to locate the device record or creates a new record if none is found, and then uses the Event identifier from the **Register** command to look up the matching session if one exists, or create a new session if no existing one is found. The server also looks up the account record using the account URL from the message.

4. Validates the account's public keys object to be sure that the server supports the signature and encryption algorithms specified in the objects.

5. Calculates a SHA-1 hash as specified in section 2.2.12, and verifies the signature to be sure that the message came from the connecting device.

6. Calculates a HMAC with the message identifier, account URL, device URL, server certificate fingerprint and the timestamp as specified in section 2.2.15, and verifies the HMAC in the **SecAccountOnNewDevice** message using the secret account key stored in the account record.

7. Decrypts the secret device key from the message using the server's encryption private key, and then stores the device public keys object and the device key to the device record if this is the first time that the device is registered with the relay server.

8. If the device has already registered with the relay server (for example, by a different account sharing the same device), verifies that the device public keys object and the device key match the ones stored in the device record.

9. If any of the preceding verification fails, the server sends an SSTP **Close** command to close the session identified by the **EventId** field from the **Register** command, with the **ReasonId** field in the **Close** command set to **DeviceAuthenticationFailed** or **UserAuthenticationFailed**, depending where the failure occurred in device layer or account layer.

10. If the relay server encounters no error while processing the **SecDeviceAccountRegister** message, sends an SSTP **RegisterResponse** command to the client by embedding a **SecDeviceAccountRegisterResponse** message along with a **SecAccountRegisterResponse** account layer message. See sections 2.2.13 and 2.2.16 on the **SecDeviceAccountRegisterResponse** and **SecAccountRegisterResponse** message formats.

11. If the server responded earlier to the **SecAttach** message with a **SecAttachResponseNewDeviceRegistrationNeeded** message, it MUST now send to the client an SSTP **AttachResponse** command by embedding a **SecAttachResponse** message to continue with the client on the account authentication process, with the **ResponseID** field in the **AttachResponse** command set to **Ok**.

3.3.5.6 SecIdentityRegister

The relay server MUST receive the **SecIdentityRegister** message in an SSTP **Register** command. The **SecIdentityRegister** message is used to update the server about identities associated with an authenticated account. The **SecIdentityRegister** message contains the account URL, a HMAC and a list of identity URLs to update on the relay server.

Upon receipt of this message, the relay server does the following:

1. Parses the message and extract all fields as defined in section 2.2.17.

2. Retrieves the device URL to locate the device record, and uses the account URL to locate the account record.

3. Calculates a HMAC with the message identifier, account URL, relay URL, device URL and the timestamp as specified in section 2.2.17, and verifies the HMAC using the secret account key stored in the account record. If either of the record is not found or the verification fails, the server sends an SSTP **Close** command with the **SessionId** field set to the same value as the **EventId** field from the **Register** command, with the **ReasonId** field in the **Close** command set to **ProtocolError**.

4. Retrieves from the message the list of identity URLs to add to the relay server, and then adds every URL to the account record, ignoring any duplicate.

5. Retrieves from the message the list of identity URLs to remove from the relay server, and then removes every URL from the list of identities in the account record.

6. Sends to the client an SSTP **RegisterResponse** command with no authentication token.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

If the underlying TCP connection goes down, the server SHOULD close all outbound and inbound sessions that were created when the connection was alive.

4 Protocol Examples

This section provides over-the-wire traces and annotations of the SSTP commands and security messages that a client and relay server exchange during registration and authentication for three common scenarios. The first scenario is for a newly created account that connects to an assigned relay server for the first time. The second scenario is for an account that has already registered with its assigned relay server and is reconnecting to the server for the first time from a new device. The third scenario is for a client that is reconnecting to its assigned relay server after its device and account have both been registered successfully with the relay server.

4.1 Registration and Authentication for a New Account

In this common scenario, a client is connecting to an assigned relay server for the first time for a new account. After the client has created and then configured an account, it first exchanges **Connect**/**ConnectResponse** commands with the server to establish an SSTP connection with authentication tokens. Because the server has no information about the connecting device, it responds with a **ConnectResponse** command with the **SecConnectResponseDeviceRegistrationNeeded** security message. The client then sends an **Attach** command to see if it can attach the account to the server. This **Attach** command results in an **Attach** session being created on the existing SSTP connection. Because the server has no information about this new account, it responds with the **SecAttachResponseAccountRegistrationNeeded** message. The client, after receiving the message about the account not being registered yet, exchanges **Register**/**RegisterResponse** commands with the server to register keys for the client's device and account. The server, after receiving the account key, also sends an **AttachResponse** command to continue with the account authentication initiated by the client's **Attach** command. The client then completes the account authentication by sending an **AttachAuthenticate** command in response to the **AttachResponse** from the server. The server then sends a **Close** command to close the Attach session to complete the registration and authentication sequence.

After the new account is successfully registered and authenticated with the server, the client can now exchange **Register**/**RegisterResponse** with the server to register identities associated with the new account.

The following diagram shows a sequence of SSTP commands and security messages that the client and the server exchange to complete the registration and authentication for this scenario. In this and the following diagrams, the embedding of security messages in an SSTP command is shown by placing the security messages within [] brackets, and the embedding of account layer messages in a security message is shown by placing the account layer message after a colon :. The letters within () parenthesis denote an SSTP session in which the SSTP command is sent between a client and a server.

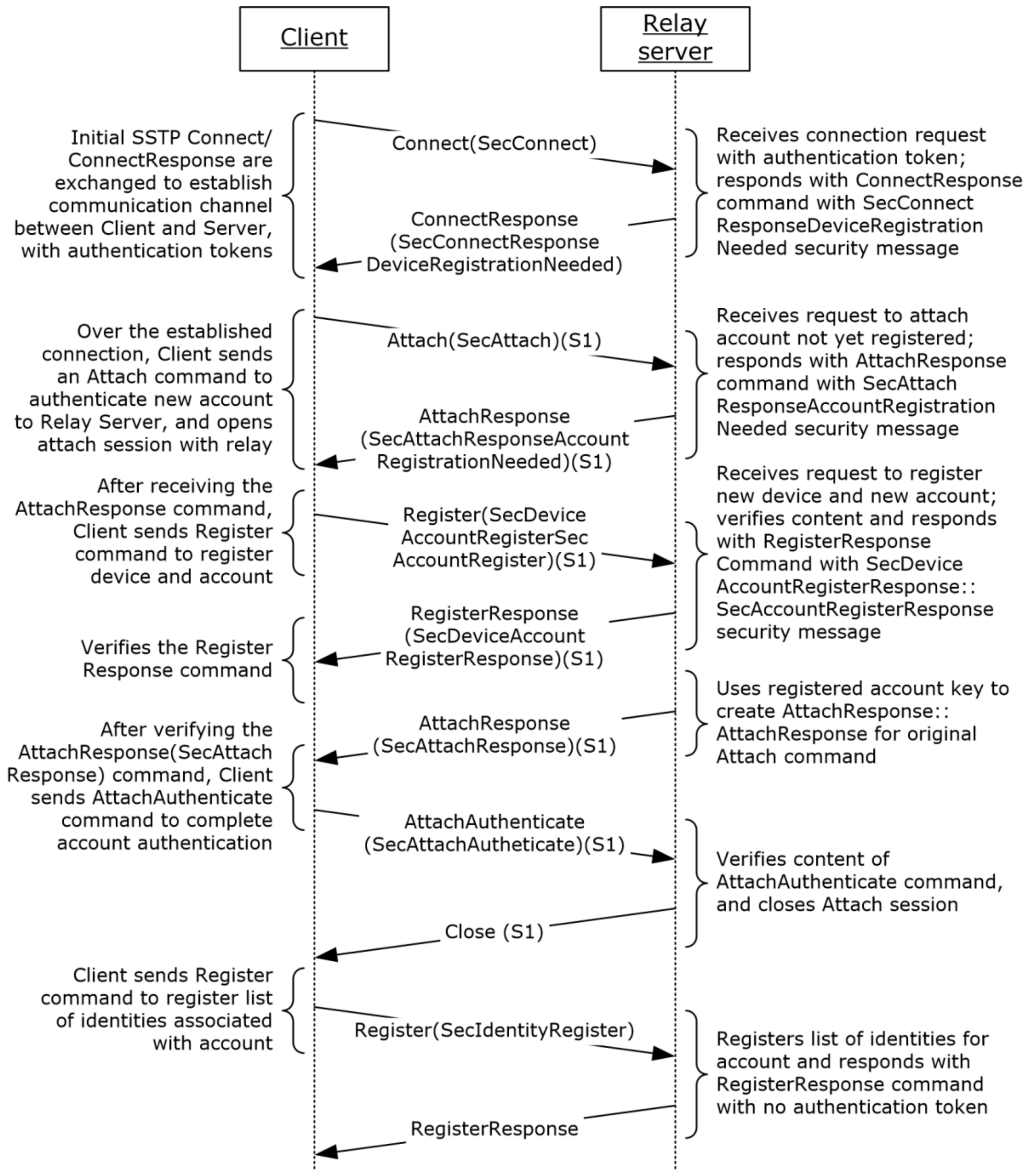
Client
Relay server
Initial SSTP Connect/ ConnectResponse are exchanged to establish communication channel between Client and Server, with authentication tokens
Connect(SecConnect)
ConnectResponse (SecConnectResponse DeviceRegistrationNeeded)
Receives connection request with authentication token; responds with ConnectResponse command with SecConnect ResponseDeviceRegistration Needed security message
Over the established connection, Client sends an Attach command to authenticate new account to Relay Server, and opens attach session with relay
Attach(SecAttach)(S1)
AttachResponse (SecAttachResponseAccount RegistrationNeeded)(S1)
Receives request to attach account not yet registered; responds with AttachResponse command with SecAttach ResponseAccountRegistration Needed security message
After receiving the AttachResponse command, Client sends Register command to register device and account
Register(SecDevice AccountRegisterSec AccountRegister)(S1)
Receives request to register new device and new account; verifies content and responds with RegisterResponse Command with SecDevice AccountRegisterResponse:: SecAccountRegisterResponse security message
Verifies the Register Response command
RegisterResponse (SecDeviceAccount RegisterResponse)(S1)
Uses registered account key to create AttachResponse:: AttachResponse for original Attach command
AttachResponse (SecAttachResponse)(S1)
After verifying the AttachResponse(SecAttach Response) command, Client sends AttachAuthenticate command to complete account authentication
AttachAuthenticate (SecAttachAutheticate)(S1)
Verifies content of AttachAuthenticate command, and closes Attach session
Close (S1)
Client sends Register command to register list of identities associated with account
Register(SecIdentityRegister)
Registers list of identities for account and responds with RegisterResponse command with no authentication token
RegisterResponse

