Preamble to Video Services Forum (VSF) Technical Recommendation TR-06-2:2023

August 16, 2023

The attached document is a revision of the 2022 TR-06-2 RIST Main Profile Specification. This revision includes the following changes:

- Corrects an error in the formula for the M1 value in section D.2, page 51.
- A clarification regarding IV re-use has been added to section D.3.4.9.
- Some of the values in the example in section D.9 were incorrectly calculated; the example values have been fixed.

The 2022 revision includes support for the VSF EtherType. The earlier revisions of TR-06-2 used the experimental EtherTypes for the Reduced Overhead and Keep-Alive packets. VSF has now registered its own EtherType with the IEEE, and thus TR-06-2 has been revised to use this value.

The following changes and additions have been made to TR-06-2:2022:

- New packet formats using the VSF EtherType have been introduced for Reduced Overhead and Keep-Alive packets. This is done using an extensible format that allows for future uses of the VSF EtherType in a compatible manner.
- A mechanism to allow a device to compliant with TR-06-2:2022 to interoperate in a transparent manner with devices compliant with previous versions of the TR.
- Appendices have been renamed to Annexes, in order to bring the TR in line with published standards.
- Full documentation of the PSK Authentication protocol referenced in Section 7.5. This protocol can be found in Annex D.
- One normative difference in Section 5.6.2 pertains to the tunnel timeout. Previous versions of the TR fixed the timeout to 60 seconds. The 2022 version still recommends a default timeout of 60 seconds, but allows the implementations to use different values.
- Various language changes to clarify certain parts of the TR, without changing the protocol operation.

For additional information about the RIST Activity group, or to find out about participating in the development of future specifications, please visit http://vsf.tv/RIST.shtml

Video Services Forum (VSF) Technical Recommendation TR-06-2

Reliable Internet Stream Transport (RIST) Protocol Specification – Main Profile

Approved August 16, 2023

© 2023 Video Services Forum

This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-nd/4.0/

or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

INTELLECTUAL PROPERTY RIGHTS

THE FORUM DRAWS ATTENTION TO THE FACT THAT IT IS CLAIMED THAT COMPLIANCE WITH THIS RECOMMENDATION MAY INVOLVE THE USE OF A PATENT ("IPR") CONCERNING SECTIONS 5 (EXCEPT SECTION 5.6.4), 6, AND 7.

THE FORUM TAKES NO POSITION CONCERNING THE EVIDENCE, VALIDITY OR SCOPE OF THIS IPR.

THE HOLDER OF THIS IPR HAS ASSURED THE FORUM THAT IT IS WILLING TO LICENSE ALL IPR IT OWNS AND ANY THIRD PARTY IPR IT HAS THE RIGHT TO SUBLICENSE WHICH MIGHT BE INFRINGED BY ANY IMPLEMENTATION OF THIS RECOMMENDATION TO THE FORUM AND THOSE LICENSEES (MEMBERS AND NON-MEMBERS ALIKE) DESIRING TO IMPLEMENT THIS RECOMMENDATION. INFORMATION MAY BE OBTAINED FROM:

VIDEO-FLOW.LTD
11 HA'AMAL ST. ROSH HA'AYIN ISRAEL, 4809241

ATTENTION IS ALSO DRAWN TO THE POSSIBILITY THAT SOME OF THE ELEMENTS OF THIS RECOMMENDATION MAY BE THE SUBJECT OF IPR OTHER THAN THOSE IDENTIFIED ABOVE. THE FORUM SHALL NOT BE RESPONSIBLE FOR IDENTIFYING ANY OR ALL SUCH IPR.

THIS RECOMMENDATION IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NONINFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS RECOMMENDATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY MPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS RECOMMENDATION.

LIMITATION OF LIABILITY

VSF SHALL NOT BE LIABLE FOR ANY AND ALL DAMAGES, DIRECT OR INDIRECT, ARISING FROM OR RELATING TO ANY USE OF THE CONTENTS CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION ANY AND ALL INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, LOSS OF PROFITS, LITIGATION, OR THE LIKE), WHETHER BASED UPON BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING NEGATION OF DAMAGES IS A FUNDAMENTAL ELEMENT OF THE USE OF THE CONTENTS HEREOF, AND THESE CONTENTS WOULD NOT BE PUBLISHED BY VSF WITHOUT SUCH LIMITATIONS.

BY-ND

Executive Summary

Many solutions exist in the market for reliable streaming over the Internet. These solutions all use the same types of techniques, but they are all proprietary and do not interoperate with each other. This Technical Recommendation contains a protocol specification for reliable streaming over the Internet, so end users can mix and match solutions from different vendors.

Recipients of this document are requested to submit notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the Recommendation set forth in this document, and to provide supporting documentation.

BY-ND

Table of Contents

cc BY-ND

BY-ND

1 Introduction (Informative)

As broadcasters and other video users increasingly utilize unconditioned Internet circuits to transport high-quality video, the demand grows for systems that can compensate for the packet losses and delay variation that often affect these streams. A variety of solutions are currently available on the market; however, incompatibilities exist between devices from different suppliers.

The Reliable Internet Stream Transport (RIST) project was launched specifically to address the lack of compatibility between devices, and to define a set of interoperability points through the use of existing or new standards and recommendations.

1.1 Contributors

The following individuals participated in the Video Services Forum RIST working group that developed this technical recommendation.

Merrick Ackermans (CBS/Paramount)	Sergio Ammirata (SipRadius/AMMUX)	Paul Atwell (Media Transport Solutions)
Uri Avni (Zixi)	John Beer (QVidium)	Rishi Chhibber (Cisco)
Mike Coleman (AWS Elemental)	Eric Fankhauser (Evertz)	Ronald Fellman (QVidium)
Michael Firth (Nevion)	Rafael Fonseca (Artel)	Oded Gants (Zixi)
Nick Nicas (AT&T)	Ciro Noronha (Cobalt Digital)	Gijs Peskens (SipRadius)
Hermann Popp (Arri)	Steve Riedl (Turner)	Adi Rozenberg (AlvaLinks)
Manjinder Sandhu (Evertz)	Wes Simpson (Telecom Product Consulting)	Mikael Wånggren (Net Insight)

This technical recommendation builds upon VSF TR-06-1. The list of contributors to TR-06-1 can be found in section 1.1 of that document.

1.2 About the Video Services Forum

The Video Services Forum, Inc. (www.videoservicesforum.org) is an international association dedicated to video transport technologies, interoperability, quality metrics and education. The VSF is composed of service providers, users and manufacturers. The organization's activities include:

- providing forums to identify issues involving the development, engineering, installation, testing and maintenance of audio and video services;
- exchanging non-proprietary information to promote the development of video transport service technology and to foster resolution of issues common to the video services industry;
- identification of video services applications and educational services utilizing video transport services;

cc BY-ND

- promoting interoperability and encouraging technical standards for national and international standards bodies.

The VSF is an association incorporated under the Not For Profit Corporation Law of the State of New York. Membership is open to businesses, public sector organizations and individuals worldwide. For more information on the Video Services Forum or this document, please call +1 929-279-1995 or e-mail opsmgr@videoservicesforum.org.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except the Introduction and any section explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; Tables shall be next; followed by formal languages; then figures; and then any other language forms.

BY-ND

3 References

VSF TR-06-1, Reliable Internet Stream Transport (RIST) Protocol Specification – Simple Profile

IETF RFC 2784, Generic Routing Encapsulation (GRE)

IETF RFC 2890, Key and Sequence Number Extensions to GRE

IETF RFC 3550, RTP: A Transport Protocol for Real-Time Applications

IETF RFC 3686, Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)

IETF RFC 5054, Using the Secure Remote Password (SRP) Protocol for TLS Authentication

IETF RFC 5216, The EAP-TLS Authentication Protocol

IETF RFC 6347, Datagram Transport Layer Security Version 1.2

IETF RFC 7468, Textual Encodings of PKIX, PKCS, and CMS Structures

IETF RFC 8018, PKCS #5: Password-Based Cryptography Specification Version 2.1

IETF RFC 8086, GRE-in-UDP Encapsulation

IETF RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format

IETF RFC 8285, A General Mechanism for RTP Header Extensions

IEEE Std 802.1X-2010, Port-Based Network Access Control

SMPTE ST 2022-1:2007, Forward Error Correction for Real-Time Video/Audio Transport Over IP Networks

SMPTE ST 2022-2:2007, Unidirectional Transport of Constant Bit Rate MPEG-2 Transport Streams on IP Networks

J. Carlson, B. Aboba, and H. Haverinen, **EAP SRP-SHA1 Authentication Protocol**, retrieved from https://tools.ietf.org/html/draft-ietf-pppext-eap-srp-03

T. Wu, **SRP Protocol Design**, retrieved from http://srp.stanford.edu/design.html

BY-ND

4 RIST Profiles (Informative)

RIST has multiple operational profiles, corresponding to increasing levels of complexity and functionality. Higher profiles include all the features and functionality of the preceding profiles. This document defines RIST Main Profile, which adds the following features to VSF TR-06-1 RIST Simple Profile:

- Stream multiplexing support
- Tunneling support
- Encryption support using DTLS
- Pre-Shared Key encryption support
- NULL packet deletion (for bandwidth optimization)
- High bitrate/high latency operation

A profile roadmap is included in TR-06-1.

5 Stream Multiplexing and Tunneling Support

The objective of stream multiplexing and tunneling is to provide the ability to combine all the communication between two RIST devices into a single UDP port, to which encryption can be applied. Encryption is provided either by DTLS as described in section 6, or by Pre-Shared Key, as described in section 7. The use of tunneling also simplifies firewall configuration. The features provided are:

RIST devices compliant with this specification shall support all the functions below:

- Combining the RTP and RTCP flows into a single port, in a manner compliant with RIST Simple Profile.

Optionally, RIST devices compliant with this specification may support the functions below:

- Combining multiple RIST flows into a single port.
- Providing support for transporting generic IP traffic into that same socket, in a manner similar to a VPN, typically for in-band control of remote devices (e.g., SNMP management).

The functions above shall be achieved using GRE-over-UDP per RFC 8086, with the constraints and additions described below.

BY-ND

5.1 General Operation

A tunnel is established between two endpoints. The endpoint that listens for a connection is called the server, and the endpoint that initiates the tunnel connection is called the client. Operation shall follow RFC 8086, with the additions, changes and exceptions indicated below:

- The roles of RIST sender and receiver are independent of the roles of server and client. The device that starts the tunnel is known as the client. Once the tunnel is established, streams may flow in either direction. Tunnel establishment is described in section 5.6.
- Streams running through the tunnel shall comply with VSF TR-06-1, RIST Simple Profile.
- RIST devices may use arbitrary UDP port numbers for the tunnel. RFC 8086 recommends the use of port 4754 if the traffic is in the clear, or port 4755 if the traffic is encrypted using DTLS, but RIST devices are not constrained by these choices.
- The server shall listen for GRE packets on a UDP port that has been pre-configured by the user. The server receives packets from the client on this port. The client may use any port number as its source port. The server shall direct reply/return packets to that source port number on the client.
- RIST devices may use a tunnel to send multiple RTP/RTCP flows.
- RIST devices may allow the tunnel to be used for other types of traffic, e.g., for in-band control. If such a function is provided, it shall be possible for the user to configure what types of traffic are allowed in the RIST tunnel, or to completely exclude non-RIST traffic from the tunnel.
- RIST devices shall discard unsupported tunneled packets. Section 5.6.3 allows the use of GRE packets for the tunnel keep-alive function; unsupported tunneled packets, even though discarded, shall be deemed to be keeping the tunnel alive if present.
- When transmitting, devices compliant with this specification should set the C (Checksum) field in the GRE header to zero to reduce overhead. The S (Sequence Number) and K (Key) shall be used as follows:
    - When using a pre-shared key with RIST, as described in section 7, the S field shall be set to 1 and a valid sequence number shall be included in the packet. The K field shall also be set to 1. The usage of the S field is described in section 7.1.
    - Transmitting devices not using pre-shared key shall set the K field to zero.
    - If the communicating devices support non-RIST traffic in the tunnel, the S field should be set to 1 and a valid sequence number should be included in the packet.
    - In all other situations, the use of the S field is optional.
- When receiving, devices compliant with this specification shall inspect the C, K and S bits in order to compute the GRE header size. Receiving devices may or may not actually process and verify the Checksum field. Devices not working in Pre-Shared Key mode (section 7) shall not be required to process the Key field. If the Sequence Number field is present (S=1), receiving devices should use it to re-order the tunnel packets.

BY-ND

- RFC 8086 defines bits 4-12 in the GRE header as **Reserved0**. This Specification defines bits 9-12 as follows (see Figure 1):
  - **RV** (bits 10-12): this field indicates the RIST Version, as follows:
    - RV = 000: TR-06-2:2020
    - RV = 001: TR-06-2:2021
    - RV = 010: TR-06-2:2022
  - **H** (bit 9): this field indicates the AES key length for PSK operation, as follows:
    - H = 0: 128-bit AES key
    - H = 1: 256-bit AES key
- The definition of the **H** bit shall be as follows:
  - If RV = 000, the **H** bit shall be set to zero on transmission and ignored on reception. In this case, the PSK key length shall be determined by out-of-band means.
  - If RV = 001 or higher, the **H** bit indicates the PSK key length. Devices not operating in PSK mode shall set the **H** bit to zero on transmission and ignore it on reception.
- The remaining **Reserved0** bits in Figure 1 shall be set to zero on transmission and ignored on reception, as per RFC 2784.

Upon receiving a packet with an RV value not listed above, the device shall process it as follows:

- If RV=011 or RV=100, the device shall assume that the packet format is backward compatible with TR-06-2:2022 and process it according to this Specification.
- If RV=101, RV=110 or RV=111, the device shall assume that the packet format is unknown and discard it.

A GRE header with no options is depicted in Figure 1 (fields are in network byte order, MSB first):

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |0| |0|0|Reserved0|H| RV  | Ver |         Protocol Type         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: GRE header with no options

A GRE header with sequence number is depicted in Figure 2:

cc BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| |0|1|Reserved0|H| RV  | Ver |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Sequence Number                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: GRE header with sequence number

5.2 Use of the VSF EtherType in GRE Packets

The Video Services Forum has been assigned the 0xCCE0 EtherType. In the GRE header, the Protocol Type field is set to the EtherType of the protocol being carried. Senders shall include the VSF Packet Header depicted in Figure 3 after the GRE header for packets using the VSF EtherType in the Protocol Type field.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       VSF Protocol Type       |      VSF Protocol Subtype     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: VSF Packet Header

The fields in the VSF Packet Header shall be set as follows:

- **VSF Protocol Type:** This field defines the protocol stack for the packet. It shall be set by the sender as follows:
    - 0x0000: RIST Packet, defined by this Specification.
    - 0x0001-0xFFFF: Reserved for future assignment by the VSF.
- **VSF Protocol Subtype:** This field defines the packet format. For RIST Packets (VSF Protocol Type set to 0x0000), this field shall be set by the sender as follows:
    - The range 0x0000 to 0x7FFF is assigned to data packets. The following values are defined:
        - 0x0000: RIST Main Profile Reduced Overhead Packets, defined in Section 5.3.2.
        - 0x0001-0x7FFF: Reserved for future use.
    - The range 0x8000 to 0xFFFF is assigned to control packets. The following values are defined:
        - 0x8000: RIST Main Profile Keep-Alive Packets, defined in Section 5.6.3.
        - 0x8001: RIST Main Profile Future Nonce Announcement Packets, defined in Section 7.6.
        - 0x8002-0xFFFF: Reserved for future use.

