# Video Services Forum (VSF) Technical Recommendation TR-10-13

## Internet Protocol Media Experience (IPMX): Privacy Encryption Protocol (PEP)

DRAFT

**January 19, 2024**

© 2024 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.

http://www.videoservicesforum.org

DRAFT

**INTELLECTUAL PROPERTY RIGHTS**

RECIPIENTS OF THIS DOCUMENT ARE REQUESTED TO SUBMIT, WITH THEIR COMMENTS, 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.

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 IMPLEMENTER 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THE FOREGOING NEGATION OF DAMAGES IS A FUNDAMENTAL ELEMENT OF THE USE OF THE CONTENTS HEREOF, AND THESE CONTENTS WOULD NOT BE PUBLISHED BY VSF WITHOUT SUCH LIMITATIONS.

## Executive Summary

This Technical Recommendation describes a method generating keying material for encrypting, decrypting and authenticating media content over multicast and unicast networks. It is designed to support multiple types of transport protocol adaptations. The default adaptation defined in this document describes privacy encryption of media streams having an RTP payload format. Other adaptations are possible for other transport protocols such as USB over IP, SRTP and SRT.

DRAFT



cc BY-ND

## Table of Contents



## 3 Introduction (informative)

The privacy encryption protocol (PEP) is based on a set of Pre-Shared Keys (PSK) stored in Sender and Receiver Devices to control access to media content. A Receiver can access content



CC BY-ND

DRAFT

from various Senders, each possibly having its own PSK. A set of PSKs are programmed in each Sender and Receiver Device in a secure way through a proprietary device configuration interface using a secure communication method.

A **privacy** attribute and an associated set of parameters in the Sender's SDP transport file and/or in NMOS transport parameters provide a Receiver with all the necessary information for deriving, the encryption key used by a Sender, by using the PSK. A Controller transports keying material information from a Sender to subscribed Receivers through the SDP transport file and/or NMOS transport parameters.

This Technical Recommendation presents the key derivation process and the associated requirements. The key derivation process may be used with various transport protocol adaptations. This Technical Recommendation presents the adaptation for RTP. Companion documents can be used to present adaptations for other transport protocols such as USB over IP, SRTP, SRT, etc.

Figure 2 illustrates the various layers that comprise the Privacy Encryption Protocol. The Key Derivation layer is responsible for deriving a **privacy_key** from a given **PSK** and associated keying material. The ECDH layer provides the extra **key_pfs** keying material for peer-to-peer scenarios. The Key Derivation layer is also responsible for providing the dynamic **key_version** to transmit in-band to the Protocol Adaptation layer (used for the KDF of the encryption key). The Protocol Adaptation layer is responsible for providing the authentication and encryption/decryption functions and for transmitting/receiving the ciphered content using an associated transport protocol. The Protocol Adaptation layer is also responsible for providing the dynamic **key_version,** received in-band, to the Key Derivation layer (used for the KDF of the decryption key).

DRAFT



CC BY-ND

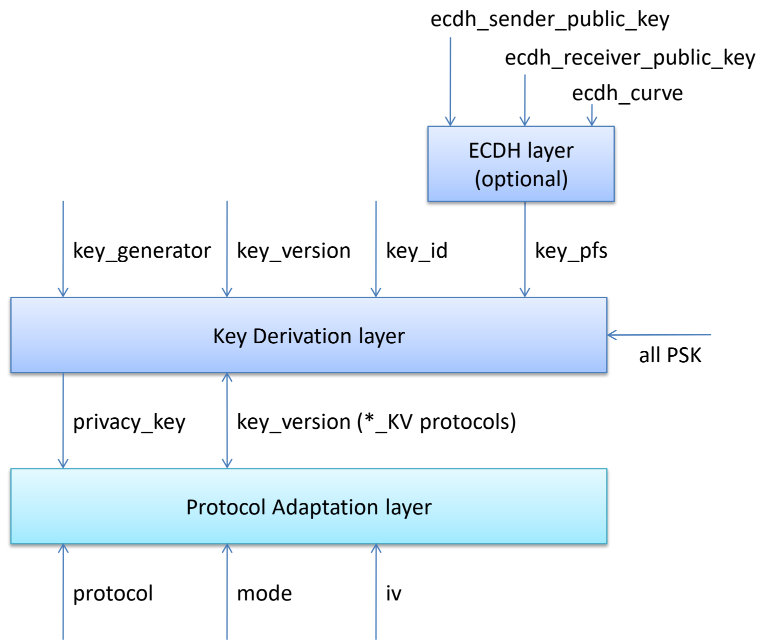



Figure 1 - Privacy Encryption Protocol layers

# 4 Contributors

The following individuals participated in the Video Services Forum IPMX working group that developed this Technical Recommendation.

| | |
|---|---|
| Alain Bouchard (Matrox) | Andreas Hildebrand (ALC NetworX) |
| Andrew Starks (Macnica) | Ben Cope (Intel) |
| Brad Gilmer (VSF) | Clark Williams (Christie Digital Systems) |
| Dan Holland (DHC) | Daniel Bouquet (Analog Way) |
| Danny Pierini (Matrox) | François-Pierre Clouet (intoPIX) |
| Greg Schlechter (Intel) | Greg Stigall (Warner Media) |
| Jack Douglass (PacketStorm) | Jean Lapierre (Matrox) |
| Jed Deame (Nextera Video) | Karl Paulsen (Diversified) |
| Lynn Rowe (AlchemediaSG LLC) | Marc Levy (Macnica) |
| Mike Boucke (AJA) | Paulo Francisco (Evertz) |
| Prinyar Boon (PHABRIX) | Raymond Hermans (Adeas) |
| Sara Seidel (Riedel) | Steve Kolta (Christie Digital) |
| Tadahiro Watanabe (Macnica) | Thomas True (NVIDIA) |
| Tim Bruylants (intoPix) | Wes Simpson (LearnIPvideo) |
| Wojtek Tryc (Ross Video) | |


BY-ND

## 5 About the Video Service 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;
- 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.

## 6 Conformance Notation

Normative text describes elements of the design that are indispensable or contain the conformance language keywords: "shall," "should," or "may."

Informative text 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 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.

DRAFT



cc BY-ND

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.

# 7 Normative References

- NIST.FIPS.180-4: FIPS PUB 180-4, Secure Hash Standard (SHS), August 2015.
- NIST.FIPS.197: Advanced Encryption Standard (AES), November 26, 2001
- NIST.FIPS.198-1: FIPS PUB 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008
- NIST.SP.800-108Rev1: Recommendation for Key Derivation Using Pseudorandom Functions, August 2022.
- NIST.SP.800-186: Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters, February 2023.
- NIST.SP.800-38a: Recommendation for Block Cipher Modes of Operation, Methods and Techniques, December 2001
- NIST.SP.800-38b: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication May 2005
- NIST.SP.800-56aRev3: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography, April 2018
- RFC 8088: How to Write an RTP Payload Format, May 2017
- RFC 8285: A General Mechanism for RTP Header Extensions, October 2017
- SEC1v2 SEC 1: Elliptic Curve Cryptography, May 21, 2009, Version 2.0
- JT-NM TR-1001-1:2020: System Environment and Device Behaviors For SMPTE ST 2110 Media Nodes in Engineered Networks, November 11, 2020
- VSF TR-10-5: Internet Protocol Media Experience (IPMX): HDCP Key Exchange Protocol, March 23, 2022
- VSF TR-10-8: Internet Protocol Media Experience (IPMX): NMOS Requirements, April 14, 2023
- ST 2022-7:2019: Seamless Protection Switching of RTP Datagrams, December 26, 2018