Figure 7: Message exchange sequence for a new account

4.1.1 Connect

The following shows the trace of the **Connect** command with the **SecConnect** security message that the client sends to the relay server (for SSTP **Connect** command fields, see [MS-GRVSSTP]):

```
0000  01 bb 00 01 05 00 67 72 6f 6f 76 65 44 4e 53 3a  ......grooveDNS:
0010  2f 2f 72 65 6c 61 79 2e 63 6f 6e 74 6f 73 6f 2e  //relay.contoso.
0020  63 6f 6d 00 01 64 70 70 3a 2f 2f 2f 37 67 77 73  com..dpp:///7gws
0030  39 6b 68 70 65 74 39 7a 34 65 7a 61 6a 76 6e 68  9khpet9z4ezajvnh
```

```
0040  62 35 64 39 66 70 6d 63 77 71 72 6a 76 33 77 7a  b5d9fpmcwqrjv3wz
0050  65 7a 32 00 4d 00 01 03 01 18 00 6a 2e 32 1c 7a  ez2.M......j.2.z
0060  29 0a 27 16 3d 2b 67 a7 00 f9 7e 1b 70 a5 7c cc  ).'.=+g...~.p.|.
0070  4d f8 f9 14 00 c6 8d 0b d9 70 66 8d 39 a0 85 81  M........pf.9...
0080  72 20 0d 09 07 83 76 a0 85 18 00 2c ef d1 93 1e  r ....v....,....
0090  fb 46 4b 49 ed 18 22 0e cb dc 5a 29 44 b4 e1 30  .FKI.."...Z)D..0
00a0  ea a1 c9 47 72 6f 6f 76 65 20 43 6c 69 65 6e 74  ...Groove Client
00b0  20 34 2e 32 20 32 36 32 33 00 00                  4.2 2623..

- SSTP: CONNECT,
  - CONNECT: SSTP Version: 1.5, ConnectFlags: 0x0, TargetDeviceURL: 
grooveDNS://relay.contoso.com
     CommandID: 1 (CONNECT) (0x1)
     CommandLength: 187 (0xBB)
     MajorVersion: 1 (0x1)
     MinorVersion: 5 (0x5)
   - ConnectFlags: .......
      NotUsed1:             (0.......) Not Used
      NotUsed2:             (.0......) Not Used
      NotUsed3:             (..0.....) Not Used
      NotUsed4:             (...0....) Not Used
      NotUsed5:             (....0...) Not Used
      NotUsed6:             (.....0..) Not Used
      NotUsed7:             (......0.) Not Used
      UpgradeRequired:      (.......0) Default
     TargetDeviceURL: grooveDNS://relay.contoso.com
     NumSourceDeviceURLs: 1 (0x1)
     SourceDeviceURL: dpp:///7gws9khpet9z4ezajvnhb5d9fpmcwqrjv3wzez2
   - AuthenticationToken: SecConnect (Length: 77)
      AuthenticationTokenLength: 77 (0x4D)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 1 (0x1)
    - SecConnect:
       IVLength: 24 (0x18)
       IV: Binary Large Object (24 Bytes)
       HMACLength: 20 (0x14)
       HMAC: Binary Large Object (20 Bytes)
       EncryptedDeviceNonceLength: 24 (0x18)
       EncryptedDeviceNonce: Binary Large Object (24 Bytes)
     PeerProductVersion: Groove Client 4.2 2623
     PeerProductCapabilities:
```

The client with device URL: dpp:///7gws9khpet9z4ezajvnhb5d9fpmcwqrjv3wzez2, connects to the relay server with relay URL: grooveDNS://relay.contoso.com. The client generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: 6A 2E 32 1C 7A 29 0A 27 16 3D 2B 67 A7 00 F9 7E 1B 70 A5 7C CC 4D F8 F9) to produce the encrypted device nonce (2C EF D1 93 1E FB 46 4B 49 ED 18 22 0E CB DC 5A 29 44 B4 E1 30 EA A1 C9). The client generated the HMAC (C6 8D 0B D9 70 66 8D 39 A0 85 81 72 20 0D 09 07 83 76 A0 85) according to section 2.2.1.

4.1.2 ConnectResponseDeviceRegistrationNeeded

The following shows an over-the-wire trace of the **ConnectResponse** command with the **SecConnectResponseDeviceRegistrationNeeded** security message:

```
0000  02 44 00 01 05 00 03 00 01 03 0a 03 47 72 6f 6f  .C..........Groo
0010  76 65 20 52 65 6c 61 79 20 31 32 2e 30 20 31 35  ve Relay 12.0 15
0020  30 31 00 00 01 67 72 6f 6f 76 65 44 4e 53 3a 2f  01...grooveDNS:/
0030  2f 72 65 6c 61 79 2e 63 6f 6e 74 6f 73 6f 2e 63  /relay.contoso.c
0040  6f 6d 00 00                                      om..

- SSTP: CONNECT_RESPONSE,
  - CONNECT_RESPONSE: SSTP Version: 1.5, Connect Response: SSTP_CONNECT_RESPONSE_ID_OK,
Flags: 0x3
     CommandID: 2 (CONNECT_RESPONSE) (0x2)
     CommandLength: 44 (0x2C)
     MajorVersion: 1 (0x1)
     MinorVersion: 5 (0x5)
     ResponseID: 0 (SSTP_CONNECT_RESPONSE_ID_OK) (0x0)
   - AuthenticationToken: SecConnectResponseDeviceRegistrationNeeded (Length: 3)
      AuthenticationTokenLength: 3 (0x3)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 10 (0xA)
      SecConnectResponseDeviceRegistrationNeeded:
   - ConnectResponseFlags: MS.....
      NotUsed1:              (0.......) Not Used
      NotUsed2:              (.0......) Not Used
      NotUsed3:              (..0.....) Not Used
      NotUsed4:              (...0....) Not Used
      NotUsed5:              (....0...) Not Used
      NotUsed6:              (.....0..) Not Used
      SingleHopFanout:       (......1.) SingleHop Fanout Supported
      MultiDropFanout:       (.......1) MultiDrop Fanout Supported
     PeerProductVersion: Groove Relay 12.0 1501
     PeerProductCapabilities:
     NumberOfTargetDeviceURLs: 1 (0x1)
     TargetDeviceURL: grooveDNS://relay.contoso.com
   - RedirectMapEntries:
      RedirectMapEntries: 0 (0x0)
```

See [MS-GRVSSTP] for details on the **ConnectResponse** command.

4.1.3 Attach

The following shows an over-the-wire trace of the **Attach** command that the client sends to authenticate an account:

```
0000  08 ad 00 0b 00 00 00 67 72 6f 6f 76 65 44 4e 53  .......grooveDNS
0010  3a 2f 2f 72 65 6c 61 79 2e 63 6f 6e 74 6f 73 6f  ://relay.contoso
0020  2e 63 6f 6d 00 67 72 6f 6f 76 65 41 63 63 6f 75  .com.grooveAccou
0030  6e 74 3a 2f 2f 6e 67 6d 6a 77 62 61 7a 6d 39 78  nt://ngmjwbazm9x
0040  69 7a 34 65 74 73 36 35 72 72 34 63 39 6b 62 78  iz4ets65rr4c9kbx
0050  6b 78 70 68 64 77 36 67 70 6b 36 73 40 00 4d 00  kxphdw6gpk6s@.M.
0060  01 04 01 18 00 ce d0 75 0e 87 0e 20 d2 58 91 80  .......u... .X..
0070  f7 c4 a5 43 65 8c 45 85 74 cb d5 06 ab 14 00 fa  ...Ce.E.t.......
0080  79 bf b1 ef 3f 33 1c 58 05 98 f8 df 1b 0f 5e 70  y...?3.X......^p
0090  f7 74 9b 18 00 61 9b 6a c5 6d c6 c7 f2 8b b7 66  .t...a.j.m.....f
00a0  cf b4 f5 5f 5b ae ec 13 fe d7 ff a8 b8           ..._[........

- SSTP: ATTACH,
  - ATTACH: Event ID: 0xb, Resource URL: grooveDNS://relay.contoso.com
     CommandID: 8 (ATTACH) (0x8)
     CommandLength: 172 (0xAD)
     EventID: 11 (0xB)
     ResourceURL: grooveDNS://relay.contoso.com
     IdentityURL: grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@
   - AuthenticationToken: SecAttach (Length: 77)
      AuthenticationTokenLength: 77 (0x4D)
      MajorVersion: 1 (0x1)
      MinorVersion: 4 (0x4)
      MessageID: 1 (0x1)
    - SecAttach:
       IVLength: 24 (0x18)
       IV: Binary Large Object (24 Bytes)
       HMACLength: 20 (0x14)
       HMAC: Binary Large Object (20 Bytes)
```

```
          EncryptedAccountNonceLength: 24 (0x18)
          EncryptedAccountNonce: Binary Large Object (24 Bytes)
```

The client sends this **Attach** command to authenticate an account represented by account URL: grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@. The client generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret account key (24 bytes) and the IV (24 bytes: ce d0 75 0e 87 0e 20 d2 58 91 80 f7 c4 a5 43 65 8c 45 85 74 cb d5 06 ab) to produce the encrypted account nonce (61 9b 6a c5 6d c6 c7 f2 8b b7 66 cf b4 f5 5f 5b ae ec 13 fe d7 ff a8 b8). The client generated the HMAC (fa 79 bf b1 ef 3f 33 1c 58 05 98 f8 df 1b 0f 5e 70 f7 74 9b) using the account URL, relay URL and device URL according to section 2.2.6.

4.1.4 AttachResponseAccountRegistrationNeeded

The following is an over-the-wire trace of the **AttachResponse** command with the **SecAttachResponseAccountRegistrationNeeded** message that the relay sends back in response to the client's **Attach** command:

```
0000  09 0d 00 0b 00 00 00 03 03 00 01 03 0a 07 08 00  ................
0010  01 00 00 00 00                                   .....

- SSTP: ATTACH_RESPONSE, OPEN_RESPONSE,
  - ATTACH_RESPONSE: Event ID: 0xb, ResponseID: SSTP_ATTACH_RESPONSE_IDAWAITING_REGISTER
     CommandID: 9 (ATTACH_RESPONSE) (0x9)
     CommandLength: 13 (0xD)
     EventID: 11 (0xB)
     ResponseId: 3 (SSTP_ATTACH_RESPONSE_IDAWAITING_REGISTER)
   - AuthenticationToken: SecAttachResponseAccountRegistrationNeeded (Length: 3)
      AuthenticationTokenLength: 3 (0x3)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 10 (0xA)
      SecAttachResponseAccountRegistrationNeeded:
  + OPEN_RESPONSE: Session ID: 0x1, Response: SSTP_OPEN_RESPONSE_ID_OK
```