Senders shall not set RV=000 or RV=001 in packets where Protocol Type=0xCCE0.

BY-ND

Figure 4 shows a full packet with the GRE header, the VSF Packet header, and the TR-06-2 payload.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:            GRE Header with Protocol Type = 0xCCE0             :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       VSF Protocol Type       |      VSF Protocol Subtype     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                        TR-06-2 Payload                        :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Full GRE Header with VSF Packet Header

5.3 Tunneling Modes

Two tunneling modes are specified in this document:

- **Full Datagram Mode:** This mode shall be supported in all implementations of this specification. In this mode, the GRE payload is a full layer-3 IP packet. All RIST devices compliant with Main Profile shall support this mode as a means of stream multiplexing. If the RIST device supports encapsulation of non-RIST traffic, this feature shall be disabled by default, and shall be enabled only by explicit user intervention.
- **Reduced Overhead Mode:** In this mode, a reduced header as defined in section 5.3.2 shall be used. Previous versions of this Specification indicated the use of an experimental EtherType for Reduced Overhead Packets. This usage is deprecated, and new implementations should use the VSF EtherType.

5.3.1 Full Datagram Mode

In this mode, a full IP packet shall be encapsulated as the GRE payload, starting from the IP header. The **Protocol Type** field in the GRE header shall be set to 0x0800, the default EtherType for IP.

5.3.2 Reduced Overhead Mode

In this mode, the encapsulated packet is assumed to be a UDP packet. The packet payload shall start with the subset of the UDP header indicated in Figure 5, denoted as “Reduced UDP Header”. The remainder of the packet payload shall be the full, unchanged payload of the original UDP packet. Fields shall be in network byte order, MSB first.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        UDP Source Port        |      UDP Destination Port     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Reduced UDP Header

BY-ND

Figure 6 indicates two options for GRE encapsulation of Reduced Overhead Mode:

- The Legacy Encapsulation was specified in TR-06-2:2020 and TR-06-2:2021. It uses the experimental EtherType value of 0x88B6. Usage of this encapsulation is deprecated and no longer recommended. Implementers may consider supporting this encapsulation for interoperability reasons.
- The Recommended Encapsulation uses the VSF EtherType value of 0xCCE0.

Legacy Encapsulation used in TR-06-2:2020 and TR-06-2:2021

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             GRE Header with Protocol Type=0x88B6              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        UDP Source Port        |     UDP Destination Port      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          UDP Payload                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Recommended Encapsulation

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             GRE Header with Protocol Type=0xCCE0              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   VSF Protocol Type=0x0000    |  VSF Protocol Subtype=0x0000  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        UDP Source Port        |     UDP Destination Port      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                          UDP Payload                          :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Reduced Overhead Mode GRE Encapsulation

The receiving RIST device shall make the following assumptions for an incoming Reduced Overhead packet:

- The payload following the Reduced UDP Header represents the payload of a UDP packet.
- The receiving tunnel endpoint shall assume that the packet is destined for it and sourced from the remote tunnel endpoint.
- The other IP header fields shall be assumed to be the same as the IP packet bearing the GRE payload, if relevant.
- Because the checksum field is not present, receiving RIST devices shall assume that the checksum of the encapsulated UDP packet is correct.

BY-ND

- The payload size of the encapsulated UDP packet shall be derived from the payload size of the received GRE packet. More specifically, the payload size of the encapsulated UDP packet shall be assumed to be equal to the payload size of the received UDP packet minus the GRE header size (4, 8 or 12 bytes) minus the Reduced UDP Header size (4 bytes) and minus the VSF Packet Header size, if present (4 bytes).
- The source and destination UDP ports for the encapsulated packet shall be the source and destination UDP ports from the Reduced UDP Header.

5.4 Processing of Tunnel Packets at the Receiving End

The RIST device receiving GRE encapsulated packets shall process them as follows:

- **Reduced Overhead Mode:**
  - The receiving device shall assume that the encapsulated packet is destined for it.
  - The receiving device shall process the packet payload in the same way as if it had received a UDP packet addressed to it, from the sender of the GRE packet, with source and destination UDP ports as specified by the Reduced UDP Header.
  - For gateway devices, received Reduced Overhead streams may be forwarded to external devices by a configuration means outside the scope of this Specification.
- **Full Datagram Mode:** In this mode the receiving device will extract a layer-3 IP packet from the GRE tunnel. This packet shall be known in this document as an "Extracted Packet".
  - The reception of any Extracted Packet from the GRE tunnel shall be deemed sufficient for keep-alive purposes, even if the Extracted Packet is discarded.
  - Receiving devices shall accept and process the Extracted Packets, in the same manner as if they had been directly received by a local network interface, if all of the following conditions are true:
    - The Extracted Packet is a UDP packet.
    - The destination UDP port in the Extracted Packet is for a socket/flow the receiving device is currently configured to accept and process.
    - The destination IP address in the Extracted Packet matches an address the receiving device is prepared to accept. This includes multicast addresses that the receiving device has been configured to receive, as well as any unicast IP addresses deemed acceptable by the receiving device.
  - Receiving devices may discard Extracted Packets that do not match the above rules.
  - Receiving devices may choose to check the IP header checksum for the Extracted Packet. If the Extracted Packet is a UDP packet, the receiving device may choose to check the UDP checksum (if present).
  - If a receiving device accepts Extracted Packets, the following rules shall apply:
    - Processing shall be disabled by default and shall only enabled by explicit user configuration.

CC BY-ND

    - Forwarding of Extracted Packets into the local networks connected to the receiving device shall be disabled by default and shall be enabled only by explicit user configuration.

Since different EtherTypes are used for Reduced Overhead and for Full Datagram modes, it is possible for a tunnel to contain both types of packets simultaneously. Support for such mixed operation is optional.

5.5 Tunnel-Level Multi-Path Operation

In some applications, the GRE packets can be sent over multiple physical or logical paths to the receiver. This mode of operation is used in the following scenarios:

- Bonding: the GRE packets are spread over multiple paths to combine them into a higher capacity link.
- Seamless switching: the GRE packets are replicated over multiple paths for redundancy, in the same fashion as SMPTE 2022-7.

Senders using tunnel-level multi-path operation should set S=1 in the GRE header and include valid sequence numbers. Receivers should use the sequence number to re-order the GRE packets.

5.6 Tunnel Establishment

5.6.1 Introduction (Informative)

When using an RFC 8086 tunnel, one of the endpoints is the server (listens on some UDP port for tunnel packets) and the other endpoint is the client (actively sends RFC 8086 packets to the server). The roles of tunnel client and server are independent of the roles of RIST sender and receiver. This is further complicated when NAT traversal is required at either end.

In VSF TR-06-1 RIST Simple Profile, the receiver is the server, and the sender is the client, as far as stream transmission is concerned. If the same roles apply for the tunnel (i.e., the RIST receiver is the tunnel server, and the RIST sender is the tunnel client), operation is straightforward - the RIST sender starts stream transmission at its convenience - the only difference is that the packets come out encapsulated in RFC 8086.

However, if the RIST receiver is the tunnel client and the RIST sender is the tunnel server, there is a startup problem because there is no negotiation in RFC 8086, and in RIST Simple Profile the receiver does not send anything until it starts receiving from the sender. The same problem exists when the device is a gateway and the RIST streams are not active or have not been configured.

The solution to this issue is to require the tunnel client to send some sort of tunnel-level keep-alive message. This way, the RIST sender becomes aware that the tunnel is up, learns the IP address of the client, and it can start sending at its convenience.

BY-ND

As far as this problem is concerned, an empty message is sufficient. However, adding this message presents an opportunity to include additional desirable functionality in RIST. This Specification defines a keep-alive message in sections 5.6.3 and 5.6.4, but allows the use of any type of periodic GRE-encapsulated message, as long as the requirements of section 5.6.2 are met.

5.6.2 Keep-Alive Message Requirements

The following are the requirements for the keep-alive messages:

- The tunnel client shall start sending messages to the tunnel server as soon as it is enabled.
- The tunnel server shall start sending messages to the tunnel client as soon as it receives the first message from it.
- Keep-alive messages shall be sent periodically. Transmission frequency shall be between 1 second and 10 seconds.
- Tunnel timeout should be 60 seconds. Tunnel timeout shall be declared when the endpoint fails to receive any data from the tunnel (either keep-alive messages and/or traffic) for the specified amount of time.
- If a tunnel endpoint is receiving keep-alive messages or traffic from the remote tunnel endpoint, the tunnel endpoint shall keep sending keep-alive messages or traffic to the remote tunnel endpoint. If the tunnel endpoint stops receiving keep-alive messages or traffic from the remote tunnel endpoint for a period of time equal to the tunnel timeout, the tunnel endpoint shall stop sending keep-alive messages and traffic to the remote tunnel endpoint.
- If an endpoint fails to receive either a keep-alive message or traffic on a session after the timeout, the endpoint shall consider the session to be terminated and shall release any resources associated with the session.
- At startup, the tunnel client shall send a minimum of 3 and a maximum of 10 back-to-back keep-alive messages to the tunnel server to get the connection started.

The requirements of this section shall apply to whatever periodic message is used for the keep-alive function.

5.6.3 Keep-Alive Message Format

Figure 7 shows two options for encapsulating the Keep-Alive message:

- The Legacy Encapsulation was specified in TR-06-2:2020 and TR-06-2:2021. It uses the experimental EtherType value of 0x88B5. Usage of this encapsulation is deprecated and no longer recommended. Implementers may consider supporting this encapsulation for interoperability reasons.
- The Recommended Encapsulation uses the VSF EtherType value of 0xCCE0.

Senders shall set the C bit in the GRE header to zero. Senders shall set the S and K according to Section 5.1.

CC BY-ND

The Keep-Alive Message is shown Figure 8. The message fields are indicated in network byte order, MSB first.

Legacy Encapsulation used in TR-06-2:2020 and TR-06-2:2021

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             GRE Header with Protocol Type=0x88B5              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                      Keep-Alive Message                       :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Recommended Encapsulation

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             GRE Header with Protocol Type=0xCCE0              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   VSF Protocol Type=0x0000    |  VSF Protocol Subtype=0x8000  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :                      Keep-Alive Message                       :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Keep-Alive Message Encapsulation

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 |   48-bit MAC Address          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                               |X|R|B|A|P|E|L|N|D|T|V|J|F|Rsvd1|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                Message Payload (JSON format)                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Keep-Alive Message

The sender shall set the message fields as follows:

- **48-bit MAC Address:** this field should be set to one of the MAC addresses of the device sending the packet. The MAC address is used for identification purposes and shall be unique for all devices participating in a RIST Main Profile session. RIST devices implemented in virtual machines shall take special care to ensure the uniqueness of this field.
- **Capability Flags:** these flags shall indicate the enabled capabilities of the device transmitting the message, as follows:

BY-ND

    - **X:** More capabilities. If this flag is set, it indicates that there are more capabilities included in the JSON message. This is reserved for future use.
    - **R:** Routing capability. If this flag is set, the device is willing to transmit and receive non-RIST traffic. If it is not set, the remote device shall not transmit non-RIST traffic. Devices operating with **R**=0 shall discard all non-RIST packets received in the tunnel. Devices operating with **R**=1 should set the **S** flag in the GRE header and should include a valid sequence number in that header.
    - **B:** If this flag is set, device supports Bonding (as specified in RIST Simple Profile)
    - **A:** If this flag is set, device supports Adaptive Encoding
    - **P:** If this flag is set, device supports SMPTE-2022 FEC
    - **E:** If this flag is set, device supports seamless redundancy switch as per SMPTE-2022-7 (already in RIST Simple Profile)
    - **L:** If this flag is set, device supports load sharing. This is reserved for future use.
    - **N:** If this flag is set, device supports NULL packet deletion (described in section 8.3).
    - **D:** If this flag is set, this is a Disconnect message (described in section 5.6.5).
    - **T:** If this flag is set, this is a Reconnect message (described in section 5.6.6).
    - **V:** If this flag is set, device supports Reduced Overhead Mode (described in section 5.3.2).
    - **J:** If this flag is set, device is capable of sending, receiving and processing JSON information (described in section 5.6.4).
    - **F:** If this flag is set, the device is capable of on-the-fly PSK passphrase change (described in section 7.4).
- **Rsvd1:** these bits are reserved for future capabilities. Current implementations shall set them to zero on transmission and ignore them on reception.

As indicated in section 5.6, this Specification allows devices to use any periodic GRE-encapsulated packet as the keep-alive message. If a device does not receive periodic keep-alive messages that comply with the format described in this section, it shall make the following assumptions:

- The sending device is not capable of sending or receiving JSON information – the **J** flag in Figure 8 is assumed to be zero.
- The sending device will never issue Disconnect and Reconnect messages (sections 5.6.5 and 5.6.6) and will ignore such messages from the other side of the tunnel.
- The information that would otherwise be indicated by the remaining capability flags in Figure 8 is unknown. If both sides are manually configured for one or more such capabilities, the use of the manually configured capabilities shall be allowed. In the absence of manual configuration, the endpoint shall assume that the feature is not supported (i.e., the corresponding flag is zero).

BY-ND

- The MAC Address of the remote device is unknown.

5.6.4 Keep-Alive Message Payload

The keep-alive message payload shall be in JSON format for extensibility, as defined in RFC 8259. Receivers with JSON support shall parse the message and shall discard any unsupported/unknown data. The JSON snippet below is an example of the minimum supported data set:

```
{
  "tunnelIP": "10.0.0.2",
  "remoteIP": "10.0.0.3",
  "excludedIP": ["192.168.1.0/24", "10.10.10.0/25"],
  "routing": true,
  "pskRotation": 600,
  "vendor": {
   "implementation": {
      "version": "2.3.5",
      "product": "Yellow RIST Machine",
      "vendorName": "RIST AG, Inc."
    },
    "features": null
  }
}
```

The parameters of the JSON keep-alive message shall be defined as follows:

- **tunnelIP:** shall be used to communicate the local tunnel IP address to the remote endpoint. It may be either an IPv4 or IPv6 address. The tunnel client may also use the values **allocateIPv4** or **allocateIPv6** to ask the server to allocate a local tunnel IP address for its use. In this case, the server may use the supplied MAC address to ensure that a given client gets the same tunnel IP address every time it connects.
- **excludedIP:** this optional parameter is a list of IP address ranges that the client is not willing to accept. It shall only be used in client-to-server messages.
- **remoteIP:** If the client has requested the server to allocate an IP address for its use, this field shall contain the allocated address. If it is present in the client-to-server message, it shall be ignored by the server. It shall only be used in server-to-client messages.
- **routing:** this optional Boolean parameter shall be used to indicate that the sender of the message is or is not willing to accept and route non-stream traffic. If the **routing** Boolean parameter is included, the following shall be implemented:
  - If the parameter is set to **false**, the device shall not allow routing of non-RIST traffic; the **R** flag in the capabilities header shall be ignored.
  - If the parameter is set to **true**, then if and only if the **R** flag is also set to 1, then the device will allow routing of non-RIST traffic.

BY-ND

This parameter shall be used to turn off the routing of non-RIST traffic if the client and server cannot agree on the client's IP address.