DRAFT



CC BY-ND

# 8 Definitions

| | |
|---|---|
| \|\| | A concatenation of Octet Strings in big-endian format. |
| **Controller** | A software component that manages, controls and coordinates the operations of a number of Senders and Receivers in a network, such as an NMOS Controller. |
| **CRLF** | A carriage return character immediately followed by a line feed character. |
| **Device** | A physical or virtual entity that is designed to perform a specific function or task within a system. |
| **Octet String** | An ordered sequence of Octets in big-endian format. |
| **Octet** | A binary value of eight bits. |
| **Perfect Forward Secrecy** | A security property in cryptographic protocols where the compromise of long-term secret keys does not compromise the confidentiality of past or future communications. |
| **Pre-Shared Key** | A cryptographic key that is agreed upon and shared in advance between communicating parties. |
| **Receiver Device** | A Device having a number of Receivers. |
| **Receiver** | As per "JT-NM TR-1001-1:2020, v1.1", it is a "Receiver Media Node" that consumes a privacy encrypted stream using ST 2110 or IPMX. |
| **Sender Device** | A Device having a number of Senders. |
| **Sender** | As per "JT-NM TR-1001-1:2020, v1.1", it is a "Sender Media Node" that produces a privacy encrypted stream using ST 2110 or IPMX. |

# 9 Abbreviations

| | |
|---|---|
| **AAD** | Additional Authenticated Data |
| **KDF** | Key derivation function |
| **MAC** | Message Authentication Code |
| **PEP** | Privacy Encryption Protocol |

DRAFT



CC BY-ND

| | |
|---|---|
| **PFS** | Perfect Forward Secrecy |
| **PRF** | Pseudo Random Function |
| **PSK** | Pre-shared key |
| **ECDH** | Elliptic Curve Diffie-Hellman |

## 10 Notations

**Octet**
An Octet is represented in text form as two hexadecimal digits. A hexadecimal digit is one of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, a, b, c, d, e, f. Unless otherwise specified both upper and lower case letters are allowed. The leftmost hexadecimal digit corresponds to the most significant 4 bit of the Octet. The rightmost digit corresponds to the least significant 4 bit.

**Octet String**
An Octet String is represented in text form as a sequence of Octets from left to right. Unless otherwise specified the Octets may be separated by spaces. The leftmost Octet corresponds to the most significant Octet of a numeral value and occupies the lowest address in memory. The rightmost Octet corresponds to the least significant Octet of a numeral value and occupies the highest address in memory. $Octet_0$ is the leftmost Octet of an Octet String. $Octet_N$ is the rightmost Octet of an Octet String counting N+1 Octet. For an Octet String named my_var, $my\_var_0$ corresponds to the $Octet_0$ of such an Octet String.

## 11 Operations and Functions

$CIPH_K(M)$
As per the definition in NIST.SP.800-38a. The variable K may be replaced by the name of a key as in $CIPH_{key}(M)$ where the key is named 'key'.

CMAC(K, M)
As per the definition in NIST.SP.800-38b. The variable K may be replaced by the name of a key as in CMAC(key, M) where the key is named 'key'.

HIGH(X)
Return the most significant N bits of X where N corresponds to half the number of bits in X

HMAC-SHA-512/256(K,M)
As per the definition in NIST.FIPS.198-1 and NIST.FIPS.180-4 The variable K may be replaced by the name of a key as in HMAC-SHA-512/256 (key, M) where the key is named 'key'.



(cc) BY-ND

| | |
|---|---|
| LOW(X) | Return the least significant N bits of X where N corresponds to half the number of bits in X |

## 12 Privacy Key Derivation

Senders and Receivers that produce or consume a privacy encrypted stream shall derive a **privacy_key** as described in this Technical Recommendation and utilize it to obtain an encryption or decryption key. Such stream may be unidirectional or bidirectional and may comprise a set of sub-streams, each independently privacy encrypted, and having its own direction. Sub-streams that are not independently privacy encrypted, such as FEC sub-streams, are not subject to this recommendation.

***Note***: By definition a Sender produces one stream and a Receiver consumes one stream. The concept of sub-stream is used to support scenarios where a Sender multiplexes a number of sub-streams and encrypts/decrypts them independently. See the section 14 for more details.

The **privacy_key** shall be derived from a Pre-Shared Key (**PSK**), a key generator (**key_generator**), a key version (**key_version**) and a Perfect Forward Secrecy shared secret (**key_pfs**) using a KDF in counter mode as per NIST.SP.800-108Rev1 section 4.1.

The KDF function shall use CMAC as the PRF as defined by NIST SP 800-108Rev1 for a KDF using a PRF in counter mode for a PSK of 128 and 256 bits.

The KDF function shall use HMAC-SHA-512/256 as the PRF as defined by NIST SP 800-108Rev1 for a KDF using a PRF in counter mode for a PSK of 512 bits. The CMAC algorithm is specified in NIST SP 800-38b and may use an AES-128 (128-bit key) or AES-256 (256-bit key) block cipher. The HMAC and SHA-512/256 algorithms are specified in NIST FIPS 198-1 and NIST FIPS 180-4.

A Perfect Forward Secrecy mode based on ECDH as per NIST.SP.800-56aRev3 section 6.1.2.2 may be used to further restrict access to a privacy encrypted stream to only the two parties on each side of a peer-to-peer connection. The ECDH is performed using domain parameters as per NIST.SP.800-186 for curves “secp256r1” (NIST P-256), “25519” (NIST Curve25519), “448” (NIST Curve448) and “secp521r1” (NIST P-521).

128-bit key derivation (PSK is 128 bits):

**privacy_key** = CMAC(**PSK**, AB || **key_generator** || **key_version** || **key_pfs**)

256-bit key derivation (PSK is 128 or 256 bits):

**privacy_key** = CMAC(**PSK**, AB || **key_generator** || **key_version**|| HIGH(**key_pfs**)) ||



cc BY-ND

CMAC(**PSK**, CD || **key_generator** || **key_version**|| LOW(**key_pfs**))

256-bit key derivation (PSK is 512 bits):

**privacy_key** = HMAC-SHA-512/256(**PSK**, AB || **key_generator** || **key_version**|| **key_pfs**)

- The **privacy_key** shall be 128 bits for AES-128 and 256 bits for AES-256. It is an Octet String in binary form.

- The **PSK** shall be 128 bits for AES-128 and may be 128 bits, 256 bits or 512 bits for AES-256. With a 128-bit PSK, CMAC(PSK, M) shall use the AES-128 block cipher. With a 256-bit PSK, CMAC(PSK, M) shall use the AES-256 block cipher. If a 128-bit PSK is used for deriving a 256-bit **privacy_key**, the CMAC(PSK, M) shall use the AES-128 block cipher. With a 512-bit PSK, HMAC-SHA-512/256(PSK, M) shall use the SHA-512/256 hash function.