4.1.5 Register for Device and Account Registration

The following is an over-the-wire trace of the **Register** command that the client sends to register the client's device and account when using the ElGamal encryption:

```
0000        0b 3e 0c 0b 00 00 00 11 0c 01 03 04 d1 14      ..............
000e        91 47 67 72 6f 6f 76 65 41 63 63 6f 75 6e      .GgrooveAccoun
001c        74 3a 2f 2f 6e 67 6d 6a 77 62 61 7a 6d 39      t://ngmjwbazm9
002a        78 69 7a 34 65 74 73 36 35 72 72 34 63 39      xiz4ets65rr4c9
0038        6b 62 78 6b 78 70 68 64 77 36 67 70 6b 36      kbxkxphdw6gpk6
0046        73 40 00 14 00 a9 7a de 47 6e 85 32 3b 78      s@....z.Gn.2;x
0054        7b 6f e9 56 b0 f6 2c 88 b5 82 24 80 01 f2      {o.V..,...$...
0062-01db   ... encrypted device key (384 bytes) ...
01dc        6f 71 4b d3 d2 e9 05 01 04 04 80 01 d4 e7      oqK...........
01ea-0363   ... encrypted account key (384 bytes) ...
0364        87 47 3f 74 00 01 76 8a 6b 56 b8 23 57 d9      .G?t..v.kV.#W.
0372-045f   ... signature (256 bytes total) ...
0460        bd 75 64 10 e6 31 e5 eb 16 15 38 03 52 53      .ud..1....8.RS
046e        41 00 45 4C 47 41 4D 41 4C 00 52 53 41 00      A.ELGAMAL.RSA.
047c        44 48 00 0C 01 00 00 30 82 01 08 02 82 01      DH.....0......
048a-0585   ... signature public key ( 268 bytes) ...
0586        97 3b e8 f2 ad a3 02 01 11 11 02 00 00 30      .;...........0
059e-0799      ... encryption public key (528 bytes) ...
079a        da 87 9b 24 4e cf eb 24 85 16 01 00 00 34      ...$N..$.....4
0798        44 36 44 39 35 44 39 2d 30 34 31 32 2d 34      D6D95D9-0412-4
07b6        34 42 37 2d 41 41 38 42 2d 34 46 38 46 31      4B7-AA8B-4F8F1
07c4        45 31 43 34 39 37 33 00 01 00 00 00 01 33      E1C4973......3
07d2-08c0   ... signature (256 bytes total) ...
08ce        50 34 00 37 03 52 53 41 00 45 4c 47 41 4d      P4.7.RSA.ELGAM
08dc        41 4c 00 52 53 41 00 44 48 00 0c 01 00 00      AL.RSA.DH.....
08ea-09f3   ... signature public key (268 bytes) ...
09f4        01 11 10 02 00 00 30 82 02 0c 02 82 01 01      ......0.......
0a02-0c07   ... encryption public key (528 bytes) ...
0c08        90 4f 18 00 7d 4e a9 fc 3a 71 15 93 49 15      .O..}N..:q..I.
0c16        76 91 84 d4 08 00 23 cf 69 30 97 e5 14 55      v.....#.i0...U
0c24        18 00 ca 97 b0 e6 df 91 05 a1 34 9f c9 89      ..........4...
0c32        21 42 a7 40 90 df 29 8a f3 80 a7 42            !B.@..)....B

- MSGRVSSTP: REGISTER,
  - REGISTER: Event ID: 0xb
     CommandID: 11 (REGISTER) (0xB)
     CommandLength: 3134 (0xC3E)
     EventID: 11 (0xB)
   - RegistrationToken: SSTPSecDeviceAccountRegister (Length: 3125)
       RegistrationTokenLength: 3125 (0xC35)
       MajorVersion: 1 (0x1)
       MinorVersion: 3 (0x3)
       MessageID: 4 (0x4)
     - SSTPSecDeviceAccountRegister:
         Timestamp: 1200690385 (0x479114D1)
         AccountURL: grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@
         FingerprintLength: 20 (0x14)
         Fingerprint: Binary Large Object (20 Bytes)
         EncryptedRelayDeviceKeyLength: 384 (0x180)
         EncryptedRelayDeviceKey: Binary Large Object (384 Bytes)
       - AccountLayerMessage: SSTPSecAccountRegister (Length: 1477)
           AuthenticationTokenLength: 1477 (0x5E9)
           MajorVersion: 1 (0x1)
           MinorVersion: 4 (0x4)
           MessageID: 4 (0x4)
         - SSTPSecAccountRegister:
             EncryptedRelayAccountKeyLength: 384 (0x180)
             EncryptedRelayAccountKey: Binary Large Object (384 Bytes)
             SignatureLength: 256 (0x100)
             Signature: Binary Large Object (256 Bytes)
             AccountPublicKeysObjectLength: 824 (0x338)
             AccountPublicKeysObject: Binary Large Object (824 Bytes)
                  SignatureAlgorithmName: RSA
                  EncryptionAlgorithmName: ELGAMAL
                  SignatureKeyAlgorithmName: RSA
                  EncryptionKeyAlgorithmName: DH
                  SignaturePublicKeyLength: 268 (0x0000010C)
                  SignaturePublicKey: Binary Large Object (268 Bytes)
                  EncryptionPublicKeyLength: 529 (0x00000211)
                  EncryptionPublicKey: Binary Large Object (529 Bytes)
             Reserved1: 1 (0x1)
             Reserved2: (1 Bytes)
             UserPreAuthToken: 4D6D95D9-0412-44B7-AA8B-4F8F1E1C4973
         Reserved1: 1 (0x1)
         Reserved2: (1 Bytes)
         SignatureLength: 256 (0x100)
         Signature: Binary Large Object (256 Bytes)
         DevicePublicKeysObjectLength: 823 (0x337)
         DevicePublicKeysObject: Binary Large Object (823 Bytes)
             SignatureAlgorithmName: RSA
             EncryptionAlgorithmName: ELGAMAL
             SignatureKeyAlgorithmName: RSA
             EncryptionKeyAlgorithmName: DH
             SignaturePublicKeyLength: 268 (0x0000010C)
             SignaturePublicKey: Binary Large Object (268 Bytes)
             EncryptionPublicKeyLength: 528 (0x00000210)
             EncryptionPublicKey: Binary Large Object (528 Bytes)
         IVLength: 24 (0x18)
         IV: Binary Large Object (24 Bytes)
         EncryptedDeviceNonceLength: 24 (0x18)
```

```
            EncryptedDeviceNonce: Binary Large Object (24 Bytes)
```

The client sends a **Register** command with a **SecDeviceAccountRegister** security message and a **SecAccountRegister** account layer message to register an account represented by account URL (grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@), and a device represented by device URL (dpp:///7gws9khpet9z4ezajvnhb5d9fpmcwqrjv3wzez2).

The timestamp (0x479114D1) is a local time on the client's computer. The fingerprint (a9 7a de 47 6e 85 32 3b 78 6f e9 56 b0 f6 2c 88 b5 82 24 80 01) is the relay server's certificate fingerprint as calculated according to section 3.1.1.2. The client encrypted the device key using the server's encryption public key to produce a 384 byte-long encrypted device key (not fully shown in the preceding trace). The signature of the **SecDeviceAccountRegister** message is generated using the message identifier (0x04), account URL, device URL, Fingerprint, and other elements according to section 2.2.12. The client generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: 7D 4E A9 FC 3A 71 15 93 49 15 76 91 84 D4 08 00 23 CF 69 30 97 E5 14 55) to produce the encrypted device nonce (24 bytes: CA 97 B0 E6 DF 91 05 A1 34 9F C9 89 21 42 A7 40 90 DF 29 8A F3 80 A7 42).

In the **SecAccountRegister** account layer message, the client encrypted the account key using the server's encryption public key to produce a 384 byte-long encrypted account key (not fully shown in the preceding trace). The signature for the account layer message is generated using the message identifier (0x04), **account URL**, **device URL**, **ServerCertificateFingerprint**, **Timestamp**, **EncryptedRelayAccountKey**, and **AccountPublicKeysObject** according to section 2.2.14.

The following is an over-the-wire trace of the **Register** command that the client sends to register the client's device and account when using the RSA public key algorithm:

```
    0000         0b 37 0a 02 00 00 00 2e 0a 01 04 04 26 b4      .7..........&´
    000e         f0 49 67 72 6f 6f 76 65 41 63 63 6f 75 6e      ðIgrooveAccoun
    001c         74 3a 2f 2f 6d 38 6b 7a 66 79 73 6d 75 69      t://m8kzfysmui
    002a         71 7a 63 68 33 6a 66 7a 6e 66 65 61 39 68      qzch3jfznfea9h
    0038         65 6d 6b 6e 75 74 6b 72 76 37 6d 76 75 74      emknutkrv7mvut
    0046         61 40 00 14 00 73 c5 db 0d 39 b2 33 d7 14      a@...sÅÛ.9²3×.
    0054         f7 aa d7 28 88 80 b3 5d 43 22 17 80 01 50      ÷ª×(??³]C".?.P
    0062-01db    ... encrypted device key (384 bytes) ...
    01dc         da 24 af bd 45 e5 04 01 04 04 80 01 c2 d7      Ú$¯½Eå....?.Â×
    01ea-0363    ... encrypted account key (384 bytes) ...
    0364         db 32 fb 23 00 01 2b 3b db 36 6f 3d 7d 2b      Û2û#..+;Û6o=}+
    0372-045f    ... signature (256 bytes total) ...
    0460         ec 2d 3f d8 af 2a d5 75 2a 77 34 02 52 53      ì-?Ø¯*Õu*w4.RS
    046e         41 00 52 53 41 00 52 53 41 00 52 53 41 00      A.RSA.RSA.RSA.
    047c         0e 01 00 00 30 82 01 0a 02 82 01 01 00 b0      ....0?...?...°
    048a-0585    ... signature public key ( 270 bytes) ...
    0586         e4 45 da 03 02 03 01 00 01 0e 01 00 00 30      äEÚ..........0
    059e-069d    ... encryption public key (270 bytes) ...
    069e         00 01 01 00 00 30 36 30 31 46 34 33 34 2d      .....0601F434-
    06ac         38 41 32 35 2d 34 41 34 32 2d 38 37 44 43      8A25-4A42-87DC
    06ba         34 42 37 2d 41 41 38 42 2d 34 46 38 46 31      -CB92F3709BD3.
    06c8         01 00 00 00 01 8E 17 03 A5 55 79 25 9B A5      .....?..¥Uy%?¥
    06d6-07c3    ... signature (256 bytes total) ...
    07c4         8c a7 ed 73 9f 9b 93 60 62 34 02 52 53 41      ?§ís???`b4.RSA
    07d2         00 52 53 41 00 52 53 41 00 52 53 41 00 0E      .RSA.RSA.RSA..
    07e0         01 00 00 30 82 01 0A 02 82 01 01 00 B8 35      ...0?...?...¸5
    07ee-08e9    ... signature public key (270 bytes) ...
    08ea         5f 2f 02 03 01 00 01 0e 01 00 00 30 82 01      _/.........0?.
    08f8-0a01    ... encryption public key (270 bytes) ...
    0a02         00 01 18 00 44 99 40 c0 71 24 af 6a 31 4d      ....D?@Àq$¯j1M
    0a10         fb 78 07 8f 58 5a b7 7c cf 0d 84 8b 52 be      ûx.□XZ·|Ï.??R¾
    0a1e         18 00 3f 00 c4 67 ea 9c c9 88 82 d8 b6 77      ..?.Ägê?É??Ø¶w
    0a2c         d5 8e 33 8f 37 49 a2 56 26 cf cf f0            Õ?3□7I¢V&ÏÏð


    - MSGRVSSTP: Register,
- Register: EventID: 0x02
   CommandID: 11 (Register) (0x0B)
   CommandLength: 2615 (0x0A37)
   EventId: 2 (0x00000002)
 - RegistrationToken: SSTPSecDeviceAccountRegister (Length: 2606)
    RegistrationTokenLength: 2606 (0x0A2E)
    MajorVersion: 1 (0x01)
    MinorVersion: 4 (0x04)
    MessageID: 4 (0x04)
  - SSTPSecDeviceAccountRegister:
     Timestamp: 1240511526 (0x49F0B426)
     AccountURL: grooveAccount://m8kzfysmuiqzch3jfznfea9hemknutkrv7mvuta@
     FingerprintLength: 20 (0x0014)
     Fingerprint: Binary Large Object (20 Bytes)
     EncryptedRelayDeviceKeyLength: 384 (0x0180)
     EncryptedRelayDeviceKey: Binary Large Object (384 Bytes)
   - AccountLayerMessage: SSTPSecAccountRegister (Length: 1253)
      AuthenticationTokenLength: 1253 (0x04E5)
      MajorVersion: 1 (0x01)
      MinorVersion: 4 (0x04)
      MessageID: 4 (0x04)
    - SSTPSecAccountRegister:
       EncryptedRelayAccountKeyLength: 384 (0x0180)
       EncryptedRelayAccountKey: Binary Large Object (384 Bytes)
       SignatureLength: 256 (0x0100)
       Signature: Binary Large Object (256 Bytes)
       AccountPublicKeysObjectLength: 564 (0x0234)
     - AccountPublicKeysObject:
        SignatureAlgorithmName: RSA
        EncryptionAlgorithmName: RSA
        SignatureKeyAlgorithmName: RSA
        EncryptionKeyAlgorithmName: RSA
        SignaturePublicKeyLength: 270 (0x0000010E)
        SignaturePublicKey: Binary Large Object (270 Bytes)
        EncryptionPublicKeyLength: 270 (0x0000010E)
        EncryptionPublicKey: Binary Large Object (270 Bytes)
       Reserved1: 1 (0x0001)
       Reserved2: 0 (0x00)
       UserPreAuthToken: 0601F434-8A25-4A42-87DC-CB92F3709BD3
     Reserved1: 1 (0x0001)
     Reserved2: 0 (0x00)
     SignatureLength: 256 (0x0100)
     Signature: Binary Large Object (256 Bytes)
     DevicePublicKeysObjectLength: 564 (0x0234)
   - DevicePublicKeysObject:
      SignatureAlgorithmName: RSA
      EncryptionAlgorithmName: RSA
      SignatureKeyAlgorithmName: RSA
      EncryptionKeyAlgorithmName: RSA
      SignaturePublicKeyLength: 270 (0x0000010E)
      SignaturePublicKey: Binary Large Object (270 Bytes)
      EncryptionPublicKeyLength: 270 (0x0000010E)
      EncryptionPublicKey: Binary Large Object (270 Bytes)
     IVLength: 24 (0x0018)
     IV: Binary Large Object (24 Bytes)
     EncryptedDeviceNonceLength: 24 (0x0018)
     EncryptedDeviceNonce: Binary Large Object (24 Bytes)
```