- **pskRotation:** If the tunnel is operating in Pre-Shared Key (PSK) mode, as described in section 7, the sender may advertise its key rotation period, expressed in seconds, using this optional field.
- **vendor:** these optional strings provide information about the device itself:
    - **version:** software/firmware version number (arbitrary vendor-defined format).
    - **product:** product name (arbitrary vendor-defined format).
    - **vendorName:** name of the device vendor (arbitrary vendor-defined format).
- **features:** This is a placeholder for extensions of the capability flags.

5.6.4.1 *Example Exchange (Informative)*

Example of a client-to-server message where the client requests that the server allocate an IPv4 tunnel address:

```
{
 "tunnelIP": "allocateIPv4",
 "vendor": {
   "implementation": {
     "version": "2.3.5",
     "product": "Yellow RIST Machine",
     "vendorName": "RIST AG, Inc."
   },
   "features": null
 }
}
```

Upon receiving this message, the server will respond as follows:

```
{
 "tunnelIP": "10.0.0.2",   ←this is the server's local IP address
 "remoteIP": "10.0.0.3",   ←this is the IP address the server has assigned to the client
 "vendor": {
   "implementation": {
     "version": "2.3.5",
     "product": "Yellow RIST Machine",
     "vendorName": "RIST AG, Inc."
   },
   "features": null
 }
}
```

In the above exchange, if the client wanted to declare its IP address instead of asking the server to allocate one, it would use this IP address instead of **allocateIPv4**. In this case, the server's response would not include the **remoteIP** entry.

CC BY-ND

It is possible that the client and server cannot agree on a set of IP addresses. This will happen in the following situations:

1. Both server and client want to use specific IP addresses, and the selected values are not acceptable to one of the sides.
2. The client asks the server to allocate an IP address, but sends a list of excluded ranges that matches the ranges that that server was planning to use for the client.

In these cases, routing is not possible. The endpoint that disagrees with the addresses will send a JSON message with **`"routing": false,`** and from that point on the GRE tunnel described in this specification will be used only for stream multiplexing**.** In other words, the **`"routing"`** JSON parameter is used to disable non-RIST traffic between endpoints that are otherwise willing to support it but cannot agree on IP address assignment.

The protocol exchanges in this example are as follows:

1. Specific IP addresses:
    - The client sends the initial keep-alive message(s) with its desired IP address.
    - If the server finds the address unacceptable, it will send its keep-alive message with **`"routing": false`** and its own IP address.
    - The server may find the IP address of the client acceptable, but the client may decide that the IP address of the server is unacceptable. From that point on, it must send **`"routing": false`** in its keep-alive messages.
2. Server-allocated addresses:
    - The client sends the initial keep-alive message with an **`excludedIP`** range.
    - The server is unable to allocate an address that satisfies the client's request. It will then send an IP address that may violate the request, and will qualify that with **`"routing": false`**.

Note that routing operation does not require JSON support or even IP address negotiation. Endpoints may be manually configured with consistent IP addresses (and routing tables if appropriate). In such cases, it is legal to have **R**=1 without JSON support (**J**=0).

5.6.5 Disconnect Message

Use of the Disconnect Message is optional but recommended.

Note: Implementers are cautioned that receivers may not make use of this message.

Section 5.6.2 indicates that the tunnel will disconnect on keep-alive message timeout. The keep-alive message header contains a **D** bit that may be used to explicitly request a disconnect. Either the client or the server may initiate a disconnect by sending a keep-alive message with **D**=1. As a response, the receiving device should send up to 3 keep-alive messages with **D**=1 as an

BY-ND

acknowledgement, terminate the tunnel and stop sending further messages. The device that initiated the disconnection shall terminate its side of the tunnel and shall stop sending messages as soon as it receives a keep-alive message with **D**=1.

All Main Profile RIST devices should implement support for receiving and processing keep-alive messages with **D**=1 as described in this section. A client device receiving a disconnect message should wait 5 seconds before attempting to connect again to the same server.

A RIST device that intends to terminate the connection may explicitly use the Disconnect Message or it may simply stop transmitting and let the connection terminate by timeout.

5.6.6 Reconnect Message

The keep-alive message header includes the **T** bit to restart the IP Address negotiation described in section 5.6.4. It can be initiated either from the server or from the client. If it is initiated from the server, it shall mean "connect again". The primary purpose of this mechanism is for an endpoint to change its tunnel IP address. In a situation where the server is allocating IP addresses and reconnection is initiated by the client, it is recommended that the server should allocate a different IP address to the client.

The behavior of devices with respect to the Reconnect Message shall be as follows:

- The device requesting reconnection shall start sending keep-alive messages with **T**=1
- The remote device, upon receiving **T**=1, shall restart the IP address negotiation, and shall send its next message with **T**=1.
    - If the client started the reconnection, the server shall treat the received message in the same fashion as an initial connection request.
    - If the server started the reconnection, the client shall consider the connection closed and start it again, transmitting the initial keep-alive message with **T**=1.
- The originating device shall respond with **T**=0, terminating the negotiation.
- Upon receiving a message with **T**=0, the remote device shall also set **T**=0 on its messages.

5.6.7 Source and Destination IP Addresses in Tunneled RIST Packets

This section specifies the rules for selecting and processing the source and destination IP addresses for the GRE-encapsulated IP packets in Full Datagram mode. There are two cases to be considered:

- **Case 1:** the endpoints have consistent and agreed upon tunnel IP addresses. This can be achieved either by JSON negotiation with keep-alive messages, or by manual static configuration. In both cases, each endpoint knows the tunnel IP address of the other endpoint.

BY-ND

- **Case 2:** the endpoints have not completed IP address negotiation, either because they tried and failed, or because they do not support it. Each endpoint has a tunnel IP address, but does not know (or does not accept) the other endpoint tunnel address.

For **Case 1**, there are no restrictions on source and destination addresses. Since the networks are consistent, users are free to configure whatever addresses they may see fit.

For **Case 2**, since the tunnel IP addresses are either not known or not consistent, the following rules shall be followed:

- The destination IP address of the transmitted RIST RTP packets shall be multicast.
- The RIST Simple Profile rules for multicast shall apply. (They are repeated here for the convenience of the reader, however implementers are encouraged to read the latest version of VSF TR-06-1):
    - RTCP packets, both from the RIST sender and from the RIST receiver, shall be transmitted to the same multicast address as the RTP flow.
    - RTCP packets, both from the RIST sender and from the RIST receiver, shall be transmitted to UDP port P+1, where P is the destination UDP port of the corresponding RTP flow.
- The source IP address of the packets should be set to the transmitting endpoint’s tunnel address.
- When differentiating between streams, a receiver shall use both the multicast destination address and the UDP port. In other words, receivers shall be required to support situations where multiple streams use the same UDP destination port and different multicast destination addresses.

Note: Implementers may use simplified configuration interfaces for ease of use. For example, a device that combines the tunnel with the RIST Simple Profile sender (e.g., a multi-channel encoder) may only expose a list of UDP ports, one per stream, and use a pre-selected default multicast for the tunnel, and a pre-selected tunnel IP address. Conversely, a combined tunnel/RIST Simple Profile receiver (e.g., a multi-channel decoder) may automatically detect the multicast and port. In cases where such pre-selected defaults are used, the device’s user interface shall provide some indication of what values are being used for ease of interoperation.

Main Profile tunnel gateways (i.e., devices that only implement the tunnel and optionally the encryption functions and forward the RIST traffic to external RIST devices) may forward the multicast unchanged, but should remap the source IP address to avoid issues with Reverse Path Forwarding.

Devices that do not use the keep-alive message defined in this document shall fall into Case 1 if they have static configurations on both endpoints or into Case 2 if they do not.

BY-ND

5.6.8 Keep-Alive Message Fragmentation

Senders shall not fragment IP-level keep-alive message. The keep-alive message plus the GRE header plus the DTLS header (if using encryption) plus the UDP/IP headers for the resulting packet shall be placed into a single MTU. If the sender of the keep-alive message needs to send a JSON message that does not fit into a single packet, the JSON message shall be broken into multiple, legal, separate JSON messages, each with a subset of the data.

6 DTLS Support

RIST senders and receivers may use DTLS version 1.2 to secure their communication and authenticate the endpoints. RIST devices offering DTLS support shall implement the DTLS protocol according to the recommendations of this section. Implementations shall follow RFC 6347 with the additional restrictions described in this document.

RIST devices using DTLS shall implement tunneling as described earlier in this document.

6.1 Session Establishment

DTLS sessions shall be established as follows:

- There shall be one single DTLS session carrying the RFC 8086 tunnel packets described earlier in this document.
- Once negotiation is complete, the RIST sender shall use RIST Simple Profile as per VSF TR-06-1 over the RFC 8086 tunnel, as described in Section 5.

Note: The roles of DTLS Server and Client are independent of the roles of RIST Sender and Receiver

Once the session is established, the final tunneled and encrypted packet size should not exceed the path MTU.

Note: For RIST Simple Profile flows using transport streams over Ethernet, this is guaranteed because there are no more than seven transport packets per RTP packet, which will leave enough space for the additional tunnel headers.

6.2 Supported DTLS Cipher Suites

The following cipher suites shall be supported by all RIST devices implementing DTLS:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_NULL_SHA256

RIST devices shall provide a means for the user to disable individual cipher suites to match their local policies.

BY-ND

Note: It is understood that disabling individual ciphers may prevent two RIST devices from establishing communication, if there is no common cipher enabled.

RIST devices may include other cipher suites specified in DTLS in their implementations.

6.3 Certificate Configuration

- The DTLS server should be configured with a certificate file – either issued by a certificate authority, or a self-signed one. There is no limitation to the certificate type, as long as it is readable by the DTLS library.
- The DTLS client may validate the authenticity of the certificate and may perform hostname validation. If offered, this shall be a user-configurable option.
- The DTLS client should be configured with a client-side certificate. This can be done using username/password.
- The DTLS server may validate the client certificate. If offered, this shall be a user-configurable option,
- Both client and server should use a certified list of CAs. This file may be taken dynamically from the following link: https://hg.mozilla.org/releases/mozilla-beta/file/tip/security/nss/lib/ckfw/builtins/certdata.txt

  If offered, the use of a certified list of CAs shall be a user-configurable option.
- When using self-signed certificates, it is up to the two end points to arrange exchange of the custom proprietary CA used to create such certificates.

6.4 TLS-SRP Support

The TLS-SRP protocol, as described in RFC 5054, is used to securely provide username/password authentication between devices, as an alternative to using certificates. RIST devices may implement TLS-SRP as an authentication method. If they do so, the implementation shall follow RFC 5054, with the following modifications and restrictions:

- RIST devices implementing TLS-SRP shall support the following cipher suites:
  - TLS_SRP_SHA_WITH_AES_128_CBC_SHA
  - TLS_SRP_SHA_WITH_AES_256_CBC_SHA
- RIST devices implementing TLS-SRP may additionally support any of the other cipher suites listed in RFC 5054 section 2.7.
- RIST servers implementing TLS-SRP shall be configured with a certificate file. This certificate file may be self-signed. RIST clients with TLS-SRP support may safely ignore the certificate expiration date without compromising security.
- In order to make TLS-SRP more secure, RIST servers should implement the following policies:
  - For a given server, user names should be unique across accounts.

BY-ND

- Servers should limit the rate of authentication attempts from a particular IP address in order to reduce the risk of brute-force password attacks.
- Servers should have reasonable password strength policies in order to reduce the risk of brute-force password attacks.

7 Pre-Shared Key Encryption Support

RIST senders and receivers may use Pre-Shared Key Encryption to secure their communication and authenticate endpoints. When offering PSK encryption, the devices shall implement the protocol according to the recommendations of this section. The GRE header structures described in section 5.1 of this document contain the information used to generate and rotate keys and initialization vectors (IV). With these keys and IV, RIST devices shall encrypt/decrypt the GRE payload using AES-128/256-CTR. The PSK mechanism described below may be used in either Full Datagram or Reduced Overhead modes as specified in section 5.2 of this document. The PSK mechanism shall apply to the payload of the GRE packet – the GRE header shall be transmitted in the clear.

7.1 GRE Header with K Field Turned On

Figure 9 shows the GRE header, as pictured in section 5.1 of this document with the optional field K included. Fields are in network byte order, MSB first.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| |1|1|Reserved0|H| RV  | Ver |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Key/Nonce                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9: GRE header with Key/Nonce

7.2 Sequence and Nonce

- When transmitting, devices compliant with this specification shall set the Key field to a random, non-zero nonce. When a non-zero key is detected, the PSK option shall be enabled.
- The entire payload of the GRE packet, not including the GRE header, shall be encrypted between all the endpoints of the tunnel using an AES key derived from the Key field plus a pre-shared passphrase. The AES key length shall be signaled in the **H** bit in the header, as described in Section 5.
- When the sender enables the PSK option by setting the non-zero K field, it shall also set the S (sequence number). The 128-bit initialization vector (IV) for the encryption

BY-ND

operation shall be derived by using the 32-bit Sequence Number field as its most significant four bytes, followed by 12 bytes of zeros. This 128-bit quantity is the counter that generates the key stream. The process is illustrated in Figure 10, in network byte order.

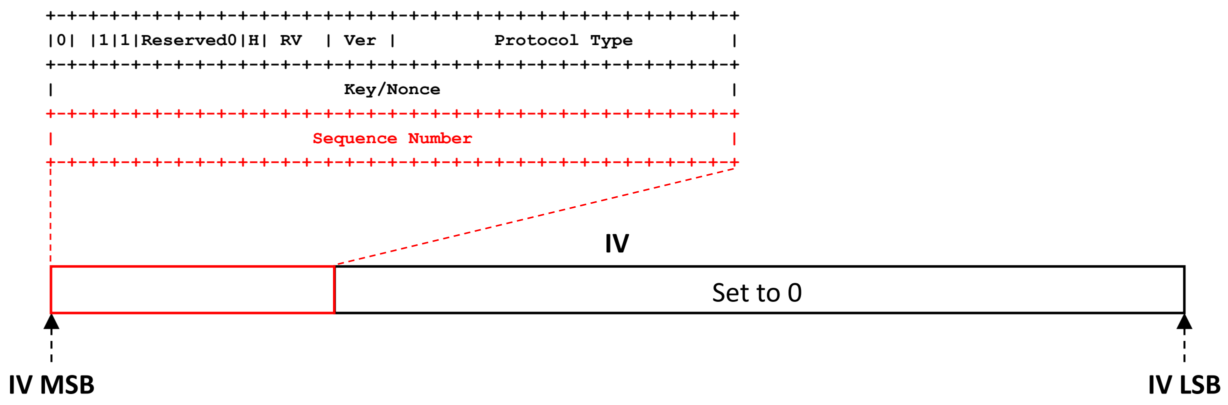

Figure 10: IV Generation

- VSF TR-06-2:2020 defined a different arrangement for the IV derivation from the Sequence Number, whereby the 32-bit Sequence Number represented the LSB of the 128-bit IV. This arrangement is insecure and is not interoperable with what is defined in this Specification. Receivers can inspect the RV field in the GRE header to determine the sender is using the insecure version, which will have RV=000. Receivers compliant with this Specification shall operate as follows:
    - If the receiver allows interoperation with insecure implementations with RV=000, such operation shall be disabled by default. It shall only be enabled by explicit operator intervention.
    - If the receiver is interoperating with an insecure implementation, it should alarm or notify the operator of this condition.
    - If the receiver does not allow interoperation with insecure implementations and detects that the incoming packets are insecure, it should alarm or notify the operator of this condition.
- A new nonce shall be generated by the sender at least every time the sequence counter/number of the GRE packet wraps to zero. This ensures that the same IV + Key combination is never reused. The sender may rotate the key more often than that.
- The receiver shall inspect the Nonce field for every received packet, and shall re-generate the key any time it changes. If RV=001 or higher, the AES key size shall be determined by the **H** bit in the GRE header.