  The **PSK** shall be provided to a Sender/Receiver by an administrator and identified by a **key_id**. The **key_id** is provided by a Sender and obtained by a Receiver using an SDP transport file and/or NMOS transport parameters. The **PSK** is an Octet String in binary form.

- The **key_generator** shall be 128 bits. It shall be provided by a Sender and obtained by a Receiver using an SDP transport file and/or NMOS transport parameters. It is an Octet String in binary form.

- The **key_version** shall be 32 bits. It shall be provided by a Sender and obtained by a Receiver using an SDP transport file and/or NMOS transport parameters. It may also be transmitted/received along with the ciphered content by a Sender/Receiver configured for in-band dynamic changes of the encryption key. The **key_version** is an Octet String in binary form.

- The **key_pfs** shall be a shared secret value calculated by each peer executing an ECDH protocol from the peer public key. For curves “secp256r1” (NIST P-256), “25519” (NIST Curve25519), “448” (NIST Curve448) and “secp521r1” (NIST P-521) the secret value has 255, 256, 448 and 521 bits respectively. The key_pfs value corresponds in size and value to the Z value as specified in NIST.SP.800-56aRev3 section 5.7.1.2. A Controller shall exchange the public keys at the activation of the Sender and Receiver. A peer public key shall be ephemeral and unique to a given activation of a Sender/Receiver. At boot / reset / init time or when a Sender/Receiver becomes inactive, a Sender/Receiver shall generate a new ECDH private/public keys pair. When using redundancy, the legs shall use the same ECDH private/public keys pair. When an ECDH mode of operation is not



BY-ND

used, the **key_pfs** value shall be an empty Octet String. It is an Octet String in binary form.

*Note*: `The standard Montgomery X-coordinate encoding used for curves 25519 and 448 uses little-endian ordering of the bytes. To keep consistency among the curves, the bytes of such encoding are reversed in` **`key_pfs`** `which is in big-endian format.`

## 13 SDP transport file parameters / NMOS transport parameters

Privacy encryption parameters, summarized in Table 1, are communicated/exchanged using an SDP transport file and/or NMOS transport parameters. Additionally, the **key_version** parameter is communicated/exchanged in-band along with the ciphered content when a protocol supporting in-band dynamic key versions is used.

For transport protocols using an SDP transport file:

- A Sender shall communicate privacy encryption parameters using a **privacy** session attribute or a number of **privacy** media attributes in the privacy encrypted stream's associated SDP transport file. A Sender in an NMOS environment shall also communicate the privacy encryption parameters using the **privacy** extended NMOS transport parameters. The privacy encryption parameters **protocol**, **mode**, **iv**, **key_generator**, **key_version** and **key_id** shall all be provided by a Sender when using the SDP or NMOS mechanisms. The privacy encryption parameters **ecdh_sender_public_key**, **ecdh_receiver_public_key** and **ecdh_curve** shall all be provided by a Sender when using the NMOS mechanism. The privacy encryption parameters provided by a Sender simultaneously in the SDP transport file and the NMOS transport parameters shall be identical. The privacy encryption parameters provided by a Sender in an SDP transport file and/or the NMOS transport parameters shall not change while the Sender is active, with the exception of **key_version** that may change in an SDP transport file when a protocol supporting in-band dynamic key versions is used.

- A Receiver in an NMOS environment shall communicate the privacy encryption parameters using the **privacy** extended NMOS transport parameters. The privacy encryption parameters **protocol**, **mode**, **iv**, **key_generator**, **key_version, key_id, ecdh_sender_public_key**, **ecdh_receiver_public_key** and **ecdh_curve** shall all be provided by a Receiver using the NMOS mechanism. The privacy encryption parameters provided by a Receiver in the NMOS transport parameters shall not change while the Receiver is active, with the exception of **key_version** that may change in an SDP transport file when a protocol supporting in-band dynamic key versions is used.



BY-ND

- When using redundancy, as in ST 2022-7, the legs shall use the same privacy encryption parameters.

- A Controller in an NMOS environment shall provide the privacy encryption parameters used by a Sender to a number of Receivers using the SDP transport file or NMOS transports parameters mechanisms.

For transport protocols that are not using an SDP transport file:

- A Sender/Receiver shall communicate the privacy encryption parameters using the **privacy** extended NMOS transport parameters. The privacy encryption parameters **protocol**, **mode**, **iv**, **key_generator**, **key_version, key_id, ecdh_sender_public_key**, **ecdh_receiver_public_key** and **ecdh_curve** shall all be provided using the NMOS mechanism. The privacy encryption parameters provided by a Sender/Receiver in the NMOS transport parameters shall not change while the Sender/Receiver is active, with the exception of **key_version** that may change in an SDP transport file when a protocol supporting in-band dynamic key versions is used.

- When using redundancy, as in ST 2022-7, the legs shall use the same privacy encryption parameters.

- A Controller in an NMOS environment shall provide the privacy encryption parameters used by a Sender to a number of Receivers using the NMOS transports parameters mechanism.

The privacy attribute of an SDP transport file shall be formatted as a follow with a semicolon and an optional space separating the parameters and a CRLF to terminate the attribute line. There shall be no semicolon after the last parameter. The placeholders <protocol>, <mode>, <iv>, <key_generator>, <key_version> and <key_id> shall be replaced with the actual value of the parameter, with Octet String represented in hexadecimal notation without spaces.

a=privacy:**protocol**=<protocol>; **mode**=<mode>; **iv**=<iv>; **key_generator**=<key_generator>;

**key_version**=<key_version>; **key_id**=<key_id> CRLF

- **protocol:** this parameter defines the privacy encryption protocol adaptation being used. It represents various aspects of the privacy encrypted stream such as the transport protocol, the encryption behavior (packet layout, encrypted sections, and authenticated sections), the support for in-band dynamic key versions, etc. It shall be a string associated with a **protocol** as documented in one of the protocol adaptations.



DRAFT
(cc) BY-ND

The **protocol** value shall not change while a Sender/Receiver is active. A Sender/Receiver shall become inactive in order to change the **protocol**.

The NULL **protocol** may be used in NMOS transport parameters to indicate that privacy encryption is not available / disabled. It shall not be possible through NMOS transport parameters to disable encryption on a Sender/Receiver that is configured to perform privacy encryption. A Sender/Receiver that does not perform privacy encryption may indicate NULL as the actual protocol adaptation in its NMOS transport parameters.

The NULL **protocol** shall not be used in an SDP transport file. Instead, the privacy attribute shall be omitted.

A Sender/Receiver may support multiple protocols**.** It may be configured to use one of such protocols through the NMOS transport parameters or through a vendor-specific mechanism. Only a vendor-specific configuration mechanism shall be able to configure a Sender/Receiver to disable privacy encryption.

- **mode**: this parameter defines the encryption, authentication and key derivation functions being used. It represents various aspects of the privacy encrypted stream such as the use of a 128 or 256 bit key, the use of authentication, the use of peer-to-peer ECDH in key derivation, etc. It shall be a string associated with a **mode** as documented in one of the protocol adaptations.

  Senders and Receivers shall support operating in AES-128-CTR **mode**. The mode of operation shall not change while a Sender or Receiver is active. A Sender or Receiver shall become inactive in order to change the **mode** of operation.