The client sends a **Register** command with a **SecDeviceAccountRegister** security message and a **SecAccountRegister** account layer message to register an account represented by account URL (grooveAccount://m8kzfysmuiqzch3jfznfea9hemknutkrv7mvuta@), and a device represented by device URL (dpp:///7gws9khpet9z4ezajvnhb5d9fpmcwqrjv3wzez2).

The timestamp (0x49F0B426) is a local time on the client's computer. The fingerprint (73 c5 db 0d 39 b2 33 d7 14 f7 aa d7 28 88 80 b3 5d 43 22 17) is the relay server's certificate fingerprint as calculated according to section 3.1.1.2. The client encrypted the device key using the server's encryption public key to produce a 384 byte-long encrypted device key (not fully shown in the preceding trace). The signature of the **SecDeviceAccountRegister** message is generated using the message identifier (0x04), account URL, device URL, Fingerprint, and other elements according to section 2.2.12. The client generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: 44 99 40 c0 71 24 af 6a 31 4d fb 78 07 8f 58 5a b7 7c cf 0d 84 8b 52 be) to produce the encrypted device nonce (24 bytes: 3f 00 c4 67 ea 9c c9 88 82 d8 b6 77 d5 8e 33 8f 37 49 a2 56 26 cf cf f0).

In the **SecAccountRegister** account layer message, the client encrypted the account key using the server's encryption public key to produce a 384 byte-long encrypted account key (not fully shown in the preceding trace). The signature for the account layer message is generated using the message identifier (0x04), **account URL**, **device URL**, **ServerCertificateFingerprint**, **Timestamp**, and other elements according to section 2.2.14.

4.1.6 RegisterResponse for Device and Account Registration

The following shows an over-the-wire trace of the **RegisterResponse** command that the relay server sends to the client in response to the previous **Register** command:

```
    0000  0c 91 00 0b 00 00 00 88 00 01 03 05 1e 00 01 03  ................
    0010  08 00 d3 14 91 47 14 00 07 67 cc c4 c0 1c ce e1  .....G...g......
    0020  33 c8 1e 67 98 91 dd 45 98 1c 3a cd 00 18 00 b6  3..g...E..:.....
    0030  7c 71 59 ef 63 f8 86 b9 26 80 96 ed 20 eb 10 ba  |qY.c...&... ...
    0040  8c 3a 09 16 a3 0a c4 14 00 95 cc b0 6d c1 32 65  .:..........m.2e
    0050  4d d6 cf fb 00 aa a0 a3 34 d3 c1 02 c2 18 00 50  M.......4......P
    0060  d5 15 c8 1a d9 b1 d8 c3 fe bd 97 9c 30 f5 ea be  ............0...
    0070  33 e1 e9 50 33 7e af 18 00 a5 78 bd 4a 57 e3 6c  3..P3~....x.JW.l
    0080  24 98 14 29 80 3f 53 3e e8 bd 32 02 ef b5 28 cd  $..).?S>..2...(.
    0090  25                                               %

    - SSTP: NOOP, REGISTER_RESPONSE, ATTACH_RESPONSE, OPEN,
      + NOOP: # Of Messages Acked: 0x3
      - REGISTER_RESPONSE: Event ID: 0xb
         CommandID: 12 (REGISTER_RESPONSE) (0xC)
         CommandLength: 145 (0x91)
         EventID: 11 (0xB)
       - RegistrationToken: SecDeviceAccountRegisterResponse (Length: 136)
          RegistrationTokenLength: 136 (0x88)
          MajorVersion: 1 (0x1)
          MinorVersion: 3 (0x3)
          MessageID: 5 (0x5)
        - SecDeviceAccountRegisterResponse:
         - AccountLayerMessage: SecAccountRegisterResponse (Length: 30)
             RegistrationTokenLength: 30 (0x1E)
             MajorVersion: 1 (0x1)
             MinorVersion: 3 (0x3)
             MessageID: 8 (0x8)
          - SecAccountRegisterResponse:
              Reserved: 0 (0x0)
              Timestamp: 1200690387 (0x479114D3)
              HMACLength: 20 (0x14)
              HMAC: Binary Large Object (20 Bytes)
            Reserved: 0 (0x0)
            IVLength: 24 (0x18)
            IV: Binary Large Object (24 Bytes)
            HMACLength: 20 (0x14)
            HMAC: Binary Large Object (20 Bytes)
            DeviceNonceLength: 24 (0x18)
            DeviceNonce: Binary Large Object (24 Bytes)
            EncryptedRelayNonceLength: 24 (0x18)
            EncryptedRelayNonce: Binary Large Object (24 Bytes)
      + ATTACH_RESPONSE: Event ID: 0xb, ResponseID: SSTP_ATTACH_RESPONSE_ID_OK
      + OPEN: EventID 0x80000001, ResourceURL: grooveWanDPP, OpenFlag: 0x0
```

The **RegisterResponse** command contains a **SecDeviceAccountRegisterResponse** security message with a **SecAccountRegisterResponse** account layer message.

In the **SecAccountRegisterResponse** account layer message, the timestamp (0x479114D3) is a local time from the server's computer. The relay server calculated the HMAC (20 bytes: 07 67 CC C4 C0 1C CE E1 33 C8 1E 67 98 91 DD 45 98 1C 3A CD) using the message identifier (0x08), account URL, device URL, Fingerprint and the timestamp according to section 2.2.16.

The relay server generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: B6 7C 71 59 EF 63 F8 86 B9 26 80 96 ED 20 EB 10 BA 8C 3A 09 16 A3 0A C4) to produce the encrypted relay nonce (24 bytes: A5 78 BD 4A 57 E3 6C 24 98 14 29 80 3F 53 3E E8 BD 32 02 EF B5 28 CD 25). The device nonce (50 D5 15 C8 1A D9 B1 D8 C3 FE BD 97 9C 30 F5 EA BE 33 E1 E9 50 33 7E AF) is the decrypted form of the encrypted device nonce received from the **SecDeviceAccountRegister** command**.** The server also generated the HMAC (20 bytes: 95 CC B0 6D C1 32 65 4D D6 CF FB 00 AA A0 A3 34 D3 C1 02 C2) in the **SecDeviceAccountRegisterResponse** message using the message identifier (0x05), account URL, device URL, and Fingerprint according to section 2.2.13.

4.1.7 AttachResponse

The following shows an over-the-wire trace of the **AttachResponse** command that the relay server sends to the client in response to the original **Attach** command:

```
0000  09 71 00 0b 00 00 00 00 67 00 01 03 02 18 00 6c  .q......g......l
0010  95 ac 96 ec ee f8 5d 37 a8 83 97 c8 3e 13 2d 36  ......]7....>.-6
0020  08 05 69 a7 55 00 af 14 00 3c 2a d3 49 4f a4 3d  ..i.U....<*.IO.=
0030  6b 7d f6 e6 83 e8 cd 41 3a f6 1a d7 a8 18 00 bf  k}.....A:.......
0040  0e b1 e0 d2 0b eb ab e0 a5 86 05 c7 5c 4c eb 9a  ............\L..
0050  b9 dd 3d 34 ec 96 f4 18 00 59 55 1b 13 fd 2f 70  ..=4.....YU.../p
0060  f8 4c 0f a5 50 fa b1 3a 17 a6 26 4f 8e 65 1c 51  .L..P..:..&O.e.Q
0070  5f                                               _

- SSTP: NOOP, REGISTER_RESPONSE, ATTACH_RESPONSE, OPEN,
  + NOOP: # Of Messages Acked: 0x3
  + REGISTER_RESPONSE: Event ID: 0xb
    - ATTACH_RESPONSE: Event ID: 0xb, ResponseID: SSTP_ATTACH_RESPONSE_ID_OK
        CommandID: 9 (ATTACH_RESPONSE) (0x9)
        CommandLength: 113 (0x71)
        EventID: 11 (0xB)
        ResponseId: 0 (SSTP_ATTACH_RESPONSE_ID_OK)
      - AuthenticationToken: SecAttachResponse (Length: 103)
          AuthenticationTokenLength: 103 (0x67)
          MajorVersion: 1 (0x1)
          MinorVersion: 3 (0x3)
          MessageID: 2 (0x2)
        - SecAttachResponse:
            IVLength: 24 (0x18)
            IV: Binary Large Object (24 Bytes)
            HMACLength: 20 (0x14)
            HMAC: Binary Large Object (20 Bytes)
            AccountNonceLength: 24 (0x18)
            AccountNonce: Binary Large Object (24 Bytes)
            EncryptedRelayNonceLength: 24 (0x18)
            EncryptedRelayNonce: Binary Large Object (24 Bytes)
  + OPEN: EventID 0x80000001, ResourceURL: grooveWanDPP, OpenFlag: 0x0
```

The relay server sends the **AttachResponse** command after having successfully registered the client's account and device.