7.3 AES Encryption Key and Sequences

- AES Encryption Key and Sequences shall be used. The payload shall be encrypted and decrypted using the Advanced Encryption Standard (AES) and Counter mode (CTR) as described in RFC 3686 section 2.1, using a 128/256-bit key derived through PBKDF2 as described in RFC 8018 section 5.2.

- As per RFC 8018, a password is considered to be an octet string of arbitrary length whose interpretation as a text string is unspecified. In the interest of interoperability, however, it is recommended that applications follow some common text encoding rules. ASCII and UTF-8 are two possibilities. Note that octet strings are not required to have null terminators. If such terminators are desired, implementations shall explicitly include them by mutual agreement.
- The PBKDF2 function shall default to SHA-256 as the hashing algorithm, with 1,024 iterations.
- PSK implementations may offer other hashing algorithms as per RFC 8018, and other values of the number of iterations. If such options are offered, they shall be enabled by explicit out-of-band configuration for all participants. Note that the SHA-1 hashing function is insecure and should be avoided.
- The key and IV used for both encryption and decryption are described in section 7.1 of this document. The PBKDF2 function shall use the nonce, transmitted by the sender in the GRE Key field, as salt to generate the actual key.

Note: Annex B presents a numerical example of the key generation algorithm described in this section.

Note: The resulting generated key is valid for up to 2^{32} packets. It is therefore safe to use the full 32-bit GRE sequence number as IV for the AES operations on single packets. Since the AES key changes continuously, there is no risk of reusing the same IV within the encrypted flow.

Note: This algorithm does not provide origin authentication. Therefore, it is susceptible to replay and data injection attacks. However, this risk is mitigated because duplicate and out of order packets are handled properly by the GRE receiver and/or by the RIST protocol, without adverse effects to the resulting decrypted output stream.

7.4 On-The-Fly Passphrase Change

On-The-Fly Passphrase Change capability is optional.

In some one-to-many situations, it may become necessary to de-authorize a subset of the receivers. This section describes an optional mechanism to change the passphrase on-the-fly with no service interruption for the receivers which remain authorized. The process is as follows:

- A new passphrase is distributed through out-of-band means to the receivers that are to remain authorized.
- This passphrase is loaded in all the relevant receivers, but remains inactive.

BY-ND

- Once all the relevant receivers are configured with the new passphrase, the sender switches to a key generated by this new passphrase. This change is signaled in the GRE packets.

If On-The-Fly Passphrase Change capability is implemented, the GRE header shall contain one bit, denoted by **B**, which identifies the passphrase to be used. The passphrase selected by **B**=0 shall be denoted as the "even passphrase", and the passphrase selected by **B**=1 shall be denoted as the "odd passphrase". Bit **B** shall be the MSB of the Nonce field, as indicated in Figure 11.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| |1|1|Reserved0|H| RV  | Ver |         Protocol Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|B|                           Key/Nonce                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 11: GRE Header with Odd/Even Bit B in the Nonce

Operation shall be as follows:

- All receivers shall use the full 32-bit Nonce field as the Nonce value for key generation.
- Receivers implementing support for odd/even passphrases shall initialize both passphrases to the same value. This ensures compatibility with senders that do not support different odd/even passphrases.
- Senders implementing support for odd/even passphrases shall initialize both passphrases to the same value, and may use any value for the Nonce.
- Passphrase changes shall occur as follows:
    - The sender and the receivers who should remain authorized are configured with the new passphrase.
    - The new passphrase shall be associated with the value of **B** that is not currently in use. For example, if at configuration time, **B**=1, then the new passphrase will be associated with **B**=0.
    - The sender switches to the new passphrase by manual user intervention or other out-of-band means. At this time, the sender shall generate a new Nonce with the inverted value of **B**. In the example above, the new Nonce will be set to **B**=0.
    - The Nonce change will trigger a key recalculation at the receivers. Receivers supporting odd/even passphrases shall use the new passphrase.
    - From that point on, if the sender decides to rotate the key, the new Nonce values shall have the same value of **B**.
    - This process can be repeated at a later point in time if a new passphrase change is required.

BY-ND

7.5 PSK Authentication using EAP SRP

TR-06-2:2020 defined a PSK authentication method based on TLS-SRP. This method was deprecated in TR-06-2:2021, and implementation of this method is not recommended. Starting with TR-06-2:2021, a new optional PSK authentication method is defined. Devices may use the RV field in the header to determine which method is in use.

The PSK authentication method shall follow the protocol described in Annex D , using EAPoL messages encapsulated in GRE using EtherType 0x888E as shown in Figure 12.

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |             GRE Header with Protocol Type=0x888E              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 :         PSK Authentication Message Defined in Annex D         :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12: Encapsulation of PSK Authentication Messages

The EAP handshake data packets transmitted over the GRE tunnel shall not be encrypted with PSK encryption.

Once the peer has been authenticated, this authentication state should be cached for the duration of the session. The details of this process are left to the discretion of the implementer.

7.6 PSK Future Nonce Announcement Message

The PSK Future Nonce Announcement Message may be used by a sender to inform the receiver of the next Nonce value the sender intends to use. If a sender makes use of the PSK Future Nonce Announcement Message, the sender shall employ the message as follows:

- Senders shall transmit the PSK Future Nonce Announcement Message prior to a change in Nonce. The period between the transmission of this message and the Nonce change shall be no less than one second.
- Senders may transmit this message multiple times to guarantee delivery.
- After sending a message with a given Nonce value, senders shall not send a Future Nonce Announcement Message with a new Nonce value until the announced Nonce is in use for encryption.

The format for the PSK Future Nonce Announcement Message is shown in Figure 13.

BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|  |0|0|Reserved0|H| RV  | Ver |      Protocol Type=0xCCE0     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   VSF Protocol Type=0x0000    |  VSF Protocol Subtype=0x8001  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Future PSK Nonce                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13: Future PSK Nonce Message

Senders shall set the **Future PSK Nonce** field in Figure 13 to the next Nonce to be used in PSK operation.

NOTE: In PSK mode, the receiving device needs to re-compute the AES decryption key from the passphrase when the Nonce changes. This operation is CPU-intensive, and it can take a significant amount of time to complete. This can become an issue in high bit rate situations, since packets with the new Nonce will have to wait for the new key to be computed, causing a delay. In such cases, senders can use the PSK Future Nonce Announcement Message to inform the receiving node of the next Nonce value, so it can pre-compute and cache the AES key.

8 NULL Packet Deletion and High Bitrate Operation

This section describes an optional RTP header extension to support NULL Packet Deletion and High Bitrate operation.

8.1 NULL Packet Deletion (Informative)

One of the most common applications of RIST is to transmit MPEG Transport Streams. Typical MPEG Transport Streams contain 3 to 5% NULL packets. These packets convey no information. However, such packets are included for stream timing purposes and cannot simply be discarded.

The bandwidth used by NULL packets can be saved by transmitting some sort of marker instead of the packet. In this case, the receiving device can re-insert the NULL packets in the same position, thus keeping the stream timing intact.

RIST achieves this function by using a bit field on an extension header to indicate the location of the NULL packets. A typical RTP packet will have up to seven transport packets. If, for example, three of these seven packets are NULL packets in positions 1, 2 and 6 in the payload, the RIST sender will transmit a shorter RTP packet with just four transport packets, and a bitmask with the value 1100010, indicating where NULL packets will need to be inserted in this group of transport packets. The receiver will infer that the original RTP payload had seven transport packets (since it received four transport packets plus three flags for the deleted NULL packets), and the locations of the NULL packets themselves.

BY-ND

8.2 High Bitrate and/or High Latency Operation (Informative)

The RTP sequence number is only 16 bits, which means it wraps around every 65,536 packets. If the RIST link is carrying a 100 Mb/s transport stream, with the usual seven transport packets per RTP payload, the RTP sequence number will wrap around every 6.9 seconds. When using ARQ and allowing for the recommended 7 retries, this means that the maximum supportable round-trip delay is around one second. This is a significant limitation, which gets even worse as the bit rates go up. Therefore, the sequence number must be extended to support this operation, ideally to 32 bits. The method for extending the sequence number is described in section 8.3 below.