  The NULL **mode** may be used in NMOS transport parameters to indicate that privacy encryption is not available / disabled. It shall not be possible to disable encryption on a Sender/Receiver that is configured to perform privacy encryption. A Sender/Receiver that does not perform privacy encryption may indicate NULL as the actual mode of operation in its NMOS transport parameters.

  The NULL **mode** shall not be used in an SDP transport file. Instead, the privacy attribute shall be omitted.

  A Sender/Receiver may support multiple modes**.** It may be configured to use one of such modes through the NMOS transport parameters or a vendor-specific mechanism. Only a vendor-specific configuration mechanism shall be able to configure a Sender/Receiver to disable privacy encryption.

- **iv**: this parameter defines the initial vector being used by the Sender at activation. It shall be a 64-bit Octet String in binary form.



BY-ND

DRAFT

The **iv** value should be randomly chosen for each stream and should be unique among all the streams encrypted by a Sender Device. Its value shall not change while the Sender is active. A Sender shall become inactive in order to change the **iv** value.

- **key_generator**: this parameter defines the key generator being used by the Sender at activation. It shall be a 128-bit Octet String in binary form.

  The **key_generator** value shall be randomly chosen by the Sender Device at boot / reset / init time. It may be shared by a number of streams / sub-streams encrypted by a Sender Device. Its value shall not change until the next boot / reset / init of the Sender Device.

- **key_version**: this parameter defines the key version being used by the Sender at activation. It shall be a 32-bit Octet String in binary form.

  When a Sender becomes active, it selects a **key_version** value that shall remain constant during the activation of the Sender unless a protocol supporting in-band dynamic key versions is used, in which case the selected **key_version** value may increment by 1 modulo $2^{32}$ to change the associated **privacy_key** during the activation.

  The **key_version** value provided by a Sender in an SDP transport file and/or NMOS transport parameters shall not change while the Sender is active. If a protocol supporting in-band dynamic key versions is used, the **key_version** value provided in the SDP transport file and/or NMOS transport parameters shall correspond to the value at the activation of the Sender.

  The **key_version** value transmitted in-band along with the ciphered content may change while a Sender is active.

  The **key_version** may be shared by a number of streams / sub-streams encrypted by a Sender Device.

- **key_id**: this parameter defines the key identifier being used by the Sender at activation. It shall be a 64-bit Octet String in binary form.

  The **key_id** value shall be associated with a Pre-Shared Key (**PSK**). A Sender shall provide the **key_id** of the **PSK** from which derives the stream's encryption key. A Receiver shall select a **PSK** from the **key_id** to derive the stream's decryption key. Its value shall not change while the Sender is active. A Sender shall become inactive in order to change the **key_id**.

  The **key_id** value associated with a Pre-Shared Key (**PSK**) shall be globally unique among all the Sender/Receiver Devices of a network under a given administrative authority. It is an administrative responsibility to ensure such uniqueness. Administrators


CC BY-ND
DRAFT

may maintain a central registry of the PSK and their associated **key_id** or other techniques to comply with the globally unique requirement.

A Receiver may allow the use of multiple **key_id** values**.** It may be configured to use one such **key_id** value through the SDP transport file and/or NMOS transport parameters. The constraints associated with the ext_privacy_key_id parameter of the Receiver should list all the **key_id** values allowed by the Receiver.

A Sender shall use a single **key_id** value. The constraints associated with the ext_privacy_key_id NMOS transport parameter of the Sender shall list the one **key_id** value associated with the Sender.

The NMOS privacy transport parameters shall be named according to the NMOS rules and start with the prefix "ext_". Each transport parameter shall have an associated constraint describing the values supported/allowed for the parameter. A constraint allowing a single value shall describe a read-only parameter that can only be programmed to the constraint value. A Sender/Receiver shall not allow a Controller to set a transport parameter to a value that is not allowed by the associated parameter constraints.

The actual value of the NMOS transport parameters and constraints shall be a string, with Octet String represented in hexadecimal notation without spaces.

| Transport Parameter Name | Type | SDP Name | Sender | Receiver |
|---|---|---|---|---|
| ext_privacy_protocol | string | protocol | r/w | r/w |
| ext_privacy_mode | string | mode | r/w | r/w |
| ext_privacy_iv | string | iv | read-only | r/w |
| ext_privacy_key_generator | string | key_generator | read-only | r/w |
| ext_privacy_key_version | string | key_version | read-only | r/w |
| ext_privacy_key_id | string | key_id | read-only | r/w |
| ext_privacy_ecdh_sender_public_key | string | - | read-only | r/w |
| ext_privacy_ecdh_receiver_public_key | string | - | r/w | read-only |
| ext_privacy_ecdh_curve | string | - | r/w | r/w |

Table 1- Transport Parameters

A device supporting the Perfect Forward Secrecy ECDH mode shall implement the **ext_privacy_ecdh_curve**, **ext_privacy_ecdh_sender_public_key** and



BY-ND

**ext_privacy_ecdh_receiver_public_key** NMOS transport parameters. This mode of operations should be used only for peer-to-peer Sender, Receiver configurations. The ECDH parameters are used by the PEP key derivation function when the **mode** of operation enables ECDH (having an "ECDH_" prefix) as indicated by the **ext_privacy_mode** NMOS transport parameter and the **"a=privacy" mode** attribute of the SDP transport file.

- **ext_privacy_ecdh_curve:** this parameter defines the elliptic curve being used by the Sender/Receiver. It shall be a string associated with an **ecdh_curve** among the following: “secp256r1”, “25519”, “448”, “secp521r1” or "NULL" if the Perfect Forward Secrecy ECDH mode is not available / supported.

  The constraint on **ext_privacy_ecdh_curve** shall indicate which elliptic curves are supported by the Sender. A constraint may indicate that no value is supported (empty enum) or that only the "NULL" value is supported to express that the Perfect Forward Secrecy ECDH mode is not supported by the Sender or Receiver. When Perfect Forward Secrecy is not supported there shall be no **mode** of operation with an "ECDH_" prefix listed in the constraints on **ext_privacy_mode.**

  The **ext_privacy_ecdh_curve** parameter shall not change while a Sender or Receiver is active. A Sender or Receiver shall become inactive in order to change the **ecdh_curve.**

- **ext_privacy_ecdh_sender_public_key:** this parameter defines the ECDH public key of the Sender. It shall be an Octet String in binary form of the corresponding number of bits for the actual elliptic curve. The 00 Octet String should be used when the **ecdh_curve** is "NULL" or the parameter is not yet defined on a Receiver. A Sender shall generate a new **ext_privacy_ecdh_sender_public_key** value randomly at initialization, when it becomes inactive either implicitly through some internal process or explicitly through an external deactivation request. The generation of the public key shall depend only on the **ecdh_curve** being used.

  *Note*: `A Sender may become inactive automatically without an explicit request from a Controller.`

  For curves secp256r1 and secp521r1 the public key shall be an Octet String in uncompressed form as per SEC 1, Version 2.0, Section 2.3.3 (prefix byte) while for curves 25519 or 448 the public key shall be a plain Octet String in uncompressed form (no prefix byte).

  *Note*: `The standard Montgomery X-coordinate encoding used for curves 25519 and 448 uses a little-endian ordering of the bytes. To keep consistency among the curves the bytes of such encoding are reversed in` **`ext_privacy_ecdh_sender_public_key`** `and`



CC BY-ND

`ext_privacy_ecdh_receiver_public_key` which are in big-endian format.