The relay server generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: 6C 95 AC 96 EC EE F8 5D 37 A8 83

97 C8 3E 13 2D 36 08 05 69 A7 55 00 AF) to produce the encrypted relay nonce (24 bytes: 59 55 1B 13 FD 2F 70 F8 4C 0F A5 50 FA B1 3A 17 A6 26 4F 8E 65 1C 51 5F). The account nonce (24 bytes: BF 0E B1 E0 D2 0B EB AB E0 A5 86 05 C7 5C 4C EB 9A B9 DD 3D 34 EC 96 F4) is the decrypted form of the encrypted account nonce that the server received from the first **SecAttach** command**.** The server also generated the HMAC (20 bytes: 3C 2A D3 49 4F A4 3D 6B 7D F6 E6 83 E8 CD 41 3A F6 1A D7 A8) using the message identifier (0x02), account URL, relay URL, device URL, and the relay EventID according to section 2.2.7.

4.1.8 AttachAuthenticate

The following shows an over-the-wire trace of the **AttachAuthenticate** command that the client sends to complete the account authentication:

```
0000  0a 40 00 0b 00 00 00 37 00 01 04 03 18 00 dc 6c  .@.....7.......l
0010  b9 c6 9a c9 14 7f 9d 81 8e 2a 84 79 17 d3 33 21  .........*.y..3!
0020  03 2d 2e 1e 70 bb 18 00 15 a7 55 66 11 03 20 89  .-..p.....Uf.. .
0030  75 a8 7a d5 5f 2a be 3a 21 11 f7 06 a2 02 d1 9d  u.z._*.:!.......

- SSTP: ATTACH_AUTHENTICATE,
  - ATTACH_AUTHENTICATE: Event ID: 0xb
     CommandID: 10 (ATTACH_AUTHENTICATE) (0xA)
     CommandLength: 64 (0x40)
     EventID: 11 (0xB)
   - AuthenticationToken: SecAttachAuthenticate (Length: 55)
      AuthenticationTokenLength: 55 (0x37)
      MajorVersion: 1 (0x1)
      MinorVersion: 4 (0x4)
      MessageID: 3 (0x3)
    - SecAttachAuthenticate:
       RelayAccountNonceLength: 24 (0x18)
       RelayAccountNonce: Binary Large Object (24 Bytes)
       RelayDeviceNonceLength: 24 (0x18)
       RelayDeviceNonce: Binary Large Object (24 Bytes)
```

The client includes two relay nonces in the **SecAttachAuthenticate** message. The relay device nonce (24 bytes: 15 A7 55 66 11 03 20 89 75 A8 7A D5 5F 2A BE 3A 21 11 F7 06 A2 02 D1 9D) is the decrypted form of the encrypted relay nonce (24 bytes: A5 78 BD 4A 57 E3 6C 24 98 14 29 80 3F 53 3E E8 BD 32 02 EF B5 28 CD 25) that the server sent in the **SecDeviceAccountRegisterResponse** message, and the relay account nonce (24 bytes: DC 6C B9 C6 9A C9 14 7F 9D 81 8E 2A 84 79 17 D3 33 21 03 2D 2E 1E 70 BB) is the decrypted form of the encrypted relay nonce (24 bytes: 59 55 1B 13 FD 2F 70 F8 4C 0F A5 50 FA B1 3A 17 A6 26 4F 8E 65 1C 51 5F) that the server sent in the **SecAttachResponse** message.

4.1.9 Close

The following shows an over-the-wire trace of the **Close** command that the relay server sends to the client to close the **Attach** session:

```
0000  11 08 00 0b 00 00 00 00                          ........

- SSTP: REGISTER_RESPONSE, MESSAGE, DATA, CLOSE, END_MESSAGE,
  + REGISTER_RESPONSE: Event ID: 0xc
  + MESSAGE: Session ID: 0x80000001, MessageCount: 0, Message Flag: 0x0
  + DATA: Session ID: 0x80000001, Data Size: 85
  - CLOSE: Session ID: 0xb, Close Reason: SSTP_CLOSE_REASON_ID_NO_REASON
     CommandID: 17 (CLOSE) (0x11)
     CommandLength: 8 (0x8)
     SessionID: 11 (0xB)
     ReasonID: 0 (0x0)
  + END_MESSAGE: Session ID: 0x80000001
```

The SessionId (0x0B) in the **Close** command matches the EventId in the **Attach**, **Register**, **RegisterResponse**, **AttachResponse** and **AttachAuthenticate** commands.

4.1.10 Register for Identity Registration

The following shows an over-the-wire trace of the **Register** command that the client sends to register identities:

```
0000  0b e8 00 0c 00 00 00 f4 00 01 04 06 d3 14 91 47  ...............G
0010  67 72 6f 6f 76 65 41 63 63 6f 75 6e 74 3a 2f 2f  grooveAccount://
0020  6e 67 6d 6a 77 62 61 7a 6d 39 78 69 7a 34 65 74  ngmjwbazm9xiz4et
0030  73 36 35 72 72 34 63 39 6b 62 78 6b 78 70 68 64  s65rr4c9kbxkxphd
0040  77 36 67 70 6b 36 73 40 00 14 00 b4 12 cd e2 0b  w6gpk6s@........
0050  bf 4d 88 d9 4c 5b 3f 4d 80 6b 53 fb 2d ab 88 00  .M..L[?M.kS.-...
0060  68 00 02 00 67 72 6f 6f 76 65 49 64 65 6e 74 69  h...grooveIdenti
0070  74 79 3a 2f 2f 74 76 78 61 62 36 68 76 38 67 36  ty://tvxab6hv8g6
0080  6b 6e 68 6d 77 33 76 78 61 75 6a 73 70 77 73 68  knhmw3vxaujspwsh
0090  63 34 67 39 68 40 00 67 72 6f 6f 76 65 49 64 65  c4g9h@.grooveIde
00a0  6e 74 69 74 79 3a 2f 2f 64 70 75 64 36 35 63 79  ntity://dpud65cy
00b0  76 35 68 35 7a 34 73 32 6e 62 76 66 6e 6e 61 32  v5h5z4s2nbvfnna2
00c0  36 77 79 32 7a 79 33 35 40 00 67 72 6f 6f 76 65  6wy2zy35@.groove
00d0  44 4e 53 3a 2f 2f 72 65 6c 61 79 2e 63 6f 6e 74  DNS://relay.cont
00e0  6f 73 6f 2e 63 6f 6d 00                          oso.com.

- SSTP: REGISTER, OPEN_RESPONSE,
  - REGISTER: Event ID: 0xc
     CommandID: 11 (REGISTER) (0xB)
     CommandLength: 232 (0xE8)
     EventID: 12 (0xC)
   - RegistrationToken: SecIdentityRegister (Length: 244)
      RegistrationTokenLength: 244 (0xF4)
      MajorVersion: 1 (0x1)
      MinorVersion: 4 (0x4)
      MessageID: 6 (0x6)
    - SecIdentityRegister:
       Timestamp: 1200690387 (0x479114D3)
       AccountURL: grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@
       HMACLength: 20 (0x14)
       HMAC: Binary Large Object (20 Bytes)
       IdentityRegistrationFlags: 0 (0x0)
       IdentityListsLength: 104 (0x68)
     - IdentityLists: IDs Added=2, IDs Removed=0
        IdentitiesToAddCount: 2 (0x2)
        IdentitiesToRemoveCount: 0 (0x0)
        IdentityURL: grooveIdentity://tvxab6hv8g6knhmw3vxaujspwshc4g9h@
        IdentityURL: grooveIdentity://dpud65cyv5h5z4s2nbvfnna26wy2zy35@
       RelayURL: grooveDNS://relay.contoso.com
  + OPEN_RESPONSE: Session ID: 0x80000001, Response: SSTP_OPEN_RESPONSE_ID_OK
```

The client sends the **Register** command to register its identities after having authenticated its device and account with the relay server.

The timestamp (0x479114D3) is a local time on the client's computer. In this command, the client wants to registers two identities: grooveIdentity://tvxab6hv8g6knhmw3vxaujspwshc4g9h@ and grooveIdentity://dpud65cyv5h5z4s2nbvfnna26wy2zy35@ for the account: grooveAccount://ngmjwbazm9xiz4ets65rr4c9kbxkxphdw6gpk6s@. The client calculates the HMAC (20 bytes: B4 12 CD E2 0B BF 4D 88 D9 4C 5B 3F 4D 80 6B 53 FB 2D AB 88) using the message identifier (0x06), account URL, relay URL, device URL and the timestamp according to section 2.2.17.

4.1.11 RegisterResponse for Identity Registration

The following shows an over-the-wire trace of the **RegisterResponse** command that the relay sends to the client in response to the **Register** command for identity registration:

```
0000  0c 09 00 0c 00 00 00 00 00                      .........

- SSTP: REGISTER_RESPONSE, MESSAGE, DATA, CLOSE, END_MESSAGE, ,
  - REGISTER_RESPONSE: Event ID: 0xc
         CommandID: 12 (REGISTER_RESPONSE) (0xC)
         CommandLength: 9 (0x9)
         EventID: 12 (0xC)
       - RegistrationToken: (Length: 0)
                 NULL_RegistrationToken: 0 (0x0)
  + MESSAGE: Session ID: 0x80000001, MessageCount: 0, Message Flag: 0x0
  + DATA: Session ID: 0x80000001, Data Size: 85
  + CLOSE: Session ID: 0xb, Close Reason: SSTP_CLOSE_REASON_ID_NO_REASON
  + END_MESSAGE: Session ID: 0x80000001
```

The event ID (0x0C) in the **RegisterResponse** command matches that from the previous **Register** command. This command contains no authentication token.

4.2 Registration and Authentication for an Account on a New Device

In this scenario, a client has already successfully registered its account with a relay server, but now is connecting to the relay server from a new device that has not registered with the relay server. The following sequence diagram shows what SSTP commands and security messages are exchanged between the client and the relay server.