8.3 RTP Header Extension to Support NULL Packet Deletion and Extended Sequence Numbers

Both NULL Packet Deletion and RTP Sequence Number Extension shall be accomplished using an RTP Header Extension, as per RFC 3550 section 5.3.1. For convenience, the generic RFC 3550 RTP Header Extension is show in Figure 14 below.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      defined by profile       |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        header extension                       |
|                             ....                              |
```

Figure 14: Generic RTP Header Extension (from RFC 3550)

The RTP Header Extension for RIST Main Profile shall be implemented as shown in Figure 15. Fields shall be in network byte order, MSB first.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    0x52 (R)   |    0x49 (I)   |          Length=1             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|E|Size |0 0 0|T| NPD bits     |   Sequence Number Extension   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 15: RTP Header Extension for RIST Main Profile

The bits in the RTP Header Extension for RIST Main Profile shall be implemented as detailed below:

- **Header Extension Identifier:** This is the 16-bit field denoted by "defined by profile" in Figure 14. For RIST Main Profile, this field shall have the value 0x5249, corresponding to the ASCII codes for "RI".

BY-ND

- **Length:** as required by RFC 3550, this is the length of the header extension in 32-bit words, excluding the four-octet extension header. For RIST Main Profile, Length shall be set to 1.
- **N:** Shall be set to 1 if Null Packet Deletion is in use. If N=1, the following bits are valid:
  - **Size:** This is an optional 3-bit field that indicates how many transport packets were in the original RTP packet. If used, senders shall set this field to the number of Transport Packets in the original RTP packet. Senders shall set this field to 000 to indicate that the payload size is to be derived from the NPD bits and the size of the received payload.
  - **T:** Transport packet size flag. Senders shall set this field to 0 to indicate 188-byte packets, or shall set this field to 1 to indicate 204-byte packets.
  - **NPD bits:** Each bit, when set, indicates that a NULL packet has been removed from the corresponding position. In this field, MSB corresponds to the first transport packet in the payload. If the original payload has less than 7 transport packets, the trailing bits that do not correspond to actual packets shall be set to zero.

  If the **N** bit is set to 0, the content of the **Size**, **T**, and **NPD** bits is undefined and shall be ignored by the receiver.
- **E:** Set to 1 if Sequence Number Extension is in use. If E=1, the following field is valid:
  - **Sequence Number Extension:** this is a 16-bit RTP sequence number extension, in network byte order (big-endian). A 32-bit sequence number is created by using the 16-bit RTP sequence number as the LSB and this field as the MSB.

  If the **E** bit is set to 0, the content of the **Sequence Number Extension** field is undefined and shall be ignored by the receiver.

A sender that only implements NULL packet deletion may omit the RTP extension header for RTP payloads not containing NULL packets. More specifically, a header where **E**=0, **N**=1, and **NPD**=0 may be omitted.

8.4 NACK Packet Support for Extended Sequence Numbers

The NACK packets defined in section 5.3 of VSF TR-06-1, RIST Simple Profile use 16-bit sequence numbers. An extension to NACK packets is defined in this section to support Extended Sequence Numbers.

This document defines a new RTCP packet. This new packet, denoted as EXTSEQ, conveys the higher 16 bits of the sequence number for the following NACK packet.

When EXTSEQ packets are in use, the RTCP compound packet stack shall be as follows: RR, CNAME, EXTSEQ and NACK. The full 32-bit sequence number for each entry in the NACK packet shall be built by pre-pending the 16 bits carried in the EXTSEQ packet with the 16-bit sequence number in the NACK packet. For Bitmask-based retransmission requests, the 16-bit

BY-ND

sequence number in the NACK packet is carried in the Packet ID (PID) field of the FCI. For Range-based retransmission requests, the 16-bit sequence number in the NACK packet is carried in the Missing Packet Sequence Start field.

If the RIST receiver needs to send NACKs for packets that have different high-order extension values, these shall be sent as different NACK request packets and shall be separated with a new EXTSEQ packet. The compound RTCP packet will then become RR, CNAME, EXTSEQ, NACK, EXTSEQ, NACK. The RIST receiver may also break this message into two RTCP compound packets, one for each value of EXTSEQ.

The EXTSEQ RTCP packet shall be implemented as an Application-Defined packet, using “RIST” as the name and a subtype of “1”, as indicated in Figure 16. Fields are in network byte order, MSB first.

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |V=2|0|Subtype=1|   PT=APP=204  |             length=3          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                  SSRC of media source                         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    0x52  (R)   |    0x49  (I)   |    0x53  (S)   |    0x54  (T)   |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Sequence Number Extension   |   reserved=0                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 16: RIST EXTSEQ RTCP Packet

Where:

SSRC of media source: 32 bits

The synchronization source identifier of the media source that this feedback request is related to. As indicated in VSF TR-06-01, the LSB of the SSRC is used to differentiate between original packets and retransmitted packets. The RIST receiver may use either value in the request packet.

Sequence Number Extension: 16 bits

MSB for all the NACK starting sequence numbers that follow this RTCP packet, in network byte order (big endian).

8.5 NULL Packet Deletion Example (Informative)

The NULL Packet Deletion technique described in this document supports RTP payloads of less than seven transport packets. The actual number of transport packets in the RTP payload can be determined by adding the number of packets received in the payload to the number of bits set in the NPD field. The suggested processing of the NPD field is as follows:

CC BY-ND

- Process the NPD field from MSB to LSB
- For each bit in the NPD field:
    - If the bit is 1, output a NULL packet
    - If the bit is 0, output the next transport packet from the payload
- Stop when either one of the following conditions are true:
    - All the seven bits in the NPD field have been processed
      or
    - The current bit in the NPD field is 0 but there are no more transport packets in the payload

Note that it is possible to construct packets that have an invalid combination of NPD bits and payload packets. Such invalid packets can be classified into two types:

1. Packets whereby the sum of the NPD bits set to one plus the received payload transport packets is more than seven (i.e., too many packets).
   or
2. Packets whereby there is at least one NPD bit set to one, and the number of NPD zero bits in more significant positions than the lowest significant NPD bit set to one is more than the number of received payload packets (i.e., not enough packets). For example, if the lowest significant NPD bit set to one is bit 4, this number will be the number of NPD bits set to zero in bit positions 5 and higher.

The receiver behavior in such error situations is left to the discretion of the implementer. In such cases, receivers should give priority to transmitting the actual payload transport packets. The receiver may use the **Size** field in the header if coded as an additional indicator. If there is a mismatch between the **Size** field and the payload size computed from the NPD bits plus the number of received transport packets, it is recommended to use the latter.

It is also possible to have a mismatch between the **T** bit (which selects 188/204 packet size) and the payload. In such cases, it is recommended that the payload size take precedence. The **T** bit is only essential for payloads composed exclusively of NULL packets.

The following is an example of processing a set of variable-size RTP payloads. Assume that the following set of packets are to be transported:

- An RTP datagram with 7 TS packets where packets 1, 2 and 7 are NULLs.
- An RTP datagram with 3 TS packets where packets 1 and 3 are NULLs.
- An RTP datagram with 5 TS packets where all the packets are NULLs.
- An RTP datagram with 4 TS packets where packets 3 and 4 are NULLs.

Using NULL Packet Deletion, the corresponding RTP packets are:

- An RTP datagram with NPD=1100001 and 4 TS packets in the payload

BY-ND

- An RTP datagram with NPD=1010000 and 1 TS packet in the payload
- An RTP datagram with NPD=1111100 and no (zero) TS packets in the payload
- An RTP datagram with NPD=0011000 and 2 TS packets in the payload

The receiving device processes these RTP datagrams as follows:

- NPD=1100001 shows 3 NULL packets, and there are 4 packets in the payload. Therefore, this will be a 7-TS datagram. The locations of the NULLs are positions 1, 2 and 7.
- NPD=1010000 shows 2 NULL packets, and there is one packet in the payload. Therefore, this will be a 3-TS datagram. Based on this determination, only the first 3 bits of the NPD are processed, and the resulting RTP datagram will contain 3 TS packets with NULLs in positions 1 and 3, and the payload packet in position 2.
- NPD=1111100 shows 5 NULL packets, and there are none in the payload. Therefore, this will be a 5-TS datagram. Based on this, the resulting RTP datagram will have 5 NULL packets.
- NPD=0011000 shows 2 NULL packets, and there are two more in the payload. Therefore, this will be a 4-TS datagram. NPD indicates that the two payload packets are transmitted first, followed by two NULL packets.

8.6 Combining NULL Packet Deletion and Sequence Number Extension with SMPTE-2022-1 FEC

The extensions described in this section may be combined with SMPTE-2022-1 FEC, as described below. In all cases, the RTP stream containing the media shall be transmitted unmodified in accordance with the previous sections of this document.

Note: The use of an extension header is not compatible with SMPTE-2022-2 operation.

8.6.1 Sequence Number Extension

If Sequence Number Extension is in use, the **SNBase ext bits** field in the FEC header described in section 8.4 of SMPTE-2022-1 shall be set to the lower 8 bits of the Sequence Number Extension, as indicated in Figure 17.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       SNBase low bits         |        Length Recovery        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|E| PT recovery |                    Mask                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          TS recovery                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|D|type |index|     Offset    |       NA      |SNBase ext bits|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 17: FEC Header (from SMPTE-2022-1)

CC BY-ND

8.6.2 NULL Packet Deletion

If NULL Packet Deletion is in use, the order of operations at the sender shall be as follows:

- The **data_bytes** (content) of any NULL packets in the payload shall be replaced by 0xFF for the purposes of FEC computation.
- The **continuity_counter**, **transport_error_indicator** and **transport_priority** fields of any NULL packets in the payload shall be replaced by zero for the purposes of FEC computation.
- FEC packets shall be computed before NULL Packet Deletion, using the modified NULL packets as above.
- The FEC XOR operation shall only include the payload and shall not include the extension header described in this document.
- NULL Packet Deletion shall be performed after the FEC computation and before the packets are transmitted.

At the receiving side, NULL packets shall be re-inserted back into the RTP payload, with **data_bytes** (content) set to 0xFF, **continuity_counter**, **transport_error_indicator** and **transport_priority** set to zero, before the FEC computation happens.

9 Compatibility between RIST Main Profile and Simple Profile Devices

RIST Main Profile adds several features to Simple Profile, but the underlying transport mechanism is still Simple Profile. Therefore, RIST Main Profile devices shall support operation in Simple Profile mode.

In RIST Simple Profile, the sender is always the client, and the receiver is always the server. Stream transmission is always initiated by the sender. If the sender is a RIST Main Profile device, the selection between RIST Main Profile and RIST Simple Profile shall be manually configured.

Note: For a RIST Main Profile receiver operating as a server, the following profile mismatch situations can happen:

- **Case 1:** the receiver is configured for Simple Profile (and thus listening on ports P and P+1) and receives a Main Profile keep-alive message (or a DTLS connection) on either P or P+1.
- **Case 2:** the receiver is configured for Main Profile (and thus listening on a single port P) and receives either RTP or RTCP packets on that port.

For either case, the receiver can discard the packets and it is recommended that it provide the user with some indication of this fact. Alternatively, receivers can operate as follows:

CC BY-ND

- For **Case 1**, the receiver can distinguish between the Simple Profile packets and the Main Profile packets, so it could automatically go into Main Profile mode and complete the negotiation.
- For **Case 2**, the receiver can try to operate in Simple Profile mode. This might or might not be possible depending on the configuration of the firewalls upstream of the receiver.

10 Compatibility with RIST Main Profile Devices Implementing TR-06-2:2021 and Earlier Versions (Informative)

This Specification defines new GRE encapsulation methods using the VSF EtherType in the Protocol Type field. Devices compliant with the 2021 and earlier versions of this TR are unaware of this EtherType and are likely to discard these packets. Moreover, the behavior of such implementations when receiving packets with unknown RV types is undefined.

On reception, a device implementing this Specification is able identify packets compliant with the 2021 and earlier versions and seamlessly process them, if desired.

On transmission, the algorithm described below is suggested for transparent communication. In the description, the term “new device” refers to a device compliant with this version of the Specification, and “legacy EtherTypes” refer to the EtherTypes used in the 2021 and earlier versions.

1. The new device starts the communication by sending traffic with RV=000 or RV=001 and the corresponding legacy EtherTypes. The new device will also send copies of whatever message it is using for the Keep-Alive function with RV=010 and the corresponding format.
2. If the new device receives any traffic with RV=010 and/or the VSF EtherType, it switches its transmission to the VSF EtherType and RV=010 and stops sending traffic with RV=000 or RV=001.
3. If the new device does not receive any traffic with RV=010 and/or the VSF EtherType, it stops sending the new version Keep-Alive packets after a timeout.

Implementers are advised to carefully consider the amount of network traffic their devices generate when using this method, so as not to overload the network. If a device is relying on a data stream for the Keep-Alive function, it is not advisable to replicate that stream.

This algorithm is only applicable to point-to-point communication. If a sender is transmitting a multicast to a set of receivers (which can change dynamically over time), the only reasonable option is manual configuration. Implementers are advised to always offer a manual configuration option.

BY-ND

Annex A Certificate Management (Informative)

This Annex describes several options for managing client and server certificates for devices using RIST Main Profile with DTLS encryption.

A.1 Certificate Processing

DTLS includes the option of supporting both server and client certificates for authentication. Processing of a certificate includes the following operations on the device that is checking the certificate:

- Has the remote site presented a certificate?
- If a certificate is presented, is this certificate signed by a Certificate Authority (CA) or CA chain that is trusted by the device checking it?
- If the certificate is signed by a CA trusted by the device, is the remote side allowed to connect to the device checking the certificate?

The processing of certificates is independent of which device is the server and which device is the client. Each device will do its own processing and decide, independently, whether it is willing to continue with the connection.

The final responsibility for defining certificate processing policies rests with the end-user of the RIST system. Implementers are encouraged to provide as much flexibility as possible, to not limit the options available to end users.

A.2 Discussion of Certificate Authorities (CA)

A certificate is normally signed by a Certificate Authority (CA). There are several public, trusted CAs available on the Internet. Web sites that need to guarantee their identity, such as those from banks, use certificates signed by a public CA. On the other hand, anybody can create a CA and use that to sign certificates. One typical example of using a "private" CA is VPN servers – they typically act as their own CA and will only accept certificates signed by their private CA.

When a device is checking a certificate, it needs to decide whether it trusts the CA who signed it. Implementers are encouraged to provide the following options to their users:

- Accept certificates signed by one of the known public CAs. This may be applicable for servers with fixed IP addresses or host names.
- If the device operates as a server, provide the option for it to become its own CA and sign all the certificates, as it is typically done with VPN servers.
- Provide the option for the user to employ an external CA independent of the device. More specifically, the user should be able to configure the device to use an arbitrary CA.

In the remainder of this section, it is assumed that one of the previous options for CA selection is in use, and, during the certificate exchange, the CA is deemed acceptable by the device checking the certificate.

BY-ND

A.3 Remote Certificate Processing at the Local RIST Device

Implementers are encouraged to provide the following options to end-users regarding the decision to accept or reject certificates in RIST devices:

- Accept all certificates, regardless of CA (for testing). If the device is a server, this means that any device can connect to it; if the device is a client, it means that connections with any server will succeed. It is strongly suggested that this option be disabled by default, and that the device be in alarm mode for as long as this option is enabled.
- Accept all certificates that have been signed by one or more configured CAs. The user may have their own CA (either integrated with the server or separate from it), and only certificates signed by this private CA will be accepted. Alternatively, the device may be configured to accept certificates signed by public CAs. This provides a very basic level of authentication without too much configuration burden. Implementers should consider making this option the default to ease initial setup.
    - If this option is provided, implementers are encouraged to provide the means to create a blocked list in the device – in other words, "accept all certificates signed by this CA except the ones in this list". This addresses the case where a remote device is no longer trusted.
- Provide a list of acceptable certificates, and only devices presenting certificates in that list are allowed to connect. Connections from remote devices presenting certificates signed by the acceptable CA but not this list will be rejected. If the local device is a server, this will be a list of allowed clients; if the local device is a client, this will be a list of the servers it is allowed to connect to. If a remote device is no longer trusted, its certificate can be removed from the list.

Note that certificates may be encrypted. Support for encrypted certificates is optional.

A.4 Notes on Blocked Lists and Accepted Lists

To implement either an accepted list or a blocked list, the device needs to use one of the certificate fields as the identifying entry for deciding whether the certificate is in the list (either blocked or accepted). It is suggested that the Common Name (CN) field be used.

A.5 Certificate Signing Requests

As indicated before, a device may need to present a valid certificate to the remote endpoint for the connection to succeed. This certificate may be obtained in the following two ways:

1. The remote endpoint (typically a server) generates a full set of credentials for the device, which includes both the certificate and the private key. This set of credentials, possibly protected by a passphrase to keep the key secret, is then provided to the device.
   or
2. The device generates a private key by itself. This key will never leave the device for security reasons. After generating the key, the device generates a Certificate Signing Request that can be sent as a file to the remote endpoint for signing by whatever CA is

BY-ND

acceptable to it. The remote endpoint then generates a certificate chain that that is returned to the device. This certificate chain does not include the private key and does not need to be kept secret.

Implementers are encouraged to provide enough functionality to support both cases described above.

A.6 Certificate and Key File Formats

To foster interoperability, it is important that the devices from different implementers agree on file formats. Devices may need to be configured with a list of one or more acceptable CAs and may need to be configured with the certificates they need to use. These certificates may be coming from equipment provided by a different implementer. Therefore, a common file format needs to be agreed between these devices.

It is suggested that, as a baseline, all equipment should support the **PEM** format, as defined in RFC 7468. A PEM file is a text file, with the binary data encoded in Base64, and can include one or more certificates and may also contain keys. It is also suggested that a single file be provided with all the information required – i.e., the CA chain, the certificate, and the private key if needed. When using the PEM format, these can be simply concatenated. The device should not assume any order for these items.

An example of a combined file with a certificate and keys is:

```
-----BEGIN CERTIFICATE-----
MIIDfjCCAuegAwIBAgIJALYx7VgDX7U1MA0GCSqGSIb3DQEBBQUAMIGHMQswCQYD
VQQGEwJVUzELMAkGA1UECBMCQ0ExEDAOBgNVBAcTB1Nhbkpvc2UxFzAVBgNVBAoT
… (certificate continues)
gQBAYeJpSnoKWk3c5Uy0PZl+/8ee9AZ/swYiES2+ehy/d4EGofuH4K+SFIx9fpH1
zg507vBRNwiAegiawxkpMhsptV3Hv5rkFy0/nkg/uYKewOu6O0k1XEM7LbRiOumf
cGA5sNColbALctgBd49Alf19sQPsvXhjjAuFqoPGNOF/Wg==
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDgtzWVqihnzLhUTtAyGvab57IzqHu0R4j9C7QOArl/dqgrgBA9
wtgRp3zwU9UHgrmzK++6NByA+VxsnGtVpl9RsiiUk0T+8uI4UcapeUE3AThQ29WE
… (key continues)
BUsETpWaKtebcnUuIaMCQQDUCipcuTEu9ITBf1uK9BNB2KQ1weF4q4pT2IjdFBEA
g1FDrlIqP72OQodz54Xw+aWH314pMofAKcaIp1HCL69i
-----END RSA PRIVATE KEY-----
```

As indicated in the example, each element starts with five dashes and the word BEGIN, followed by the type of the element, followed by another five dashes. The element ends the same way:

BY-ND

five dashes, followed by the word END, followed by the same element type, followed by another five dashes.

cc BY-ND

Annex B PSK Key Generation Example (Informative)

This Annex provides one example of 128-bit and 256-bit AES keys generated from a known passphrase and nonce. It is provided to allow implementations to be checked against known values.

The inputs are:

- Passphrase: **`Reliable Internet Stream Transport`**
- Nonce: **`0x52495354`**

Figure 18 shows the packet received from the network with the above nonce.

```
    0                   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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0| |1|1|Reserved0|H| RV  | Ver |         Protocol Type         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     0x52      |     0x49      |     0x53      |     0x54      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Sequence Number                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18: Sample Received GRE Packet

Using the PBKDF2 hashing algorithm specified in section 7.3, the following keys are derived from this input:

- 128 bit key: `1c2b0cfc90ae2638fea78c7fb2977047`
- 256 bit key: `1c2b0cfc90ae2638fea78c7fb297704718bff7f4052743001a9b7ebb51cc9f1c`

The following Python 3 code can be used to generate the keys:

```python
import hashlib

key = hashlib.pbkdf2_hmac("sha256", b'Reliable Internet Stream Transport', bytes.fromhex('52495354'), 1024, 16)
print("Derived 128 bit key:", key.hex())

key = hashlib.pbkdf2_hmac("sha256", b'Reliable Internet Stream Transport', bytes.fromhex('52495354'), 1024, 32)
print("Derived 256 bit key:", key.hex())
```

BY-ND

Annex C Supporting Multiple Clients Using the Same Server UDP Port (Informative)

A RIST Main Profile server may offer the option of supporting multiple clients connected to the same UDP port. From a traffic standpoint, the server can differentiate the packets from the various clients by using the combination of the source IP address and source UDP port in the packet. Simple application examples for this use case are:

- Multiple reporters in the field, each equipped with an encoder, sending live video content back to the newsroom. Having a single UDP port open to receive all communication at the newsroom simplifies the setup of the encoders.
- Multiple decoders connecting to an encoder to receive live video.

There are situations where it becomes necessary to uniquely identify the client in order to either send specific content to it, or to direct the content coming from it to a specific receiver. One example would be a situation where there are multiple feeds available at one site, and when a client connects, the "correct" feed needs to be sent to it.

The following RIST Main Profile mechanisms are available to uniquely identify the clients:

1. If DTLS is used with certificates, the various clients may be identified by the certificates they present to the server (see the discussion in Annex A). It is suggested that the Common Name (CN) be used as the identifier. In this case, the system configuration must ensure that each client has a different CN.
2. If TLS-SRP (see Section 6.4) or EAP SRP (see Section 7.5), each client will have a different username and password, and this combination may be used to differentiate between clients.
3. If the keep-alive messages defined in Section 5.6.3 are used, the 48-bit MAC address included in the message may be used to differentiate between clients. Implementations must ensure that the MAC addresses in the keep-alive messages are unique.
4. Clients may be differentiated using the inner (tunnel) IP address. In this mode, addresses are assigned a-priori by static configuration and are known to the server and clients. The server can detect the client inner (tunnel) IP address using one of the following mechanisms:
    a. If the keep-alive messages defined in Section 5.6.3 are used, and the message has a JSON payload as described in Section 5.6.4, the server can inspect the `tunnelIP` element to find the client's tunnel IP address.
    b. If the server and clients are using the Full Datagram Mode defined in Section 5.3.1, the inner (tunnel) IP address can be read from the incoming encapsulated packets. Note that, in this case, the client cannot be identified until it sends its first encapsulated IP packet.