For a Sender, the constraints on the transport parameter **ext_privacy_ecdh_sender_public_key** shall allow only one value, indicating that it cannot be changed by a Controller. The value shall correspond to the ECDH public key of the Sender that a Controller shall provide to the peer Receiver. A constraint may indicate that no value is supported (empty enum), or that only the "00" value is supported, to express that the Perfect Forward Secrecy ECDH mode is not supported by the Sender.

For a Receiver, the constraints on the transport parameter **ext_privacy_ecdh_sender_public_key** should allow any Octet String of the corresponding number of bits for the actual elliptic curve. A constraint may indicate that no value is supported (empty enum) or that only the 00 value is supported to express that the Perfect Forward Secrecy ECDH mode is not supported by the Receiver.

The **ext_privacy_ecdh_sender_public_key** parameter shall not change while a Sender or Receiver is active. A Sender or Receiver shall become inactive in order to change the **ext_privacy_ecdh_sender_public_key.**

- **ext_privacy_ecdh_receiver_public_key:** this parameter defines the ECDH public key of the Receiver. It shall be an Octet String in binary form of the corresponding number of bits for the actual elliptic curve. The 00 Octet String should be used when the **ecdh_curve** is "NULL", or the parameter is not yet defined on a Sender. A Receiver shall generate a new **ext_privacy_ecdh_receiver_public_key** value randomly at initialization, when it becomes inactive either implicitly through some internal process, or explicitly through an external deactivation request. The generation of the public key shall depend only on the **ecdh_curve** being used.

  For curves secp256r1 and secp521r1 the public key shall be an Octet String in uncompressed form as per SEC 1, Version 2.0, Section 2.3.3 (prefix byte), while for curves 25519 or 448, the public key shall be a plain Octet String in uncompressed form (no prefix byte).

  ***Note***: `The standard Montgomery X-coordinate encoding used for curves 25519 and 448 uses a little-endian ordering of the bytes. To keep consistency among the curves, the bytes of such encoding are reversed in` **`ext_privacy_ecdh_sender_public_key`** `and` **`ext_privacy_ecdh_receiver_public_key`**`, which are in big-endian format.`

  For a Receiver, the constraints on the transport parameter **ext_privacy_ecdh_receiver_public_key** shall allow only one value, indicating that it



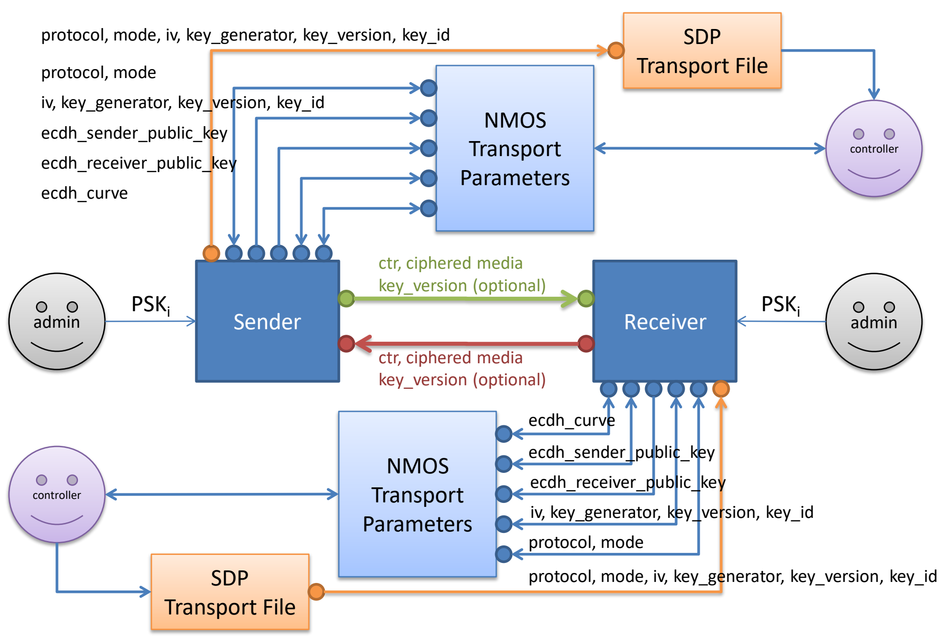
DRAFT
(cc) BY-ND

cannot be changed by a Controller. The value shall correspond to the ECDH public key of the Receiver that a Controller shall provide to the peer Sender. A constraint may indicate that no value is supported (empty enum), or that only the 00 value is supported, to express that the Perfect Forward Secrecy ECDH mode is not supported by the Receiver.

For a Sender, the constraints on the transport parameter **ext_privacy_ecdh_receiver_public_key** should allow any Octet String of the corresponding number of bits for the actual elliptic curve. A constraint may indicate that no value is supported (empty enum), or that only the 00 value is supported, to express that the Perfect Forward Secrecy ECDH mode is not supported by the Sender.

The **ext_privacy_ecdh_receiver_public_key** parameter shall not change while a Sender or Receiver is active. A Sender or Receiver shall become inactive in order to change the **ext_privacy_ecdh_sender_public_key.**

A Controller participating in a Privacy Perfect Forward Secrecy ECDH exchange should exchange the **ext_privacy_ecdh_sender_public_key** transport parameter of a Sender and of **ext_privacy_ecdh_receiver_public_key** of a Receiver prior to the activation of the Sender/Receiver. The **ext_privacy_ecdh_curve** value of transport parameters may affect the generation of those values. A controller should read the **ext_privacy_ecdh_sender_public_key** transport parameter of a Sender and of **ext_privacy_ecdh_receiver_public_key** of a Receiver after completing the programming of the **ext_privacy_ecdh_curve** transport parameter and explicitly deactivating the Sender/Receiver.






(cc) BY-ND

Figure 2 - SDP and NMOS parameters relationship

## 14 Multiplexed streams and Bidirectional streams

The **iv'** parameter, which is used by the Privacy Cipher to encrypt/decrypt a stream/sub-stream, derives from **iv** and shall be equal to the **iv** value (see Figure 3) unless multiplexed sub-streams (see Figure 4) or bidirectional sub-streams are used (see Figure 5). The **iv'** value should be unique among all the streams and sub-streams encrypted by a Sender/Receiver Device. Its value shall not change while the Sender/Receiver is active. A Sender/Receiver shall become inactive in order to change the **iv'** value of a stream or sub-stream.

The actual value of the quartet (**iv'**, **key_generator**, **key_version**, **key_id**) shall be unique among all the streams and sub-streams encrypted by the Sender/Receiver Device. A Sender/Receiver Device may enforce this rule statically by ensuring that the **iv'** values are unique among all the streams and sub-streams encrypted by the device.

For multiplexed streams where each sub-stream is encrypted independently, the effective **iv'** value for a given sub-stream shall be obtained by adding modulo $2^{64}$, the sub-stream id in the range [0, 1023], with the base **iv** value (see Figure 4). The sub-stream id is a concept understood by the Sender and Receiver of such a multiplexed stream and is adaptation specific. This Technical Recommendation requires only that such sub-streams be allocated an identifier in the range [0, 1023] thus restricting such a multiplexed stream to a maximum of 1024 sub-streams.