The following diagram shows a sequence of SSTP commands and security messages that the client and the server exchange to complete the registration and authentication for this scenario. This message sequence is very similar to the previous one for the new account registration. Most SSTP commands and security messages exchanged between the client and the server are the same as in the previous scenario. What is different here is that the relay server responds with the **SecAttachResponseNewDeviceRegistrationNeeded** security message instead of the **SecAttachResponseAccountRegistrationNeeded** message, after having received the initial **Attach** command from the client. When registering an account on a new device, the client sends a **Register** command embedding a **SecDeviceAccountRegister** message with the **SecAccountOnNewDevice** account layer message.

After the account from the new device is successfully authenticated with the server, the client again exchanges **Register**/**RegisterResponse** with the server to register identities associated with the account.

Because most SSTP commands and security messages exchanged in this scenario are the same as in the previous scenario, only the new commands and messages are annotated here.

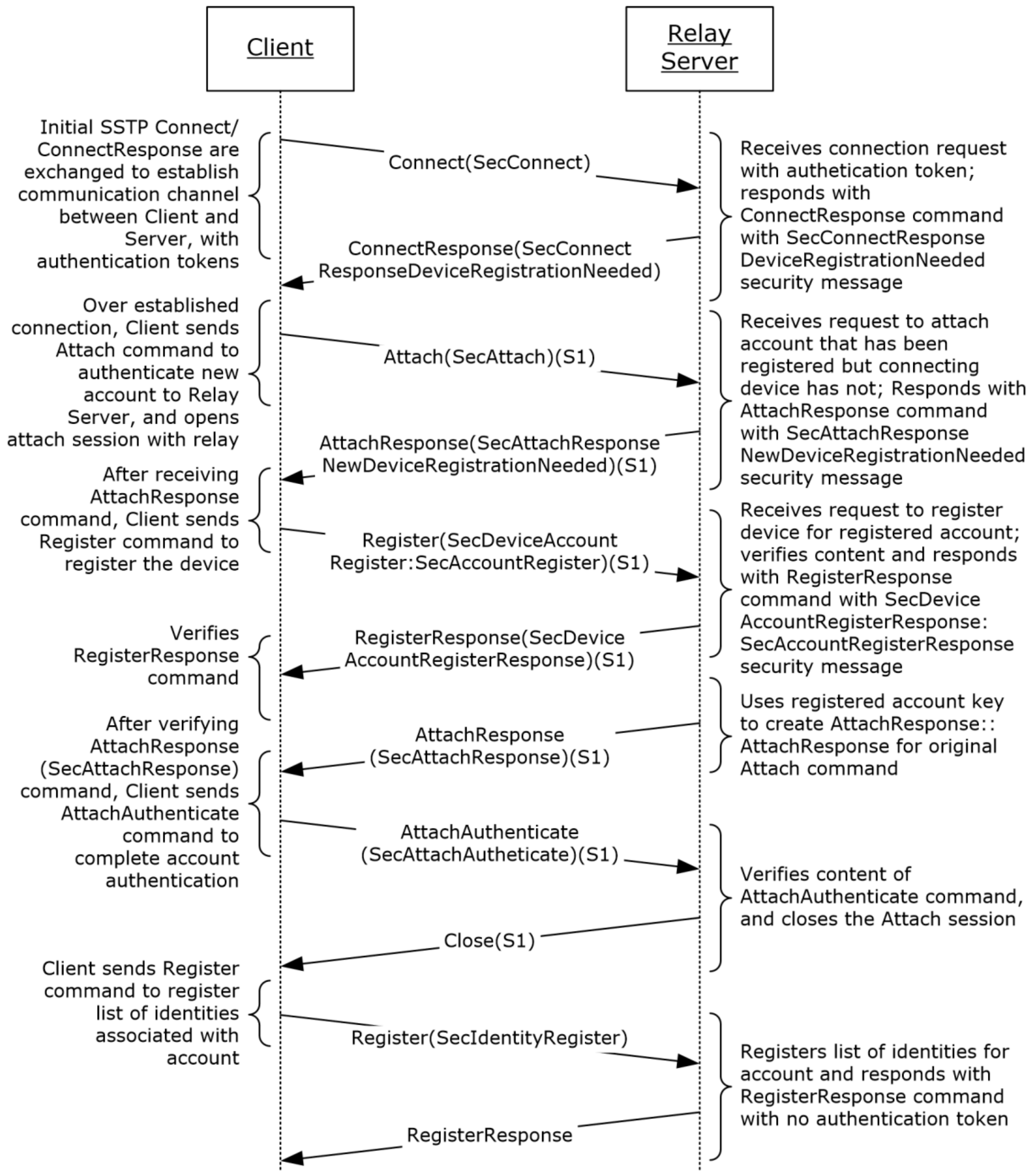
Client
Relay Server
Initial SSTP Connect/ ConnectResponse are exchanged to establish communication channel between Client and Server, with authentication tokens
Connect(SecConnect)
Receives connection request with authetication token; responds with ConnectResponse command with SecConnectResponse DeviceRegistrationNeeded security message
ConnectResponse(SecConnect ResponseDeviceRegistrationNeeded)
Over established connection, Client sends Attach command to authenticate new account to Relay Server, and opens attach session with relay
Attach(SecAttach)(S1)
Receives request to attach account that has been registered but connecting device has not; Responds with AttachResponse command with SecAttachResponse NewDeviceRegistrationNeeded security message
AttachResponse(SecAttachResponse NewDeviceRegistrationNeeded)(S1)
After receiving AttachResponse command, Client sends Register command to register the device
Register(SecDeviceAccount Register:SecAccountRegister)(S1)
Receives request to register device for registered account; verifies content and responds with RegisterResponse command with SecDevice AccountRegisterResponse: SecAccountRegisterResponse security message
Verifies RegisterResponse command
RegisterResponse(SecDevice AccountRegisterResponse)(S1)
Uses registered account key to create AttachResponse:: AttachResponse for original Attach command
After verifying AttachResponse (SecAttachResponse) command, Client sends AttachAuthenticate command to complete account authentication
AttachResponse (SecAttachResponse)(S1)
AttachAuthenticate (SecAttachAutheticate)(S1)
Verifies content of AttachAuthenticate command, and closes the Attach session
Close(S1)
Client sends Register command to register list of identities associated with account
Register(SecIdentityRegister)
Registers list of identities for account and responds with RegisterResponse command with no authentication token
RegisterResponse

Figure 8: Message exchange sequence for an account on a new device

4.2.1 AttachResponseNewDeviceRegistrationNeeded

When the relay server receives an account attach request from the client, it searches its metadata for registration information about the account and the connecting device. If the server finds out that the account has been registered, but the device has not, it sends an **AttachResponse** command with the **SecAttachResponseNewDeviceRegistrationNeeded** message to the client.

The following shows an over-the-wire trace of the **AttachResponse** command with the **SecAttachResponseNewDeviceRegistrationNeeded** message:

```
0000  09 0d 00 07 00 00 00 03 03 00 01 03 0b           .............

- SSTP: ATTACH_RESPONSE, OPEN_RESPONSE,
  - ATTACH_RESPONSE: Event ID: 0x7, ResponseID: SSTP_ATTACH_RESPONSE_IDAWAITING_REGISTER
     CommandID: 9 (ATTACH_RESPONSE) (0x9)
     CommandLength: 13 (0xD)
     EventID: 7 (0x7)
     ResponseId: 3 (SSTP_ATTACH_RESPONSE_IDAWAITING_REGISTER)
   - AuthenticationToken: SecAttachResponseNewDeviceRegistrationNeeded (Length: 3)
      AuthenticationTokenLength: 3 (0x3)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 11 (0xB)
      SecAttachResponseNewDeviceRegistrationNeeded:
  + OPEN_RESPONSE: Session ID: 0x4, Response: SSTP_OPEN_RESPONSE_ID_OK
```

The **ResponseId** field in the **AttachResponse** command is **AwaitingRegister**, indicating that the relay server is waiting for the client to register its account.

4.2.2 Register for Account on New Device

When the client sees the **SecAttachResponseNewDeviceRegistrationNeeded** message, it creates a **Register** command with the **SecDeviceAccountRegister** message to register with the server the account on the new device. The device layer information in the security message is very much similar to that in the 1st **Register** command annotated in the previous section. What is different is that the account layer message that contains the **SecAccountOnNewDevice** message. An example **SecAccountOnNewDevice** message follows:

```
0000  19 00 01 04 05 14 00 75 fd 1a 0a 48 6c 02 5d 6b  .......u...Hl.]k
0010  f5 05 a3 ea c0 0e 52 6e 7d 62 ca                 ......Rn}b.

- SecAccountOnNewDevice:
   HMACLength: 20 (0x14)
   HMAC: Binary Large Object (20 Bytes)
```

The client generates the HMAC using the message identifier, account URL, device URL, Fingerprint and the timestamp according to section 2.2.15.

4.2.3 RegisterResponse

The **RegisterResponse** command that the server sends back to the client in response to the **Register** command for an account on a new device is exactly the same as the first **RegisterResponse** command annotated in the previous section. The server generates a **SecDeviceAccountRegisterResponse** message that contains a **SecAccountRegisterResponse** message, and then sends the **SecDeviceAccountRegisterResponse** as an authentication token in a SSTP **RegisterResponse** command**.**

4.3 Authentications for a Reconnecting Client

The following diagram shows a sequence of SSTP commands and security messages that the client and the server exchange to complete the authentications for this scenario.