CC BY-ND

5. If the server and clients are using the Reduced Overhead Mode defined in Section 5.3.2, the source UDP port in the reduced UDP header shown in Figure 5 can be used to differentiate clients. In this mode, ports are assigned a-priori by static configuration and are known to the server and clients. Note that, in this case, the client cannot be identified until it sends its first encapsulated IP packet.

If possible, the preferred way to identify the clients should be options 1 or 2 above. Options 3, 4, and 5 do not require the use of DTLS and are available to clients without encryption support. Option 5 should only be used as the last possible resort if no other options are available.

BY-ND

Annex D Specification of the EAP SHA256-SRP6a Authentication Protocol (Normative)

D.1 General Overview

RIST Main Profile devices using PSK encryption have an intrinsic authentication mechanism, namely the knowledge of the Pre-Shared Key. More specifically, if a device has been configured with the correct key and can decrypt the content, this device may be considered authenticated. This level of authentication may be sufficient for some applications.

The method described in this Annex is available for applications where one additional level of authentication is required. One example would be a situation where a server has a set of clients, all configured with the same Pre-Shared Key, but with different levels of access to content present in the server.

For each authentication instance, the model is as follows:

- There is one server and one or more clients.
- Clients shall only communicate with the server. All packets sent from the client to the server shall be unicast.
- Clients initiate the communication with the server by following the steps in section 5.6, or by sending the optional EAPOL-Start packet described below.
- Upon tunnel establishment, the server shall initiate the authentication protocol described in this Annex.
- The server and client shall not send any data packets and shall discard any received data packets until the authentication completes successfully. Data packets are any GRE packets where the Protocol Type field is not 0x888E (see Figure 12).

The authentication packets described in this Annex shall be transmitted encapsulated in GRE with Protocol Type 0x888E, as per Figure 12. The EAPOL packets described in this Annex represent the payload of such GRE packets.

D.2 Authentication Algorithm and Protocol

Fundamentally, the authentication algorithm is based on a username/password pair. The server has a list of all the usernames and passwords (in a secure format); an authorized client knows its username and password as well. The authentication algorithm provides a secure way for the client to prove to the server that that it knows a valid username/password pair, in a way that a third-party monitoring the transactions cannot use the information exchanged to later successfully login to the server. It also provides a secure way for the client to authenticate the server. Finally, the algorithm also generates a common strong ephemeral shared key that may be used to encrypt future unicast communication between the authenticated client and the server.

BY-ND

The algorithm is based on two fundamental values, the generator value “g” and the prime modulus “N”. The server may use different values of “g” and “N” per client, since these values are communicated to the client at the beginning of the negotiation. The following rules shall be observed:

1. As indicated in section D.3.4.3, if “g” is not indicated, the client shall assume the default value of 2.
2. “N” shall be a large safe prime, of at least 512 bits. If “N” is not indicated, the client shall assume the default 2048-bit value indicated in section D.3.4.3.
3. The value “g” shall not be higher than “N”.

In the algorithm description below, the following operations are used:

% Modulo operator (as in the C programming language); the remainder of a division. The notation x % y shall indicate the remainder of the division of “x” by “y”.

^ Modulo “N” exponentiation. The notation x ^ y shall indicate the remainder of “x” to the power “y” when divided by the prime modulus “N”. Example:
x = 2
y = 10
N = 263
x^y = (2^10) modulo 263 = 1024 modulo 263 = 235 (remainder of 1024/263)

| String concatenation. This operator creates a string that is a concatenation of its two arguments, in the same manner as the “strcat” function in a standard C library.

The algorithm shall use the SHA256 hashing algorithm. If multiple arguments are provided to the SHA256 function, this indicates that the arguments shall be concatenated, and the SHA256 function applied to the combined value. For example, SHA256(x, y) means “create a buffer with the contents of “x”, followed by the contents of “y”, and apply the SHA256 algorithm to the resulting buffer”. When the SHA256 hash is applied to a string, the null terminator (if present) shall not be included in the hash computation.

For each client, the server shall select a random salt “s”, containing at least four octets. If the client password is denoted by “P” and the client username by “I”, the server shall compute the value “x” defined by:

x = SHA256(s, SHA256(I | “:” | P))

For each client, the server shall compute the password verifier “v” as follows:

v = g^x

The server shall store the values “s” and “v” for each client, indexed by the client username “I”. The server should not store the cleartext password “P”.

BY-ND

The client starts the authentication process by contacting the server using the EAPOL-Start message of section D.3.1 (or any empty message, such as a Keep-Alive). The server requests the client username "I" using the Identity Request message of section D.3.3.1, and the client returns this information using the Identity Response message of section D.3.3.1.

If the username "I" is known to the server, the server sends the corresponding "s", "g" and "N" values to the client using the Challenge Request Packet of section D.3.4.3. If the client's username is not known to the server, it may abort at this point or continue with fake values for "s", "g" and "N". This is described in detail in Section D.4.

Upon reception of the Challenge Request Packet, the client caches the "s", "g", and "N" values, and generates a random number "a" between 1 and N-1. It then computes the value "A" as follows:

$$A = g\hat{}a$$

The value "A" is returned to the server in the Client Key Response Packet of section D.3.4.4. The server caches this value for later use. Upon reception of the Client Key Response, the server generates a random number "b" between 1 and N-1, and computes the value "B" as follows:

$$k = SHA256(N, g)$$
$$B = (kv + g\hat{}b) \% N$$

The value "B" is returned to the client in the Server Key Request Packet of section D.3.4.5. Upon receiving the value "B", the client performs the following computations:

$$u = SHA256(A, B)$$
$$x = SHA256(s, SHA256(I, \text{“:”}, P))$$
$$k = SHA256(N, g)$$
$$S = ((B - kg\hat{}x) \hat{} (a + ux)) \% N$$
$$K = SHA256(S)$$
$$M1 = SHA256(SHA256(N) \text{ xor } SHA256(g), SHA256(I), s, A, B, K)$$

The value "M1" above is returned to the server in the Client Validator Response Packet from section D.3.4.6, and the client caches the session key "K".

Upon receiving the "M1" value, the server performs the following computations:

$$u = SHA256(A, B)$$
$$S = ((Av\hat{}u) \hat{} b) \% N$$
$$K = SHA256(S)$$
$$M1 = SHA256(SHA256(N) \text{ xor } SHA256(g), SHA256(I, s, A, B, K))$$

If the local "M1" calculation yields the same value as the received "M1" value, the server shall consider the client as authenticated. It computes the value "M2" as follows:

cc BY-ND

$$M2 = SHA256(A, M1, K)$$

The “M2” value is returned to the client using the Server Validator Request Packet from section D.3.4.7. Upon receiving this packet, the client performs the same computation, and if the local calculation yields the same “M2” value as the received packet, the client shall consider the server authenticated.

D.3 Packet Formats

This section describes the various packet formats used in the authentication process. The packet formats follow a hierarchical structure based on GRE-encapsulated EAPOL packets. The hierarchy is depicted in Figure 19. The highlighted items in Figure 19 represent the actual packets transmitted in the network at the various phases of the protocol and are documented in this section. All protocol transactions shall use unicast addressing between server and client.

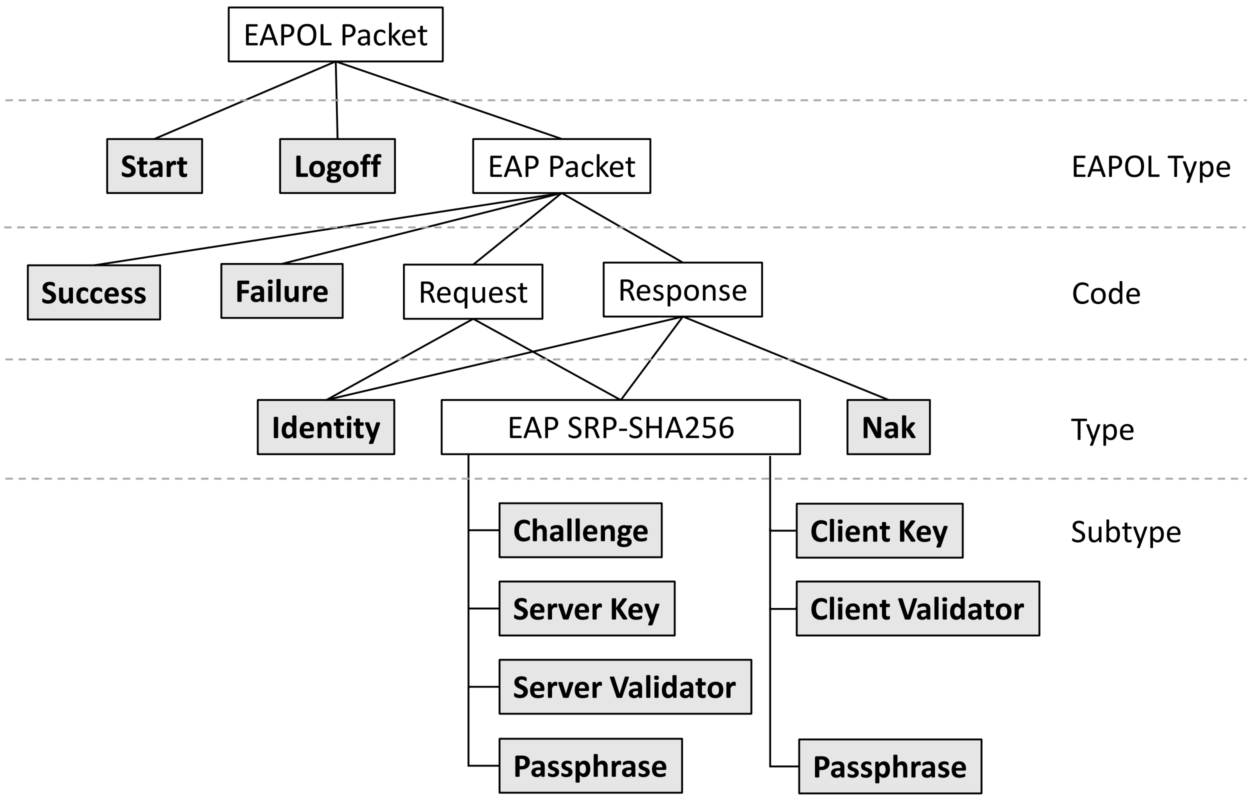

Figure 19: Authentication Packet Format Hierarchy

D.3.1 EAP Encapsulation

Figure 20 shows the authentication packet format.

BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 |  EAPOL Type   |        Payload Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                 Packet Payload (EAPOL Type=0)                 :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 20: Authentication Packet Format

The sender shall set the fields in Figure 20 as follows:

- **EAP Version (8 bits):** Senders shall set this field to "3" to indicate compliance with this Specification.
- **EAPOL Type (8 bits):** Senders shall set this field as follows:
    - 0x00: EAPOL-EAP: the packet payload field contains an EAP packet.
    - 0x01: EAPOL-Start: optional packet sent by the client to initiate the authentication process with the server. EAPOL-Start packets do not have a payload field.
    - 0x02: EAPOL-Logoff: optional packet sent before closing the connection, to revert to the non-authenticated state. This packet can be sent by either server or client. EAPOL-Logoff packets do not have a payload field.
    - 0x03-0xFF: Reserved. Receivers shall silently discard packets with these types.
- **Payload Length (16 bits):** Senders shall set this field to the length, in bytes, of the Packet Payload field. Since EAPOL-Start and EAPOL-Logoff packets have no payload, senders shall set this field to "0" zero for these two packet types.

D.3.2 EAPOL Type 0 Packet Format

Figure 21 shows the packet payload format for EAPOL Type 0.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |        Payload Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data (variable, depends on Code)  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 21: Packet Payload for EAPOL Type=0

The sender shall set the fields in Figure 21 as follows:

- **Code (8 bits):** Senders shall set this field to identify the type of EAP packet, as follows:
    - 0x01: Request
    - 0x02: Response

BY-ND

  - 0x03: Success
  - 0x04: Failure
  - All other values: reserved. Receivers shall silently discard packets with these values.
- **Identifier (8 bits):** This field is used to match responses with requests. The sender shall select an arbitrary value for the request packet. The receiver shall use the same value for the response to that request, which may be a Response packet, a Success packet, or a Failure packet. The identifier shall be incremented by one by the sender at every new request message and shall not be changed on a retransmission of a message. Receivers shall discard non-matching Response messages. As indicated in Section D.4, a full authentication exchange includes up to four distinct packets originated by the authentication server. The authentication server shall use four consecutive values for the Identifier field for each protocol exchange, and successive protocol exchanges for the same connection (identified by client IP address and source UDP port) shall use non-overlapping Identifier values. Since the Identifier field is only 8 bits, the value 0x00 is deemed to follow the value 0xFF.
- **Length (16 bits):** Senders shall set this field to the length of the EAP packet, including the Code, Identifier, Length and the variable-size data. If the overall packet is longer than what is indicated by the length field, additional octets shall be ignored by receivers. If the overall packet is truncated (i.e., not enough octets received to satisfy the length field), receivers shall discard the packet.
- **Data (variable):** The data field is zero or more octets. The size and format of the data field depends on the Code field, as described below.

D.3.3 Request And Response Packet Formats

Figure 22 shows the packet payload format for Request (Code=0x01) and Response (Code=0x02) packets.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |        Payload Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Code=1 or 2  |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      | Type-Data (variable, depends on Type) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 22: Packet Payload Format for Request/Response Packets

The sender shall set the fields in Figure 22 as follows:

- **Type (8 bits):** The Type field shall be set by the sender to indicate the format of the messages used in the Request/Response exchanges. The following values are defined in this document:
    - 0x01: Identity
    - 0x03: Nak (only valid for Response packets, shall not be sent on Request packets)
    - 0x13: EAP SRP-SHA256
    - All other values reserved.
- **Type-Data (variable):** The sender shall set this field depending on the Type value, as documented below.

D.3.3.1 Identity Request/Response Packets

Identity Request packets are sent from the server to the client. Upon reception of the Identity request packet, the client shall answer with the Identity Response packet. In the Identity Request packet, the server may include a displayable message in the Type-Data field. The client shall return its identity as a string (typically the Username) in the Type-Data field. Strings shall not be null-terminated.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Code=1 or 2  |   Identifier  |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x01   | Message for Code=1, Username for Code=2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 23: Identity Packets

D.3.3.2 Nak Response Packets

Nak Response Packets shall be sent in response to any unknown or unsupported requests. The Type-Data field shall be set by the sender to one octet with value 0x13, to indicate that only EAP SRP-SHA256 authentication is supported. The Nak response packet is shown in Figure 24.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |        Payload Length=6       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=2     |   Identifier  |            Length=6           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x03   |   Data=0x13   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 24: Nak Response Packet

D.3.4 EAP SRP-SHA256 Packet Formats

Figure 25 shows the packet formats for the EAP SRP-SHA256 packets.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Code=1 or 2  |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |    Subtype    |                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+   Subtype-Data (variable)     :
:                                                               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 25: EAP SRP-SHA256 Packet Formats

The sender shall set the fields in Figure 25 as follows:

- **Subtype (8 bits):** The subtype field shall be set by the sender to one of the following values:
    - 0x01: Challenge / Client Key
    - 0x02: Server Key / Client Validator
    - 0x03: Server Validator
    - 0x10: Passphrase Request / Response
    - All other values reserved. If a device receives an unknown subtype, it shall respond with a packet of Type Nak.
- **Subtype Data (variable):** The sender shall set this field depending on the Type and Subtype values, as documented below.