For bidirectional and optionally multiplexed streams, the sub-streams in each direction shall be encrypted independently, and the effective **iv'** value for a given sub-stream shall be obtained by adding modulo $2^{64}$, the sub-stream id in the range [0, 1023] with the base **iv** value (see Figure 5). The sub-stream id shall be an even number for the Sender to Receiver direction and an odd number for the Receiver to Sender direction. The sub-stream id is a concept understood by the Sender and Receiver of such a bidirectional stream and is adaptation specific. This Technical Recommendation requires only that such sub-streams be allocated an identifier in the range [0, 1023] with proper odd/even allocation thus restricting such a bidirectional stream to a maximum of 512 bidirectional sub-streams.

DRAFT



BY-ND

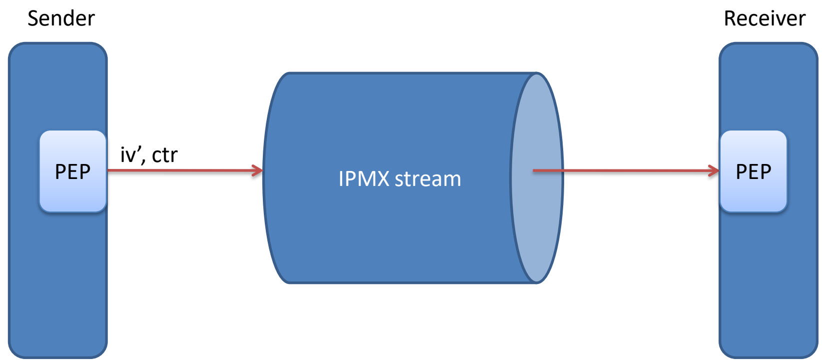



Figure 3 - Single unidirectional stream

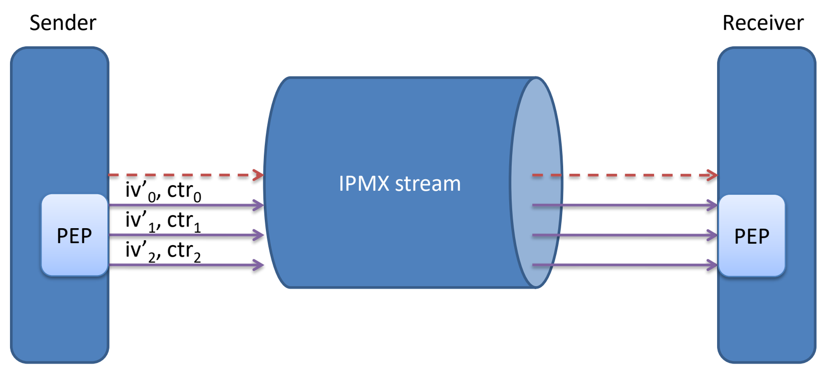



Figure 4 - Multiplexed unidirectional streams



cc BY-ND

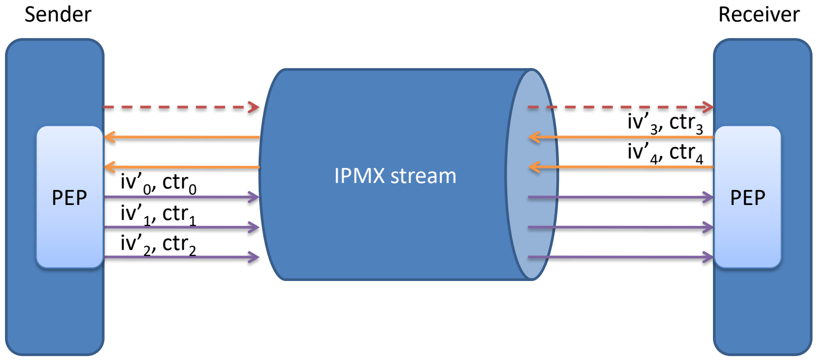



Figure 5 - Multiplexed bidirectional streams

## 15 Privacy Cipher

The privacy cipher shall be based on the AES-128 or AES-256 block cipher operating in CTR or GCM mode (NIST.FIPS.197, NIST.SP.800-38a, NIST.SP.800-38d)

cipher_content$_i$ = CIPH$_{\mathbf{key}}$(**iv'_ctr**) XOR plain_content$_i$

- The **key** shall be 128 bits for AES-128 and 256 bits for AES-256. It shall be generated from a Pre-Shared Key (**PSK**) and other values obtained from separate channels. It is an Octet String in binary form.

- The **iv'_ctr** shall be a 128-bit Octet String in binary form. It shall be generated from a number of associated values depending on the transport protocol adaptation. Usually, such values comprise an initial vector **iv'** and a sequence counter **ctr**. The **iv'_ctr** value shall not be used more than once with a given **key**.

- cipher_content$_i$ is a 128-bit Octet String of binary data from of a stream or sub-stream

- plain_content$_i$ is a 128-bit Octet String of binary data from of a stream or sub-stream

The effective value of iv'_ctr used by the CIPH$_{key}$ function may differ according to the transport protocol adaptation.

*Note*: Depending on the transport protocol adaptation, the size and combination of **iv'** and **ctr** values into **iv'_ctr** may be slightly different than what is described in this Technical Recommendation for



BY-ND

```
the RTP transport protocol, while still being compliant with the AES
cipher in CTR/GCM mode specification. This Technical Recommendation
assumes full control over the encryption process when using the RTP
transport protocol, while still allowing a transport protocol specific
encryption process based on the privacy_key defined in this document.
```

The sections of the stream that are encrypted and/or authenticated are protocol and adaptation specific.

Modes of operation using a message authentication code (MAC) of some sections of the stream, excluding GCM, shall operate in a mac-then-encrypt scheme and store the encrypted MAC as the last N bytes of the encrypted payload (the MAC is encrypted along with the payload). The mode of operation shall dictate the size (N), in bytes, of the MAC and the use of AAD. The MAC shall be calculated over the payload data and optionally it may also be calculated over some additional authenticated data (AAD). It then applies to the payload data that is to be encrypted and optionally to some data that is not to be encrypted.

The GCM mode operates in an encrypt-then-mac scheme and stores the MAC in clear as the last N bytes of the encrypted payload (the MAC is not encrypted). The mode of operation shall dictate the size (N), in bytes, of the MAC and the use of AAD. The MAC shall be calculated over the encrypted payload data and optionally it may also be calculated over some additional authenticated data (AAD). It then applies to the encrypted payload data and optionally to some data that is not to be encrypted.

The truncated MAC shall be comprised of the most significant bit of the MAC, as required by NIST SP 800-38b and NIST SP 800-38d.

## 16 Sender / Receiver Model (Informative)

Figure 6 and Figure 7 describe a model of a Sender and a Receiver implementing the Privacy Encryption Protocol.

Without using dynamic changes of the **key_version:**

- This is the simplest configuration. The PEP parameters are static for the duration of active state of a Sender/Receiver and completely under the control of the Sender Device.
- One or many sub-streams may be encrypted/decrypted in parallel, possibly using the same **key**. Each sub-stream has a dedicated **iv'** and **ctr** parameter.
- There are as many KDFe and KDFd modules as there are distinct **keys**.
- As the **key_version** is not allowed to change dynamically there is only one **key_version** state shared by all the KDFe and KDFd modules.
- The Sender Device possibly shares the same **key_id**, **key_version** and **key_generator** with multiple Senders.