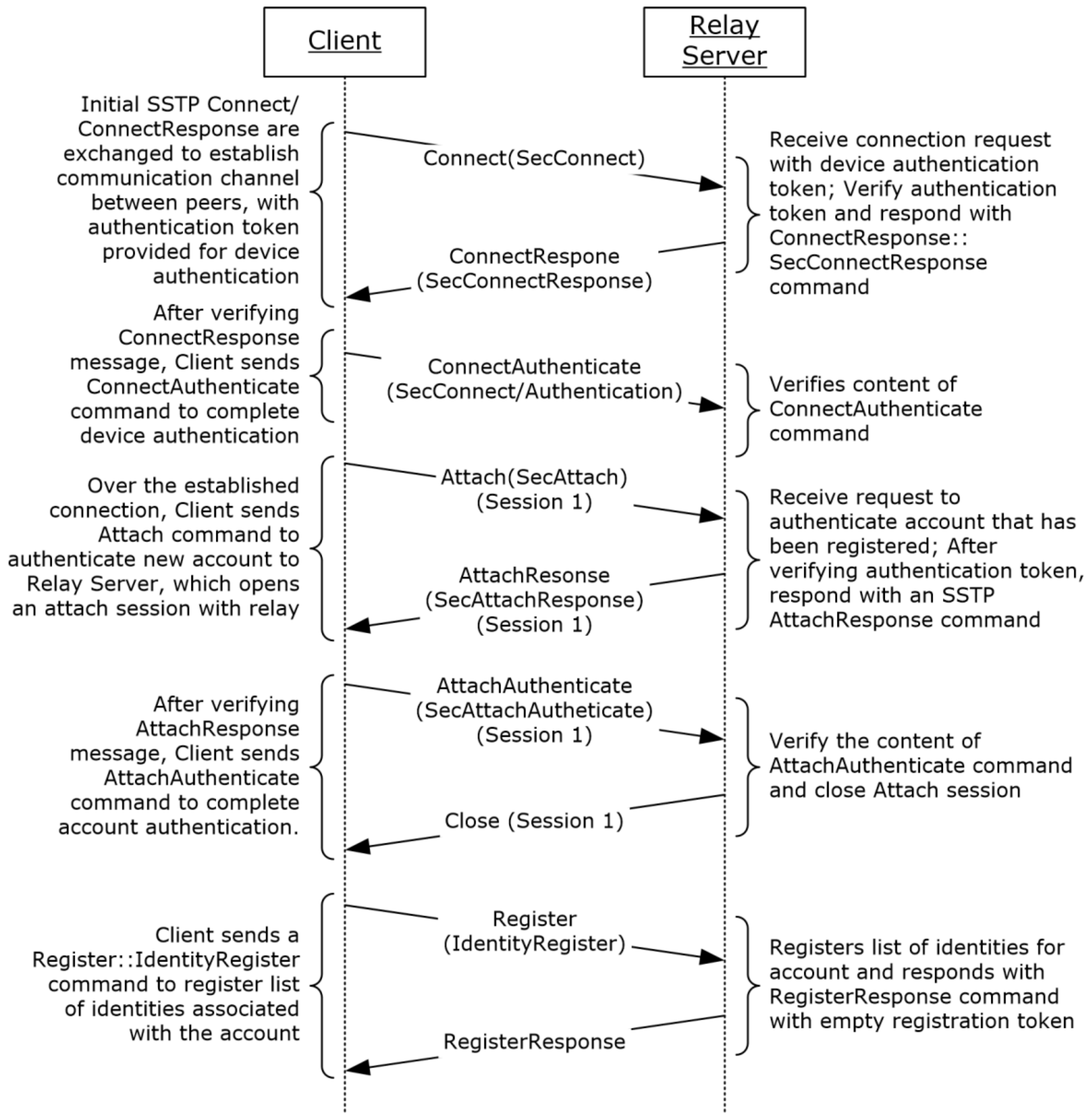
Client
Relay Server
Initial SSTP Connect/ ConnectResponse are exchanged to establish communication channel between peers, with authentication token provided for device authentication
Connect(SecConnect)
Receive connection request with device authentication token; Verify authentication token and respond with ConnectResponse:: SecConnectResponse command
ConnectRespone (SecConnectResponse)
After verifying ConnectResponse message, Client sends ConnectAuthenticate command to complete device authentication
ConnectAuthenticate (SecConnect/Authentication)
Verifies content of ConnectAuthenticate command
Over the established connection, Client sends Attach command to authenticate new account to Relay Server, which opens an attach session with relay
Attach(SecAttach) (Session 1)
Receive request to authenticate account that has been registered; After verifying authentication token, respond with an SSTP AttachResponse command
AttachResonse (SecAttachResponse) (Session 1)
After verifying AttachResponse message, Client sends AttachAuthenticate command to complete account authentication.
AttachAuthenticate (SecAttachAutheticate) (Session 1)
Verify the content of AttachAuthenticate command and close Attach session
Close (Session 1)
Client sends a Register::IdentityRegister command to register list of identities associated with the account
Register (IdentityRegister)
Registers list of identities for account and responds with RegisterResponse command with empty registration token
RegisterResponse

Figure 9: Message exchange sequence for a reconnecting client

In this common scenario, a client is re-connecting to a server after having registered its account and device with the server. The client exchanges **Connect**/**ConnectResponse**/**ConnectAuthenticate** commands to authenticate its device, and then exchanges **Attach**/**AttachResponse**/**AttachAuthenticate** commands to authenticate its account. This scenario skips the exchange of **Register**/**RegisterResponse** commands for key registrations as the server has saved the keys from a previous registration sequence from the client.

After the account is successfully authenticated with the server, the client again exchanges **Register**/**RegisterResponse** with the server to register identities associated with the account.

Most SSTP commands and security messages in this scenario have been annotated in the previous sections. So only the following two commands: **ConnectResponse** and **ConnectAuthenticate** are annotated here.

4.3.1 ConnectResponse

The following shows an over-the-wire trace of the **ConnectResponse** command that the server sends to the client in response to the **Connect** command (for SSTP fields, see [MS-GRVSSTP]):

```
0000  02 a8 00 01 05 00 67 00 01 03 02 18 00 0c 82 7b  ......g........{
0010  10 aa f3 3c 92 b2 df f7 c6 10 8a 89 8e a7 d6 c9  ...<............
0020  2b f7 bd c2 5d 14 00 ce ff 54 50 5c 96 ee cf 79  +...]....TP\...y
0030  91 4d fa 6d 62 32 3f d5 83 8a 4b 18 00 5b 71 5b  .M.mb2?...K..[q[
0040  38 69 dd e2 bb 8e 61 2c 94 cd b0 a3 bf b6 db 5b  8i....a,.......[
0050  e0 df 92 3f 04 18 00 8e 96 dd 74 c4 5b 11 70 db  ...?......t.[.p.
0060  b6 a4 53 3b ce 58 00 06 b5 df a5 d1 a7 2b 70 03  ..S;.X.......+p.
0070  47 72 6f 6f 76 65 20 52 65 6c 61 79 20 31 32 2e  Groove Relay 12.
0080  30 20 31 35 30 31 00 00 01 67 72 6f 6f 76 65 44  0 1501...grooveD
0090  4e 53 3a 2f 2f 72 65 6c 61 79 2e 63 6f 6e 74 6f  NS://relay.conto
00a0  73 6f 2e 63 6f 6d 00 00                          so.com..

- SSTP: CONNECT_RESPONSE,
  - CONNECT_RESPONSE: SSTP Version: 1.5, Connect Response: SSTP_CONNECT_RESPONSE_ID_OK,
Flags: 0x3
     CommandID: 2 (CONNECT_RESPONSE) (0x2)
     CommandLength: 167 (0xA8)
     MajorVersion: 1 (0x1)
     MinorVersion: 5 (0x5)
     ResponseID: 0 (SSTP_CONNECT_RESPONSE_ID_OK) (0x0)
   - AuthenticationToken: SecConnectResponse (Length: 103)
      AuthenticationTokenLength: 103 (0x67)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 2 (0x2)
    - SecConnectResponse:
       IVLength: 24 (0x18)
       IV: Binary Large Object (24 Bytes)
       HMACLength: 20 (0x14)
       HMAC: Binary Large Object (20 Bytes)
       DeviceNonceLength: 24 (0x18)
       DeviceNonce: Binary Large Object (24 Bytes)
       EncryptedRelayNonceLength: 24 (0x18)
       EncryptedRelayNonce: Binary Large Object (24 Bytes)
   - ConnectResponseFlags: MS.....
      r1:             (0.......) Not Used
      r2:             (.0......) Not Used
      r3:             (..0.....) Not Used
      r4:             (...0....) Not Used
      r5:             (....0...) Not Used
      C:             (.....0..) Not Used
      S:      (......1.) SingleHop Fanout Supported
      M:      (.......1) MultiDrop Fanout Supported
     PeerProductVersion: Groove Relay 12.0 1501
     PeerProductCapabilities:
     NumberOfTargetDeviceURLs: 1 (0x1)
     TargetDeviceURL: grooveDNS://relay.contoso.com
   - RedirectMapEntries:
      RedirectMapEntries: 0 (0x0)
```

The relay server generated a random 24 byte-long number and encrypted it using the MARC4 stream cipher with a secret device key (24 bytes) and the IV (24 bytes: 0C 82 7B 10 AA F3 3C 92 B2 DF F7 C6 10 8A 89 8E A7 D6 C9 2B F7 BD C2 5D) to produce the encrypted relay nonce (24 bytes: 8E 96 DD 74 C4 5B 11 70 DB B6 A4 53 3B CE 58 00 06 B5 DF A5 D1 A7 2B 70). The device nonce (24 bytes: 5B 71 5B 38 69 DD E2 BB 8E 61 2C 94 CD B0 A3 BF B6 DB 5B E0 DF 92 3F 04) is the decrypted form of the encrypted device nonce that the server received from the **SecConnect** command**.** The server also generated the HMAC (20 bytes: CE FF 54 50 5C 96 EE CF 79 91 4D FA 6D 62 32 3F D5 83 8A 4B) using the message identifier (0x02), device URL, Fingerprint, and the relay nonce according to section 2.2.2.

4.3.2 ConnectAuthenticate

The following shows an over-the-wire trace of the **ConnectAuthenticate** command that the client sends to the server in response to the **ConnectResponse** command (for SSTP fields, see [MS-GRVSSTP]):

```
0000  03 22 00 1d 00 01 03 03 18 00 bb d7 6b 00 c9 74  .".........k..t
0010  b0 28 41 c0 00 9d 9b 31 e0 f3 c5 d1 f8 0e 40 dd  .(A....1......@.
0020  b3 fd                                            ..

- SSTP: CONNECT_AUTHENTICATE,
  - CONNECT_AUTHENTICATE:
     CommandID: 3 (CONNECT_AUTHENTICATE) (0x3)
     CommandLength: 34 (0x22)
   - DeviceRegistrationToken: SecConnectAuthenticate (Length: 29)
      AuthenticationTokenLength: 29 (0x1D)
      MajorVersion: 1 (0x1)
      MinorVersion: 3 (0x3)
      MessageID: 3 (0x3)
    - SecConnectAuthenticate:
       RelayNonceLength: 24 (0x18)
       RelayNonce: Binary Large Object (24 Bytes)
```

The client sends the SecConnectAuthenticate message with the relay nonce (24 bytes: BB D7 6B 00 C9 74 B0 28 41 C0 00 9D 9B 31 E0 F3 C5 D1 F8 0E 40 DD B3 FD) to the relay server. The relay nonce is the decrypted form of the encrypted relay nonce (24 bytes: 8E 96 DD 74 C4 5B 11 70 DB B6 A4 53 3B CE 58 00 06 B5 DF A5 D1 A7 2B 70) from the previous ***SecConnectResponse*** message.

5 Security

5.1 Security Considerations for Implementers

5.1.1 Use of semi-weak algorithms

The current protocol uses SHA1 and HMAC-SHA1 for message digest and keyed message digest. While there are no practical attacks against SHA1 at this point, it is showing signs of weakness.

5.1.2 Use of weak algorithms

The current protocol uses MARC4 (RC4-drop(256)) for **symmetric key** encryption. It's considered somewhat weak at this point, because of problems with its key scheduling algorithm.

5.1.3 Use of non-standard/suspect algorithms

The current protocol uses ELGAMAL for public key encryption. ELGAMAL is known to be insecure in the face of certain cipher text attacks. In addition, ELGAMAL hasn't been scrutinized as heavily as some other encryption algorithms.

5.1.4 Insufficient integrity protection of SSTP headers

The current protocol only integrity-protects parts of the message header when the entire SSTP message header could be integrity-protected. This would make it possible for an active attacker to tamper with SSTP message header in transit. This tampering can't be detected by the message recipient.

