VSF

VIDEO SERVICES FORUM

Video Services Forum (VSF) Technical Recommendation TR-06-2:2020 Deprecation Notice

April 22, 2021

VSF Technical Recommendation TR-06-2:2020 defined a Pre-Shared Key (PSK) operation mode, where a secret passphrase is distributed through some out-of-band means to every participating node. The passphrase is used to derive a 128-bit or 256-bit key. Content is encrypted using AES-CTR and this key.

AES-CTR operates by encrypting a set of Initialization Vectors (IVs) to create ciphertext blocks, which are subsequently XOR-ed with the message. This method is secure as long as IVs are not re-used with the same key.

The IV generation method specified in TR-06-2:2020 Section 7.2 and illustrated in Figure 6 is insecure because IVs are partially reused from packet to packet. This issue is addressed in TR-06-2:2021. However, PSK implementations following TR-06-2:2020 are **NOT** interoperable with TR-06-2:2021. A version field has been added to the packet headers in TR-06-2:2021 to allow secure implementations to identify TR-06-2:2020 devices, if desired.

Additionally, the optional PSK Authentication method described in Section 7.5 was found to be too cumbersome to implement and has been deprecated. A new method is included in TR-06-2:2021.

Implementation of the PSK functionality as described in TR-06-2:2020 IS NOT recommended due to security issue described in this note.

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

DEPRECATED!

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

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

DEPRECATED!

Approved by VSF Board

March 24, 2020

© 2020 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.

BY ND

DEPRECATED!

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 5.5.4), 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.

DEPRECATED!

cc 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.

DEPRECATED!

cc BY-ND

Table of Contents

DEPRECATED!

BY-ND

DEPRECATED!

cc BY-ND

DEPRECATED!

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 (MVA Broadcast Consulting)	Sergio Ammirata (DVEO)	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)	Hermann Popp (Arri)
Steve Riedl (Turner)	Adi Rozenberg (VideoFlow)	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;

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.

DEPRECATED!

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

DEPRECATED!

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

cc BY-ND

- 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.

DEPRECATED!

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.

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 5.5.
- Streams running through the tunnel shall comply with VSF TR-06-1, RIST Simple Profile.

BY-ND

- 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 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 5.5.3 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.

DEPRECATED!

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

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

Figure 1: GRE header with no options

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

BY-ND

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

Figure 2: GRE header with sequence number

The **Reserved0** field shall be set to 0 (zero) by the sender and shall be ignored by the receiver.

5.2 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 5.2.2 shall be used. Implementation of this mode will require the Video Services Forum to register an EtherType. This Specification proposes an interim value for the EtherType, which may change in the future. Support for Reduced Overhead Mode is optional.

5.2.1 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.

5.2.2 Reduced Overhead Mode

In this mode, the encapsulated packet is assumed to be a UDP packet. Implementations shall use the value of 0x88B6 for the **Protocol Type** field. The GRE payload shall start with the subset of the UDP header indicated in Figure 3, denoted as "Reduced UDP Header". Fields shall be in network byte order, MSB first.

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

Figure 3: Reduced UDP Header

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.

DEPRECATED!

cc BY-ND

- The receiving tunnel end point 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.
- 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).
- The source and destination UDP ports for the encapsulated packet shall be the source and destination UDP ports from the Reduced UDP Header.

5.3 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.
- **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).

DEPRECATED!

cc BY-ND

    - 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.
        - 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. Such mixed operation shall be permitted by this Specification, but this is optional.

5.4 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.

5.5 Tunnel Establishment

5.5.1 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.

DEPRECATED!

BY-ND

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.

As far as this particular 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 5.5.3 and 5.5.4, but allows the use of any type of periodic GRE-encapsulated message, as long as the requirements of section 5.5.2 are met.

5.5.2 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. Keep-alive message timeout shall be 60 seconds.
- As long as the tunnel server is receiving keep-alive messages from the tunnel client, it shall keep sending messages to the tunnel client. If the tunnel server stops receiving messages from the tunnel client, the tunnel server shall stop sending messages to the tunnel client after a timeout.
- If an endpoint fails to receive either a keep-alive message or traffic on a session after the 60-second 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 5 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.