D.3.4.3 EAP SRP-SHA256 Challenge Request Packet

The EAP SRP-SHA256 Challenge is a Request packet sent from the server to the client once the username has been received from the client. It includes the unauthenticated server name for verification purposes, the password salt “s”, the generator value “g”, and the prime modulus “N”. The packet format is shown in Figure 26.

BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |        Payload Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=1     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |  Subtype=0x01 |  Name Length  |Name (variable):
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Salt Length  |    Salt (variable) ...                        :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Gen Length   |    Generator (variable) ...                   :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Prime Modulus (variable) ...                                 :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 26: EAP SRP-SHA256 Challenge Packet

The server shall set the fields in the packet in Figure 26 as follows:

- **Name Length (8 bits):** A single octet giving the length of the Name field in octets.
- **Name (variable):** The server name. This field is not authenticated. It shall not be used by the client as an authenticated peer name. The server name may be used by the client in an implementation-dependent manner.
- **Salt Length (8 bits):** A single octet giving the length of the Salt field in octets. The Salt Length shall be at least 4 octets and may be any length up to 255 octets.
- **Salt (variable):** Password salt; may contain any values. The contents of this field shall correspond to “s” in the SRP algorithm.
- **Gen Length (8 bits):** A single octet giving the length of the Generator field in octets. If this field has the value zero, the default generator value of 2 shall be used and the Generator field shall not be present.
- **Generator (variable):** The Generator value, called “g” in the SRP algorithm, is in network byte order. If the Gen Length field is zero, then the Generator field shall be omitted, and “g” shall be set to 2.
- **Prime Modulus (variable):** The Prime Modulus value, called “N” in the SRP algorithm, is in network byte order and fills the rest of the message to the length specified by the Length field in the EAP header. If the Gen Length field is zero, the Prime Modulus field may be omitted to select the default “N” value listed below. If the Prime Modulus field is present, then it should be at least 64 octets (512 bits). Longer values are recommended.

If the **Prime Modulus** field in Figure 26 is empty, the client shall assume the value below for value “N”. In this case, the **Gen Length** value shall be zero and the **Generator** value shall be assumed as “2”. The value for “N” is:

cc BY-ND

```
0xAC, 0x6B, 0xDB, 0x41, 0x32, 0x4A, 0x9A, 0x9B, 0xF1, 0x66,
0xDE, 0x5E, 0x13, 0x89, 0x58, 0x2F, 0xAF, 0x72, 0xB6, 0x65,
0x19, 0x87, 0xEE, 0x07, 0xFC, 0x31, 0x92, 0x94, 0x3D, 0xB5,
0x60, 0x50, 0xA3, 0x73, 0x29, 0xCB, 0xB4, 0xA0, 0x99, 0xED,
0x81, 0x93, 0xE0, 0x75, 0x77, 0x67, 0xA1, 0x3D, 0xD5, 0x23,
0x12, 0xAB, 0x4B, 0x03, 0x31, 0x0D, 0xCD, 0x7F, 0x48, 0xA9,
0xDA, 0x04, 0xFD, 0x50, 0xE8, 0x08, 0x39, 0x69, 0xED, 0xB7,
0x67, 0xB0, 0xCF, 0x60, 0x95, 0x17, 0x9A, 0x16, 0x3A, 0xB3,
0x66, 0x1A, 0x05, 0xFB, 0xD5, 0xFA, 0xAA, 0xE8, 0x29, 0x18,
0xA9, 0x96, 0x2F, 0x0B, 0x93, 0xB8, 0x55, 0xF9, 0x79, 0x93,
0xEC, 0x97, 0x5E, 0xEA, 0xA8, 0x0D, 0x74, 0x0A, 0xDB, 0xF4,
0xFF, 0x74, 0x73, 0x59, 0xD0, 0x41, 0xD5, 0xC3, 0x3E, 0xA7,
0x1D, 0x28, 0x1E, 0x44, 0x6B, 0x14, 0x77, 0x3B, 0xCA, 0x97,
0xB4, 0x3A, 0x23, 0xFB, 0x80, 0x16, 0x76, 0xBD, 0x20, 0x7A,
0x43, 0x6C, 0x64, 0x81, 0xF1, 0xD2, 0xB9, 0x07, 0x87, 0x17,
0x46, 0x1A, 0x5B, 0x9D, 0x32, 0xE6, 0x88, 0xF8, 0x77, 0x48,
0x54, 0x45, 0x23, 0xB5, 0x24, 0xB0, 0xD5, 0x7D, 0x5E, 0xA7,
0x7A, 0x27, 0x75, 0xD2, 0xEC, 0xFA, 0x03, 0x2C, 0xFB, 0xDB,
0xF5, 0x2F, 0xB3, 0x78, 0x61, 0x60, 0x27, 0x90, 0x04, 0xE5,
0x7A, 0xE6, 0xAF, 0x87, 0x4E, 0x73, 0x03, 0xCE, 0x53, 0x29,
0x9C, 0xCC, 0x04, 0x1C, 0x7B, 0xC3, 0x08, 0xD8, 0x2A, 0x56,
0x98, 0xF3, 0xA8, 0xD0, 0xC3, 0x82, 0x71, 0xAE, 0x35, 0xF8,
0xE9, 0xDB, 0xFB, 0xB6, 0x94, 0xB5, 0xC8, 0x03, 0xD8, 0x9F,
0x7A, 0xE4, 0x35, 0xDE, 0x23, 0x6D, 0x52, 0x5F, 0x54, 0x75,
0x9B, 0x65, 0xE3, 0x72, 0xFC, 0xD6, 0x8E, 0xF2, 0x0F, 0xA7,
0x11, 0x1F, 0x9E, 0x4A, 0xFF, 0x73
```

D.3.4.4 EAP SRP-SHA256 Client Key Response Packet

The EAP SRP-SHA256 Client Key is a Response packet sent by the client in response the Challenge packet described in section D.3.4.3. The packet format is shown in Figure 27.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=2     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |  Subtype=0x01 | Value A (variable) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 27: EAP SRP-SHA256 Client Key Packet Format

cc BY-ND

The client shall set the fields in Figure 27 as follows:

- **Value A (variable):** The result of g^a, where "a" is a randomly chosen number in the range 1 .. N (exclusive), as described in section D.2. The randomly chosen number is the client's private key, and the Value A field is the corresponding public key. This field shall be in network byte order and shall not be padded. The "A" value shall not be zero modulo N. If the server receives a bad value for this field, it shall send a Failure packet described in Section D.3.5 and shall disconnect the link.

D.3.4.5 EAP SRP-SHA256 Server Key Request Packet

The EAP SRP-SHA256 Server Key is a Request packet sent by the server after it has received the Client Key packet described in section D.3.4.4. The packet format is shown in Figure 28.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=1     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |  Subtype=0x02 | Value B (variable) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 28: EAP SRP-SHA256 Server Key Packet Format

The server shall set the fields in Figure 28 as follows:

- **Value B (variable):** The result of (kv + g^b) % N, where "b" is a randomly chosen number in the range 1 .. N (exclusive), "v" is the stored verifier from the authentication database, and k=SHA256(N, g), as described in section D.2. The randomly chosen number is the server's private key, and the Value B field is the corresponding public key. This field shall be in network byte order and shall not be padded. The B value shall not be zero modulo N. If the client receives a bad value for this field, it shall send a Failure packet described in Section D.3.5 and shall disconnect the link.

D.3.4.6 EAP SRP-SHA256 Client Validator Response Packet

The EAP SRP-SHA256 Client Validator is a Response packet sent by the client in response to the Server Key packet described in section D.3.4.5. The packet format is shown in Figure 29.

BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=2     |  Identifier   |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |  Subtype=0x02 |             Reserved1         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved2         |U| Value M1 (32 octets) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 29: EAP SRP-SHA256 Client Validator Packet Format

The client shall set the fields in Figure 29 as follows:

- **Reserved1 (16 bits), Reserved2 (15 bits):** The client shall set these bits to zero on transmission, and the server shall ignore them on reception.
- **U (1 bit):** The client shall set this bit to signal to the sender that it intends to use the derived key K (see section D.2 and the M1 computation below) as the PSK passphrase. If this bit is set, the server shall use K as the passphrase to decrypt the traffic received from the client.
- **M1 (32 octets):** The 32 octet values are calculated as follows (see section D.2):
  x = SHA256(s, SHA256(I | ":" | P))
  u = SHA256(A, B)
  S = ((B – kg^x) ^ (a +ux)) % N
  K = SHA256(S)
  M1 = SHA256(SHA256(N) xor SHA256(g), SHA256(I, s, A, B, K))

Upon reception of the Client Validator response, the server shall compute the M1 value and check against the value that is received from the client. If the value matches, the client is deemed authenticated, and the server sends the Server Validator Request packet described in section D.3.4.7. If the M1 value does not match, authentication has failed. The server shall send the Failure packet described in section D.3.5 and shall disconnect the link.

D.3.4.7 EAP SRP-SHA256 Server Validator Request Packet

The EAP SRP-SHA256 Server Validator is a Request packet sent by the server in response to the Client Validator packet described in section D.3.4.6. The packet format is shown in Figure 30.

BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=1     |  Identifier   |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   |  Subtype=0x03 |             Reserved1         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Reserved2         |U| Value M2 (32 octets) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 30: EAP SRP-SHA256 Server Validator Packet Format

The server shall set the fields in Figure 30 as follows:

- **Reserved1 (16 bits), Reserved2 (15 bits):** The server shall set these bits to zero on transmission, and the client shall ignore them on reception.
- **U (1 bit):** The server shall set this bit to signal to the client that it intends to use the derived key K (see section D.2 and the M2 computation below) as the PSK passphrase. If this bit is set, the client shall use K as the passphrase to decrypt the traffic received from the server.
- **M2 (32 octets):** The 32 octet values are calculated as follows:
  u = SHA256(A, B)
  S = ((Av^u) ^ b) % N
  K = SHA256(S)
  M2 = SHA256(A, M1, K)

Upon reception of the Server Validator request, the client shall compute the M2 value and check against what was received from the server. If the value matches, the server is deemed authenticated, and the client shall send the Success packet described in section D.3.5 and the authentication process is complete. If the M2 value does not match, authentication has failed. The client shall send the Failure packet described in section D.3.5 and shall disconnect the link.

D.3.4.8 EAP SRP-SHA256 Passphrase Request Packet

The EAP SRP-SHA256 Passphrase Request packet is used by a receiver to request the passphrase currently in use by the sender. This packet may be used by the client and/or the server, as different passphrases may be in use on either communication direction. The packet format is shown in Figure 31.

CC BY-ND

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |       Payload Length=6        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=1     |  Identifier   |           Length=6            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   | Subtype=0x10  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 31: EAP SRP-SHA256 Passphrase Request Packet

If used, this packet shall only be transmitted after both the server and the client have been authenticated. The recipient of the packet shall respond to it as follows:

- If the recipient of the packet does not support passphrase requests, it shall respond with the Nak packet as per section D.3.3.2.
- If the recipient of the packet supports passphrase requests, it shall respond as follows:
  - If authentication is not complete, the recipient of the packet shall respond with the Failure packet as per section D.3.5.
  - If authentication is complete, the recipient of the packet shall respond with the EAP SRP-SHA256 Passphrase Response as per section D.3.4.9.

D.3.4.9 EAP SRP-SHA256 Passphrase Response Packet

The EAP SRP-SHA256 Passphrase Response provides the passphrase, encrypted by the common session K. As indicated in section D.3.4.8, this response is only issued if authentication has successfully completed. The packet format is shown in Figure 32.

```
 0                   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EAP Version=3 | EAPOL Type=0  |        Payload Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Code=2     |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type=0x13   | Subtype=0x10  |U|H| Reserved  |               :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               :
:                Encrypted Passphrase (variable) ...            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 32: EAP SRP-SHA256 Passphrase Response Packet

The sender shall set the fields in Figure 32 as follows:

- **U (1 bit):** This bit shall be set to the same value as the **U** bit in the validator packet transmitted by the sender (either the Client Validator in Figure 29 or the Server Validator in Figure 30). If this bit is set, it indicates that the session key K is to be used as the

BY-ND

passphrase. In this case, the Encrypted Passphrase field in Figure 32 is not included in the packet.

- **H (1 bit):** This bit indicates the AES key length used in the encryption of the passphrase, as follows:
  - H=0: 128-bit
  - H=1: 256-bit
- **Reserved (6 bits):** This field shall be set to zero by the sender and ignored by the receiver.
- **Encrypted Passphrase (variable):** This field shall be set to the encrypted passphrase, generated as follows:
  - Encryption algorithm: AES-CTR.
  - Key length: as indicated by the H bit.
  - Key: the first 128 or 256 bits of the session key K.
  - IV: the most significant 8 bits of the IV shall be set to the value in the Identifier field. The remainder of the IV shall be padded with zeros.

The recipient of the EAP SRP-SHA256 Passphrase Response packet shall acknowledge its reception by sending a Success packet (see section D.3.5) with the same Identifier as the Passphrase Response packet being acknowledged.

If the sender is employing the on-the-fly passphrase change mechanism described in section 7.4, it may send an unsolicited Passphrase Response Packet with the new passphrase. In such a situation, the sender shall use a value for the Identifier field that is different from the previous Passphrase Response packet. The sender shall not send a new passphrase until it starts using the passphrase from the last Passphrase Response packet to generate the AES key used to encrypt the stream. In a multicast environment, the sender is responsible for contacting all receivers that are selected to receive the new passphrase; the details of this process are left to the discretion of the implementer.

In a long running session, it is theoretically possible to have an IV be re-used, since the Identifier field is only 8 bits. This is a security issue if the same (IV, session key K) combination is used to encrypt different passphrases. The sender should avoid this situation. The following techniques may be used to prevent this IV re-use:

1. The sender invokes the re-authentication procedure described in Section D.5 any time its passphrase is changed.
2. For unsolicited passphrase responses, the sender starts the Identifier field at 255, and decrements it at every passphrase response sent. After sending a passphrase response with identifier 129, the sender invokes the re-authentication procedure of Section D.5. For passphrase requests, the sender starts the Identifier field at 0 and increments it at every request. After receiving the passphrase response for the passphrase request with identifier 127, the sender invokes the re-authentication procedure of Section D.5.

(cc) BY-ND

D.3.5 Success and Failure Packet Formats

Figure 33 shows the packet format for Success (code 3) and Failure (code 4) packets. Such packets have no additional data. The sender shall set the Identifier to match the value of the corresponding Request/Response packet.

```
  0                   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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | EAP Version=3 | EAPOL Type=0  |       Payload Length=4        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Code=3 or 4  |  Identifier   |           Length=4            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 33: Packet Payload Format for Success/Failure Packets

D.4 Protocol Exchanges

This section describes the authentication protocol exchanges. For each exchange, the server selects a starting value for the 8-bit Identifier field as described in section D.3.2, and increments this value for each successive request. In the protocol exchanges presented in this section, the starting value of the Identifier field is denoted by *n*.

Figure 34 shows the protocol exchange for a successful authentication exchange. The client can start the process (message 1) using the EAPOL-Start message or any unsolicited message, such as a Keep Alive packet.