BY-ND

- In this configuration the Sender and Receiver are deactivated to change the PEP parameters. It is not possible to change the encryption key without stopping the Sender and Receiver.

Using dynamic changes of the **key_version:**

- This is the most flexible configuration. The PEP parameters, except **key_version,** are static for the duration of active state of a Sender/Receiver and completely under the control of the Sender Device. The **key_version** is dynamic, and is communicated in-band along with the ciphered content. It is completely under the control of the device performing the encryption (Sender or Receiver).
- In this configuration the KDFd modules are not connected to the **key_version** but are connected to the dynamic_key_version instead.
- One or many sub-streams may be encrypted/decrypted in parallel, possibly using the same **key**. Each sub-stream has a dedicated **iv'** and **ctr** parameter.
- There are as many KDFe modules as there are distinct **keys**.
- There are as many KDFd modules as there are sub-streams. A number of sub-streams could possibly be using the same **key**, but as the decryption module does not have a priori knowledge of such information, it assumes the worst case of all sub-streams using a different **key**.
- As the **key_version** is allowed to change dynamically there are as many **key_version** states as there are KDFe modules. Note that the number of KDFe modules of a Sender and Receiver will differ and is totally under the control of the Sender and Receiver.
- There are as many dynamic_key_version states transmitted in-band as there are sub-streams encrypted in parallel, each receiving the **key_version** parameter associated with the key used for encryption.
- There are as many dynamic_key_version states received in-band as there are sub-streams decrypted in parallel, each receiving a dynamic **key_version** parameter associated with the key used for decryption.
- The Sender Device possibly shares the same **key_id**, **key_version** and **key_generator** with multiple Senders.
- The Receiver Device possibly shares the same **key_id**, **key_version** and **key_generator** with multiple Receivers.
- In this configuration the Sender and Receiver are deactivated to change all the PEP parameters but **key_version**. It is possible to change the encryption/decryption key without stopping the Sender and Receiver using dynamic **key_version**.

DRAFT



CC BY-ND

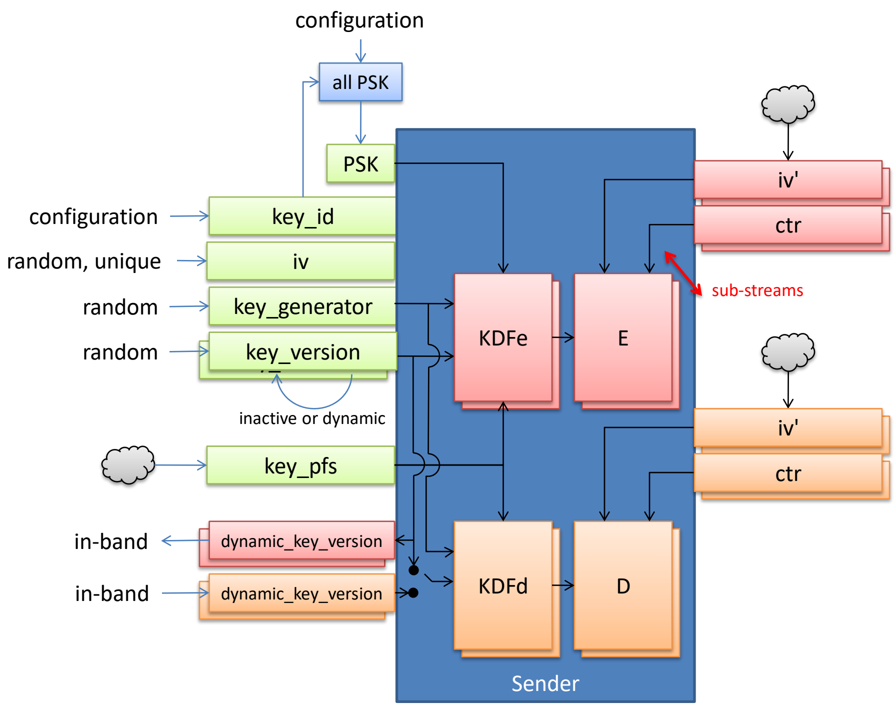



Figure 6 - Sender model

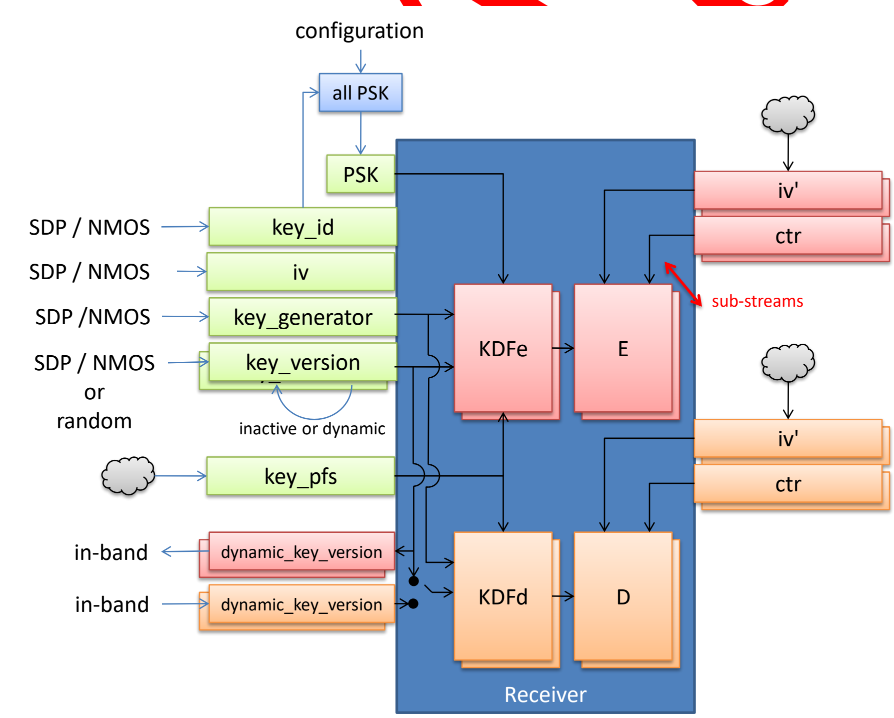



Figure 7 - Receiver model



BY-ND

## 17 Key distribution

A set of **PSK** values shall be programmed in both Sender and Receiver Devices by means out of the scope of this Technical Recommendation. Only devices sharing the same secret PSK value and size shall be allowed to successfully exchange privacy encrypted media content when the ECDH **mode** is not used. Only devices sharing the same secret PSK value and size and ECDH secret value shall be allowed to successfully exchange privacy encrypted media content when the ECDH **mode** is used.

A Pre-Shared Key has a value (the secret itself) and a size (the number of bits of the associated PSK). This Technical Recommendation describes the requirements for PSK of 128, 256 and 512 bits. The programming of the PSK in the devices shall target one such size. The default target size shall be 128 bits and shall be supported by all devices. The programming interface should clearly indicate the target number of bits for PSK of 256 and 512 bits. If a device does not support PSK of 256 or 512 bits it shall clearly indicate to an administrator using such programming interface that such PSK target sizes are not supported by the device.