5.1.5 Insufficient encryption of SSTP headers

The current protocol only encrypts parts of the message header when most of SSTP message header could be encrypted. This would make it possible for a passive attacker to read most data in all SSTP message headers.

5.1.6 Use of the same key for encryption and HMAC

The current protocol uses the same secret key for both encryption and integrity protection. This opens the door for related key attacks.

5.1.7 Version number is not included in the HMAC

The current protocol does not include the version number in the HMAC. This makes the protocol susceptible to downgrade attacks.

5.1.8 Susceptibility to TCP sessions hijacking

Once the device and account authentications succeed, the SSTP protocol does not secure any subsequent messages. This makes it possible for an active attacker to hijack the TCP session, and to cause denial of service for the originally connected client. In some scenarios, this also makes it possible for an active attacker to receive messages targeted for the originally authenticated client (such as when both attacker and victim are on the same LAN).

5.2 Index of Security Parameters

Security Parameter	Section
Client device encryption public key	1.3.1.1, 1.3.1.2, 1.5, 2.2.12, 3.1.1.3, 3.2.1, 3.2.3, 3.2.4.3.1, 3.2.4.3.2, 3.3.1, 3.3.5.5.1, 3.3.5.5.2, 4.1.5
Client device encryption private key	1.3.1.1, 1.5, 3.1.1.3, 3.2.1, 3.2.3
Client device signature public key	1.3.1.1, 1.3.1.2, 1.5, 2.2.12, 3.1.1.3, 3.2.1, 3.2.3, 3.2.4.3.1, 3.2.4.3.2, 3.3.1, 3.3.5.5.1, 3.3.5.5.2, 4.1.5
Client device signature private key	1.3.1.1, 1.5, 2.2.12, 3.1.1.3, 3.2.1, 3.2.3, 3.3.5.5.1, 3.3.5.5.2
Account encryption public key	1.3.1.1, 1.3.1.2, 1.5, 2.2.14, 3.1.1.3, 3.2.1, 3.2.3, 3.2.4.3.1, 3.3.1, 3.3.5.5.1, 4.1.5
Account encryption private key	1.3.1.1, 1.5, 3.1.1.3, 3.2.1, 3.2.3
Account signature public key	1.3.1.1, 1.3.1.2, 1.5, 2.2.14, 3.1.1.3, 3.2.1, 3.2.3, 3.2.4.3.1, 3.3.1, 3.3.5.5.1, 4.1.5
Account signature private key	1.3.1.1, 1.5, 2.2.14, 3.1.1.3, 3.2.1, 3.2.3, 3.3.5.5.1
Relay server encryption public key	1.3.3.1, 1.5, 2.2.12, 2.2.14, 3.1.1.1, 3.1.1.2, 3.3.3, 3.3.5.5.1, 3.3.5.5.2, 4.1.5
Relay server encryption private key	1.3.3.1, 1.5, 3.1.1.1, 3.3.3, 3.3.5.5.1, 3.3.5.5.2
Relay server signature public key	1.5, 3.1.1.1, 3.3.3
server signature Relay private key	1.5, 3.1.1.1, 3.3.3
Secret key shared between the client device and the relay server	1, 1.3, 1.3.1, 1.3.1.2, 1.3.3.1, 1.3.3.2, 2.2.1, 2.2.2, 2.2.3, 2.2.7, 2.2.8, 2.2.9, 2.2.12, 2.2.13, 3.1.1.4, 3.1.3.3, 3.2.1, 3.2.4.1, 3.2.4.3.1, 3.2.4.3.2, 3.2.5.1, 3.2.5.8, 3.3.1, 3.3.5.1, 3.3.5.5.1, 4.1, 4.1.1, 4.1.5, 4.1.6, 4.1.7, 4.3.1
Secret key shared between the account and the relay server	1, 1.3, 1.3.1, 1.3.1.2, 1.3.3.1, 1.3.3.2, 2.2.6, 2.2.7, 2.2.8, 2.2.12, 2.2.14, 2.2.15, 2.2.16, 2.2.17, 3.1.1.4, 3.1.3.3, 3.2.1, 3.2.4.2, 3.2.4.3.1, 3.2.5.5, 3.3.1, 3.3.5.3, 3.3.5.5.1, 3.3.5.5.2, 3.3.5.6, 4.1, 4.1.3, 4.1.5
Secret key encryption algorithm	2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.11, 2.2.12, 2.2.13, 3.1.1.4, 3.2.4.1, 3.2.4.2, 3.2.5.1, 3.2.5.5, 3.2.5.8, 3.3.5.1, 3.3.5.3, 4.1.1, 4.1.3, 4.1.5, 4.1.6, 4.1.7, 4.3.1, 5.1.2
Public key encryption algorithm	3.1.1.1, 3.1.1.2, 3.1.1.3, 3.3.5.5.1, 3.3.5.5.2, 4.1.5, 5.1.3
Signature algorithm	2.2.12, 2.2.14, 3.1.1.1, 3.1.1.3, 3.3.5.5.1, 3.3.5.5.2, 4.1.5
Hash algorithm	2.2, 2.2.1, 2.2.2, 2.2.6, 2.2.7, 2.2.12, 2.2.13, 2.2.14, 2.2.15, 2.2.16, 2.2.17, 3.1.1.1, 3.1.1.2, 3.3.5.5.1, 3.3.5.5.2, 5.1.1
HMAC algorithm	2.2.1, 2.2.2, 2.2.6, 2.2.7, 2.2.13, 2.2.15, 2.2.16, 2.2.17, 5.1.1

Security Parameter	Section
Initialization vector	2.2.1, 2.2.1, 2.2.2, 2.2.6, 2.2.7, 2.2.12, 2.2.13, 3.1.1.4, 3.2.4.1, 3.2.4.2, 3.2.5.1, 3.2.5.5, 3.2.5.8, 3.3.5.1, 3.3.5.3, 4.1.1, 4.1.3, 4.1.6, 4.1.7, 4.3.1
Message signature	2.2.12, 2.2.14, 3.3.5.5.1, 3.3.5.5.2, 4.1.5
Message HMAC	1.3.3.1, 2.2.1, 2.2.2, 2.2.6, 2.2.7, 2.2.13, 2.2.15, 2.2.16, 2.2.17, 3.3.5.5.1, 5.1.6
Nonce	2.2, 2.2.1, 2.2.2, 2.2.2, 2.2.4, 2.2.5, 2.2.6, 2.2.7, 2.2.11, 2.2.12, 2.2.13, 2.2.14, 3.1.1.4, 3.2.4.1, 3.2.4.2, 3.2.5.1, 3.2.5.5.5, 3.2.5.8.8, 3.3.5.1, 3.3.5.2, 3.3.5.3, 3.3.5.4, 3.3.5.5.1, 4.1.1, 4.1.3, 4.1.5, 4.1.6, 4.1.7, 4.1.8, 4.3.1, 4.3.2, 6

6 Appendix A: 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: The Office Groove 2007 and Microsoft SharePoint Workspace 2010 clients set the minor version number to 4 for account layer messages, and set the minor version number to 3 for device layer messages, while the Office Groove Server 2007 and Microsoft Groove Server 2010 relay servers always set the minor version number to 4.

<2> Section 3.1.1.1: The encryption key in Office Groove 2007 and Office Groove Server 2007 is the **Distinguished Encoding Rules (DER)** encoded using the **ASN.1** syntax [RFC3641]:

```
DHPublicKeyForElgamalEncryption ::= SEQUENCE {
size INTEGER, -- canned prime size, is  1536
 y     INTEGER, -- public key (g^x mod p,
                 where x is the 1536-bit Diffie-Hellman private key)
}
```

Diffie-Hellman group parameters used in this case are as follows:

Prime p is 2^1536 minus 0x16F055.

Generator g is 3.

The encryption key in SharePoint Workspace 2010 and Groove Server 2010 is the Distinguished Encoding Rules (DER) encoded using the ASN.1 syntax:

```
DHPublicKeyForElgamalEncryption ::= SEQUENCE {
      p    INTEGER, -- prime, p
      q    INTEGER OPTIONAL, -- factor of p-1, only present when p = j*q+1,
                             -- where j is not 2
      g    INTEGER, -- generator, g
      y    INTEGER -- public key (g^x mod p, where x is the private key)


}
```

<3> Section 3.1.1.3: Office Groove 2007 uses ElGamal encryption to generate the encryption key pair and SharePoint Workspace 2010 uses RSA encryption to generate the encryption key pair.

<4> Section 3.2.1: Identity URLs start with the "grooveIdentity://" prefix, and the length of the URL after the prefix does not exceed 80 characters.

<5> Section 3.2.4.3.1: The Office Groove 2007 and SharePoint Workspace 2010 clients use the user pre-auth token from the first identity in an account when constructing the **SecAccountRegister** message.

<6> Section 3.2.5.3: The Office Groove 2007 and SharePoint Workspace 2010 clients make the best effort to send a **ConnectClose**, but the command can be sent from different sources, and the TCP connection could have been terminated before the command is sent.

<7> Section 3.2.5.4: The Office Groove 2007 and SharePoint Workspace 2010 clients make the best effort to send a **ConnectClose**, but the command can be sent from different sources, and the TCP connection could have been terminated before the command is sent.

<8> Section 3.2.5.5: The Office Groove 2007 and SharePoint Workspace 2010 clients send no Close command when the **SecAttachResponse** message verification fails.

<9> Section 3.2.5.8: The Office Groove 2007 and SharePoint Workspace 2010 clients send no Close command when the **SecDeviceAccountResponse** message verification fails.

<10> Section 3.3.5: The Office Groove Server 2007 and Groove Server 2010 relay servers allow a valid **SecConnectAuthenticate** message to be received at any time following receipt of the initial **SecConnect**, provided that the contents of the duplicate **SecConnectAuthenticate** validates correctly according to the specified cryptographic algorithms.

<11> Section 3.3.5.1: If the Office Groove Server 2007 or Groove Server 2010 relay server finds incorrect version numbers in the **SecConnect** security message header, or encounters any error in parsing the message, it embeds a **SecConnectResponseDeviceRegistrationNeeded** security message in a **ConnectResponse** command, sets the **ResponseID** field in the **ConnectResponse** to **Ok** (see [MS-GRVSSTP] for more details), and then sends the **ConnectResponse** to the client.

<12> Section 3.3.5.4: In the Office Groove Server 2007 or Groove Server 2010 relay server, when verifying the device nonce from the **SecAttachAuthenticate** message, the relay server checks the device nonce only if it has generated one and sent the client the **SecDeviceAccountRegisterResponse** message. If the relay server has not sent the **SecDeviceAccountRegisterResponse** message to the client in the same SSTP connection, it ignores the device nonce received from the **SecAttachAuthenticate** message.

<13> Section 3.3.5.5: In the Office Groove Server 2007 or Groove Server 2010 relay server, if the server finds incorrect version numbers in the **SecDeviceAccountRegister** security message header, or encounters any error in parsing the message, or detects any account layer message other than **SecAccountRegister** or **SecAccountOnNewDeviceRegister**, it ignores the Register command that contains the **SecDeviceAccountRegister** message.

7 Change Tracking

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

8 Index