Msg	Client			Direction	Server		
	Message	**Section**	**Identifier**		**Message**	**Section**	**Identifier**
1	EAPOL-Start	D.3.1	-	→			
2				←	Identity Req	D.3.3.1	*n*
3	Identity Res	D.3.3.1	*n*	→			
4				←	Challenge	D.3.4.3	*n*+1
5	Client Key	D.3.4.4	*n*+1	→			
6				←	Server Key	D.3.4.4	*n*+2
7	Client Val	D.3.4.6	*n*+2	→			
8				←	Server Val	D.3.4.7	*n*+3
9	Success	D.3.5	*n*+3	→			

Figure 34: Protocol Exchange for Successful Authentication

Figure 35 shows a possible protocol exchange for the case where the username is unknown to the server. Use of this protocol exchange for unknown usernames allows a third party to "test" whether some usernames exist in the server. If this is a concern, the protocol exchange shown in Figure 36 may be used instead, with the server sending random data in messages 4 and 6.

BY-ND

Msg	Client			Direction	Server		
	Message	**Section**	**Identifier**		**Message**	**Section**	**Identifier**
1	EAPOL-Start	D.3.1	-	→			
2				←	Identity Req	D.3.3.1	*n*
3	Identity Res	D.3.3.1	*n*	→			
4				←	Failure	D.3.5	*n*

Figure 35: Protocol Exchange for Unknown Username

Figure 36 shows the protocol exchange when the client authentication fails. This is determined at the server when the value carried in the Client Validator packet in message 7 does not match the value computed by the server. This will happen if the client does not have the correct password. This exchange can also be used when the client username does not exist in the server, with random (fake) data in messages 4 and 6.

Msg	Client			Direction	Server		
	Message	**Section**	**Identifier**		**Message**	**Section**	**Identifier**
1	EAPOL-Start	D.3.1	-	→			
2				←	Identity Req	D.3.3.1	*n*
3	Identity Res	D.3.3.1	*n*	→			
4				←	Challenge	D.3.4.3	*n*+1
5	Client Key	D.3.4.4	*n*+1	→			
6				←	Server Key	D.3.4.4	*n*+2
7	Client Val	D.3.4.6	*n*+2	→			
8				←	Failure	D.3.5	*n*+2

Figure 36: Protocol Exchange for Client Authentication Failure

Figure 37 shows the protocol exchange when the server validation fails. This will likely only happen if the client is connecting to a device who is impersonating the server. Such a device will have no knowledge of the password; it can go through the steps and pretend they completed successfully, but at message 8, the client detects that the server validation has failed.

Message 9 in Figure 37 is optional. If server validation fails, the client may simply stop communicating with the server and may ignore all further messages from it.

CC BY-ND

Msg	Client			Direction	Server		
	Message	Section	Identifier		Message	Section	Identifier
1	EAPOL-Start	D.3.1	-	→			
2				←	Identity Req	D.3.3.1	*n*
3	Identity Res	D.3.3.1	*n*	→			
4				←	Challenge	D.3.4.3	*n*+1
5	Client Key	D.3.4.4	*n*+1	→			
6				←	Server Key	D.3.4.4	*n*+2
7	Client Val	D.3.4.6	*n*+2	→			
8				←	Server Val	D.3.4.7	*n*+3
9	Failure	D.3.5	*n*+3	→			

Figure 37: Protocol Exchange for Server Authentication Failure

D.5 Re-Authentication

In some situations, it is desirable to periodically re-authenticate the endpoints (server and/or client). If requested by the other endpoint, server and client shall respond to the re-authentication process described in this section. Server and client may support initiating the re-authentication process.

During re-authentication, normal data exchange between server and client shall continue. If re-authentication fails, all communication between server and client shall stop.

To start re-authentication, the server shall send an Identity Request packet (section D.3.3.1) to the client to start the process. This is packet 2 in Figure 34. The protocol will then follow the steps shown in Figure 34 for successful authentication, or Figure 35, Figure 36 or Figure 37 for authentication failures.

To start re-authentication, the client shall send an EAPOL-Start packet (section D.3.1) to the server to start the process. This is packet 1 in Figure 34. The protocol will then follow the steps shown in Figure 34 for successful authentication, or Figure 35, Figure 36 or Figure 37 for authentication failures.

The re-authentication process will yield a new session key K. If either the server and/or the client signal the use of the session key K as the PSK passphrase by using the **U** bit in the messages described in sections D.3.4.6 and/or D.3.4.7, the protocol described in section 7.4 shall be used to switch to the new passphrase. If server and/or client are using the session key as the PSK passphrase, they are not required to set the **U** bit in the re-authentication session; in such a case, the original passphrase remains in use.

The interval between successive re-authentication sessions between a server and a given client shall be no less than 60 seconds.

BY-ND

D.6 UDP Transport Considerations

The protocol described in this Annex runs over UDP. Therefore, it is possible for packets to be re-ordered, duplicated, or dropped.

The authentication process is driven by the server. If an expected response is not received by a given timeout, the request is retried. The server shall discard duplicate responses. After a certain number of retries, the server shall discard all the information received and shall wait to be contacted again by the client. The number of retries is left at the discretion of the implementer but should be no less than three. The timeout is also left at the discretion of the implementer, and it should be a multiple of the round-trip time between server and client. While RIST includes mechanisms to measure the round-trip time, such mechanisms are only available after the connection is established. Therefore, the round-trip time needs to be determined by means outside of this Specification.

As indicated in Figure 34, a successful authentication exchange requires four packets from the server, and the first packet is the Identity Request (section D.3.3.1). The client shall save the Identifier from that message, which we will denote by n. The client shall only respond to packets with Identifier in the range of n to $n+3$ and shall silently discard all packets with identifier values not in this range. If a client receives a packet with an identifier value of k, with $n < k \leq n+3$, the client shall silently discard this packet if it has yet not received the packet with identifier $k-1$. If the last response the client sent was for identifier k, with $n < k \leq n+3$, and now the client receives a packet with identifier m, with $n \leq m \leq k$, it shall re-send the original response for identifier m. During an authentication session, the random server and client keys shall not change. For example, if the client receives a duplicate Challenge packet (message 4 in Figure 34), it shall re-send the same Client Key response as with the original request.

The client shall implement a timeout mechanism for expected messages from the server, until authentication completes. If a protocol message does not arrive by the timeout, the client shall restart the authentication process. This timeout is left to the discretion of the implementer and should be a multiple of the round-trip time.

For the Passphrase Request/Response messages in sections D.3.4.8 and D.3.4.9, the sender of each message shall implement a timeout mechanism after each message. If a response is not received by the timeout, the message is retried. After a certain number of retries, the sender shall abort the process. The recovery mechanism in this case is left to the discretion of the implementer.

D.7 Multicast Authentication (Informative)

The mechanism described in this Annex can be used to provide authentication in a multicast environment under the following conditions:

BY-ND

1. A sender is transmitting a RIST Main Profile stream with PSK encryption to an IP multicast address.
2. While any device in the network can join the multicast, only authenticated devices are allowed to decrypt the content.
3. Unicast bidirectional communication is possible between the sender and all receivers of the multicast.
4. Receivers are configured with the multicast address and UDP port through external means not covered by this Specification.
5. The sender of the stream is also the authentication server. Alternatively, a separate authentication server can be used, under the following conditions:
    a. The multicast receivers are configured with the IP address and UDP port of the authentication server through external means not covered by this Specification.
    b. The authentication server has knowledge of the passphrase in use to encrypt the multicast stream.

Operation is as follows:

- The receiver joins the multicast. Since it does not have the passphrase yet, it cannot decrypt the content.
- The receiver uses the source IP address and source UDP port from the received multicast packets to initiate an authentication session with the sender of the content. Alternatively, the receiver may instead contact a pre-configured authentication server. The authentication session is initiated with an EAPOL-Start packet, as described in in section D.3.1.
- If the authentication session finishes successfully, the receiver requests the PSK passphrase using the passphrase request packet described in section D.3.4.8, and the server responds with the passphrase response packet described in section D.3.4.9.
- At this point, the receiver can now decrypt the PSK stream. As required by this Specification, the receiver will send keep-alive unicast packets back to the sender.

The re-authentication process described in section D.5can be used in this multicast environment. As indicated in that section, the re-authentication can be started either by the client (receiver) or the server (sender).

If a receiver-initiated re-authentication fails, the receiver simply leaves the multicast and stops processing the data.

Senders know which receivers are active due to the keep-alive messages. If the re-authentication fails with a subset of the receivers, the sender will need to change the PSK passphrase to remove their access. The process will happen as follows:

- The server generates a new passphrase through means outside this Specification.

cc BY-ND

- The server sends the new passphrase to the subset of receivers for which authentication succeeded, using the passphrase response packet and mechanism from section D.3.4.9. For each allowed receiver, the session key from the new successful authentication will be used.
- Once all the allowed receivers have the new passphrase, the on-the-fly passphrase change mechanism from section 7.4 is used to switch to the new passphrase.

D.8 Multi-Link Operation (Informative)

It is possible to use the authentication mechanism described in this Annex in a multi-link application. In this case, there are multiple links between the server and the client. These links can be used in Bonding (Load Share) mode, where the traffic is spread between the links, or in Seamless Switching mode, where the traffic is replicated across the links. The following considerations apply to both modes of operation, at the discretion of the implementer:

- One option is to run independent authentication sessions on each link. Even if the same username/password is used in all links, the session key will be different per link. If the session key is used as the passphrase, each link will have its own PSK encryption key.
- Another option is to combine the links. In this case, the GRE sequence number (see Figure 2) needs to be included in the packet. Each endpoint implements a re-order section in the receive buffer (as described in TR-06-1 Section 5.3.1), where packets are re-ordered and duplicates are removed. The resulting packet flow is then used for authentication. In this case, a single authentication session and passphrase will be used across all links. This case also allows for links to be dynamically added as needed at any time.

D.9 Authentication Example (Informative)

This section provides a numerical example of an authentication exchange. It is provided to allow implementations to be checked against known values. In the example below, all numeric values are presented in base 16 (hexadecimal). Underlined values are random numbers, bold values are computation results. The example uses the weakest legal modulus value (512-bit) for simplicity. Use of this modulus length in actual implementations is discouraged.

The inputs for the algorithm in this example are:

Modulus "N":	
	`D66AAFE8E245F9AC245A199F62CE61AB8FA90A4D80C71CD2A` `DFD0B9DA163B29F2A34AFBDB3B1B5D0102559CE63D8B6E86B` `0AA59C14E79D4AA62D1748E4249DF3`
Generator "g":	`2`
Username "I":	`rist`
Password "P":	`mainprofile`

BY-ND

The server generates a random salt “s” for each username/password pair. In this example, the following salt value is used:

Salt “s”:

```
72F9D5383B7EB7599FB63028F47475B60A55F313D40E0BE02
3E026C97C0A2C32
```

The server computes the password verifier “v” as follows (see section D.2):

x = SHA256(s, SHA256(I | “:” | P))

v = g^x

For the example inputs, the values will be:

Value “x”:

```
850D72F3946EC76BA4A52097E6DF990F88CFB9A40252B7F52
BEC2E0D20BFE892
```

Verifier “v”:

```
2E06FEA163D6E9FF0FA7ED6C59233389D0DBA0C08C0F72F6D
AD1E2A3D8B92A772F070439D1C11B87FA990D2DAF04EB830C
C77D61ACC4B253297379CD8E6DC3AF
```

The server stores the salt “s” and the verifier “v” indexed by the username “I”. There is no need for the server to store the password “P”. Once this is in place, the server is ready to authenticate the client.

As illustrated in Figure 34, the client starts the authentication process by sending the EAPOL-Start packet. The server will send the Identity Request packet, and the client will respond with the Identity Response packet, indicating the “**`rist`**” username. The server will then send the Challenge Request packet from section D.3.4.3 containing the values “s”, “g” and “N”, which are cached by the client.

The client generates a random number “a” between 1 and N-1. In this example, the following value is used:

Value “a”:

```
138AB4045633AD14961CB1AD0720B1989104151C070879449
1113302CCCC27D5
```

The client uses the random value “a” to compute the client key “A” using the following formula:

A = g^a

In this example, the client key “A” value is computed as:

BY-ND

Client key “A”:

**92C4CEFB95A1AE2E576A252B19273FD4613F44FDA4AC8CC84
A089D5740756223943882BAD34CB55F35139CDDB60E0D19AC
D2B884CFB27F53C8EA969269ABE014**

The client key “A” is returned to the server using the Client Key Response Packet from section D.3.4.4. The server checks that A % N is not zero and caches the value “A”.

The server generates a random number “b” between 1 and N-1. In this example, the following value is used:

Value “b”:

**ED0D58FF861A1FC75A0829BEA5F1392D2B13AB2B05CBCD6ED
1E71AAAD761E856**

The server computes the server key “B” as follows:

k = SHA256(N, g)
B = (kv + g^b) % N

In this example, the server key “B” is computed as follows:

Value “k”:

**890D0AC9E42A7F909D3CAA9A0FF115C52A1DC8DED10839EF9
583C4E35EA76E78**

Server key “B:

**858CDC811B5EEAA7F58C12767D309EBD2DF1D46F59EF56860
52E6511CF853CA4E66910BDBD28CBEAE2F2DEE7F6BF375675
7BD69E88D48C77B5371A82EF52AD84**

The server key “B” is sent to the client using the Server Key Request Packet from section D.3.4.5. Upon receiving this value, the client performs the following computations:

x = SHA256(s, SHA256(I, “:”, P))
k = SHA256(N, g)
u = SHA256(A, B)
S = ((B – kg^x) ^ (a +ux)) % N
K = SHA256(S)
M1 = SHA256(SHA256(N) xor SHA256(g), SHA256(I), s, A, B, K)

The values “x” and “k” for this example have already been computed and their values can be found above. The remaining values are:

BY-ND

Value “u”:

```
4C53609F8B4F9F6F534DF35ABFDD760E8EEC1117EB01421A6
6A425C059789A94
```

Value “S”:

```
BEAEF3B089BE9022135C8798C777C30609546C1C0F305186E
F30070677F6FFC221EEC9E2BFDD405FEA6589A0D8BB54DF44
7187C265218AB3333064587DB7F27C
```

Session key “K”:

```
D2270AB6B54F80D246E474F8DD76FC7DECA3F49FBDF419E08
2DC989B38608C34
```

Validator value “M1”:

```
E28147C801BAB9C37647C1FF4A29FA720E3F5676434FB85EA
9A752CC1F9B1AD4
```

The client caches the session “K” for possible future use as a passphrase and returns the validator value “M1” back to the server using the Client Validator Response Packet from section D.3.4.6.

Upon receiving the “M1” value from the client, the server computes its own version of “K” and “M1” using the following formulas:

u = SHA256(A, B)
S = ((Av^u) ^ b) % N
K = SHA256(S)
M1 = SHA256(SHA256(N) xor SHA256(g), SHA256(I), s, A, B, K)

If the “M1” value computed by the server matches what it received from the client, the client is deemed authenticated. In this case, the session key “K” will also match. The server computes its validator value “M2” as follows:

M2 = SHA256(A, M1, K)

For this example, the value is:

Validator value “M2”:

```
84F19797916FBDCAB1321CA78B575B145B586150248AFAA15
6361B8BCB139B32
```

The server sends the validator value “M2” to the client using the Server Validator Request Packet from section D.3.4.7. The client computes “M2” using the same formula as the server, and, if the values match, the server is deemed validated from the client.

(cc) BY-ND