5.5.3 Keep-Alive Message Format

The optional keep-alive message defined in this document, shall be encapsulated in a GRE packet with Protocol Type set to 0x88B5. The C bit shall be set to zero. The S and K bits shall be set according to the relevant section of this document. The payload of the keep-alive message shall be encoded in JSON format, as defined in RFC 8259. The message is depicted in Figure 4, including a minimal GRE header. Fields shall be in network byte order, MSB first.

DEPRECATED!

BY-ND

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C| |K|S| Reserved0       | Ver | Protocol Type = 0x88B5        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|   48-bit MAC Address          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |X|R|B|A|P|E|L|N|D|T|V|J| Rsvd1 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Message Payload (JSON format)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Keep-Alive Message

DEPRECATED!

The fields shall be set 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:
    - **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 5.5.5).
    - **T:** If this flag is set, this is a Reconnect message (described in section 5.5.6).
    - **V:** If this flag is set, device supports Reduced Overhead Mode (described in section 5.2.2).

BY-ND

  - **J:** If this flag is set, device is capable of sending, receiving and processing JSON information (described in section 5.5.4).
- **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 5.5, this Specification allows devices to use any periodic GRE-encapsulated packet as the keep-alive message. If a device receives periodic keep-alive messages that do not 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 4 is assumed to be zero.
- The sending device will never issue Disconnect and Reconnect messages (sections 5.5.5 and 5.5.6) 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 4 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).
- The MAC Address of the remote device is unknown.

5.5.4 Keep-Alive Message Payload

The keep-alive message payload shall be in JSON format for extensibility. 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:

DEPRECATED!

cc BY-ND

- **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.
  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.

5.5.4.1 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
 }
}
```

DEPRECATED!

BY-ND

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.

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.

DEPRECATED!

BY-ND

- 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 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).

DEPRECATED!

5.5.5 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 5.5.2 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 acknowledgement. In any case, upon receipt of a keep-alive message with **D**=1, the device receiving the message shall terminate the tunnel and shall stop sending messages if the device processes the Disconnect Message. 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.

5.5.6 Reconnect Message

The keep-alive message header includes the **T** bit to restart the IP Address negotiation described in section 5.5.4. 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.

cc BY-ND

DEPRECATED!

    - 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.

5.5.7 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.
- **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.

BY-ND

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.

5.5.8 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.

DEPRECATED!

BY-ND

DEPRECATED!

Note: For RIST Simple Profile flows using transport streams over Ethernet, this is guaranteed because there are 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.

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.

BY-ND

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.
  - 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.

DEPRECATED!

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 5 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.

BY-ND

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

Figure 5: GRE header with Key/Nonce

DEPRECATED!

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.
- 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 operation shall be derived by padding the 4-byte S field with 12 bytes of zeros. Bytes 0 through 11 of the IV shall be set to zero, and bytes 12 through 15 shall be set to the value of the S field, with byte 12 being the MSB of the S field. This process is illustrated in Figure 6.

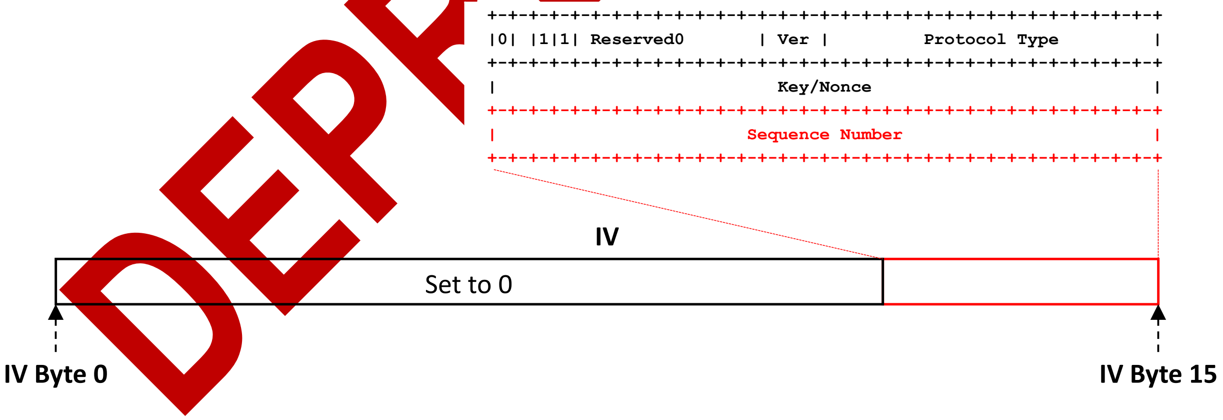

Figure 6: IV Generation

- 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.

cc BY-ND

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 and Appendix B.1.
- 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. A numerical example is provided in Appendix B.

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.

DEPRECATED!

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 7.

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

Figure 7: 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.

DEPRECATED!

BY-ND

7.5 PSK Authentication Using EAPOL-TLS-SRP

The TLS-SRP protocol, as described in RFC 5054, may be used to securely provide username/password authentication between devices when using PSK. The details of the TLS-SRP implementation described in section 6.4. Implementation of this mechanism is optional.

Devices implementing TLS-SRP for authentication of PSK clients shall use the EAP-TLS protocol, as described in RFC 5216, to implement a TLS handshake channel.

The EAP packets shall be encapsulated into GRE with Ethernet type 0x888E following the EAPoL specification, as defined in 802.1X-2010 section 11.

At the end of a successful handshake and authentication exchange, the derived key provided by TLS-SRP shall not be used for any PSK related processing. The encryption key shall be generated as described in section 7.3.

Once the peer has been authenticated, its source IP address and port combination shall be cached as authorized and its data be processed normally until the session ends.

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

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.

DEPRECATED!

cc 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 8 below.

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

Figure 8: Generic RTP Header Extension (from RFC 3550)

The RTP Header Extension for RIST Main Profile shall be implemented as shown in Figure 9. Fields shall be in network byte order, MSB first.

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

Figure 9: 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 8. For RIST Main Profile, this field shall have the value 0x5249, corresponding to the ASCII codes for "RI".
- **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 always be set to 1.

DEPRECATED!

BY-ND

- **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 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.

DEPRECATED!

BY-ND

If the RIST receiver needs to send NACKs for packet blocks (ranges or bitmasks) that have different high-order extension values, these requests need to 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 10. Fields are in network byte order, MSB first.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|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 10: 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 actually received in the payload to the number of bits set in the NPD field. The suggested processing of the NPD field is as follows:

- 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:

DEPRECATED!

BY-ND

- 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 useful for payloads composed of only 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
- 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:

DEPRECATED!

cc BY-ND

- 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.

DEPRECATED!

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 11.

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

Figure 11: FEC Header (from SMPTE-2022-1)

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.

BY-ND

- 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.

DEPRECATED!

9 Compatibility between RIST Main Profile and Simple Profile Devices

RIST Main Profile adds a number of 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:

- 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.

cc BY-ND

Appendix A Certificate Management (Informative)

This Appendix describes a number of 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, in order 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 a number of 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 or not 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.

DEPRECATED!

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 blacklist 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 (a whitelist), 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 Whitelists and Blacklists

In order to implement either a whitelist or a blacklist, the device needs to use one of the certificate fields as the identifying entry for deciding whether or not the certificate is in the list (either white or black). 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

DEPRECATED!

cc 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

In order 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 particular order for these items.

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

```
-----BEGIN CERTIFICATE-----
MIIDfjCCAuegAwIBAgIJALYx7VgDX7U1MA0GCSqGSIb3DQEBBQUAMIGHMQswCQYD
VQQGEwJVUzELMAkGA1UECBMCQ0ExEDAOBgNVBAcTB1Nhbkpvc2UxFzAVBgNVBAoT
… (certificate continues)
gQBAYeJpSnoKWk3c5Uy0PZ1+/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:

DEPRECATED!

BY-ND

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

DEPRECATED!

BY-ND

Appendix B PSK Key Generation Example (Informative)

This appendix 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 12 shows the packet received from the network with the above nonce.

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

Figure 12: 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:

```
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())
```

DEPRECATED!

BY-ND

Appendix 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 Appendix 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 is used (see Section 6.4), 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 5.5.3 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 5.5.3 are used, and the message has a JSON payload as described in Section 5.5.4, 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 5.2.1, 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.
5. If the server and clients are using the Reduced Overhead Mode defined in Section 5.2.2, the source UDP port in the reduced UDP header shown in Figure 3 can be used to

DEPRECATED!

cc BY-ND

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.

DEPRECATED!

cc BY-ND