Note: `When an administrator associates a secret key of value` *abcd* `with a Sender it also associates a size, for example 128 bits. When sharing such` *abcd* `secret key with some Receivers, the administrator must not only provide the` *abcd* `value but also the associated size to the Receivers.`

A given PSK is associated with a unique **key_id**. For security purposes there should be, at most, one **PSK** value associated with a given **key_id**. A cryptographically secure random number generator should be used to generate a **PSK** value and provide a high level of confidence in the uniqueness of the **PSK** value associated with a given **key_id**. A deployment with extreme security requirements may further ensure that no one **PSK** value is associated with more than one **key_id**.

A number of **PSK** may be programmed into a Sender Device, each key having an associated **key_id** and an associated set of streams (Senders). In its simplest configuration a Sender Device uses one **PSK** for all the streams it transmits over the network. In a complex environment, a Sender Device may have a **PSK** associated with a specific set of streams (Senders) in order to control the accessibility of such streams by Receivers.

A number of **PSK** may be programmed into a Receiver Device, each key having an associated **key_id**. Programming a PSK in a Receiver Device implies allowing such Receiver Device to access the content of the streams encrypted from such PSK.

It shall be assumed that once a **PSK** associated with a given **key_id** is given to a Receiver Device, it is able to access a stream associated with this **PSK** forever. To prevent further access to a stream by such Receiver Device, the Sender Device shall stop using such **key_id**.

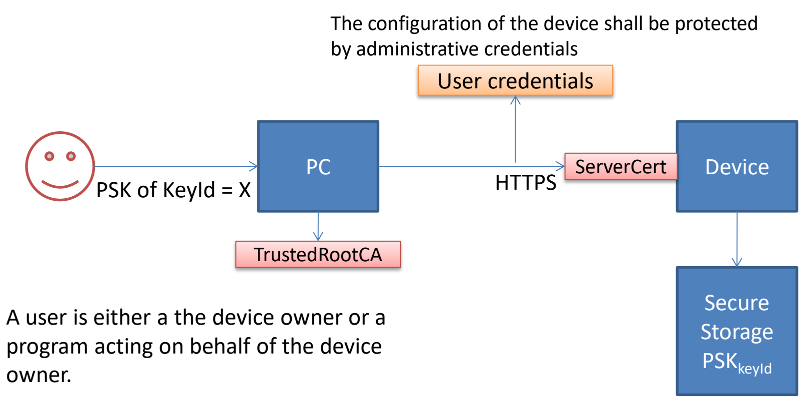


BY-ND

The SDP transport file and/or NMOS transport parameters provide the **key_id** associated with the stream the Receiver is subscribing, such that the Receiver can look up the proper **PSK** for accessing the content of the stream.

```
                key_id
                  |
Stream ---- SDP Transport File ---- Sender
        NMOS Transport Parameters
```

## 18.1 Key distribution through HTTPS (informative)




Sender device

| KeyId | Senders | PSK | Size |
|---|---|---|---|
| 1 | AV | ************ | 128 |
| 2 | Data | ************ | 128 |
| 3 | AVSec | ************ | 256 |

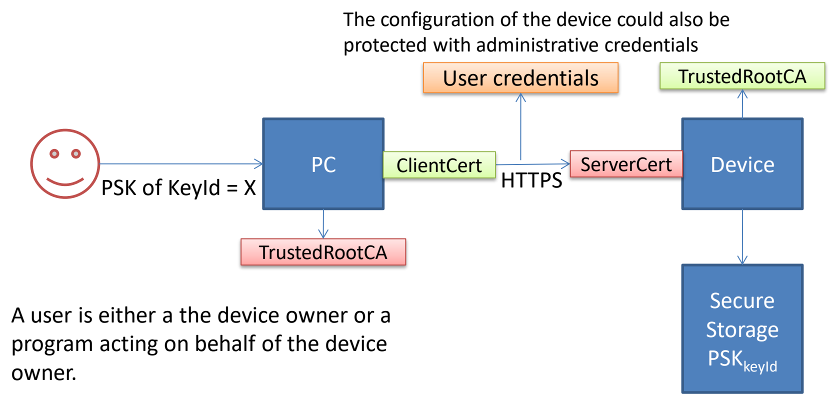
Receiver device

| KeyId | PSK | Size |
|---|---|---|
| 1 | ************ | 128 |
| 3 | ************ | 256 |
| | | |

Figure 8 - Configuring PSKs using HTTPS Server certificates



cc BY-ND




Sender device

| KeyId | Senders | PSK | Size |
|---|---|---|---|
| 1 | AV | ************ | 128 |
| 2 | Data | ************ | 128 |
| 3 | AVSec | ************ | 256 |

Receiver device

| KeyId | PSK | Size |
|---|---|---|
| 1 | ************ | 128 |
| 3 | ************ | 256 |
| | | |

Figure 9 - Configuring PSKs using HTTPS Client-Server certificates

DRAFT

## 18 Safety

The privacy encryption protocol uses various channels for the transport of key material. For instance, the Pre-Shared Key (PSK) is programmed into the device by means out of the scope of this Technical Recommendation. It is expected to be performed in a secure way through a device proprietary configuration interface. For example, such interface may be using a secure connection such as HTTPS as illustrated in Figure 8 and Figure 9, with proper verification of the certificates and using administrator credentials.

The SDP transport file and/or the NMOS transport parameters provide the **key_id**, **iv** and **key_generator** values along with the **key_version** associated with the **key_generator**. All these values are not secret values but it is expected that a secure deployment of Sender and Receiver Devices would use HTTPS to exchange NMOS messages and SDP transport files, adding an extra layer of security over the **privacy** attributes of the SDP transport file and NMOS transport parameters. Further security may be achieved using IS-10 to protect the exchange of SDP transport file and NMOS transport parameters. The value of the **key_generator** attribute of the SDP transport file and/or NMOS transport parameters is subject to an initial random shuffle and will change at every boot / reset / init of the Sender Device.

The in-band dynamic **key_version** received along with the ciphered content stream, may be protected using a privacy encryption **protocol** and **mode** of operation providing authentication of the **key_version** value.



CC BY-ND

When an in-band dynamic **key_version** is received along with the ciphered content stream, a Sender/Receiver should verify that it is making forward progress or has not changed. The incoming **key_version** value shall be larger or equal to the last **key_version** value received from the ciphered content stream, taking into account the $2^{32}$ wrap around and the monotonic incremental nature of the value, and that the effective value cannot be farther than $2^{31}$ units from the initial or previous one. After a Receiver is subscribed to a media stream, the first **key_version** value received along with the ciphered content stream by the subscribed Receiver or by the Sender cannot be verified for making forward progress.

The in-band dynamic **key_version** protocol (i.e protocol name ending with _KV) should be used with a **mode** of operation providing authentication.

The **ctr** received along with the ciphered content stream, may be protected using a privacy encryption **protocol** and **mode** of operation providing authentication of the **ctr** value.

A Sender/Receiver should verify that the **ctr** value received along with the ciphered content stream is making forward progress. The incoming **ctr** value shall be larger to the last **ctr** value received from the ciphered content stream, taking into account the $2^{64}$ wrap around and the monotonic incremental nature of the value. After a Receiver is subscribed to a media stream, the first **ctr** value received along with the ciphered content stream by the subscribed Receiver or by the Sender cannot be verified for making forward progress. The first **ctr** value received may not be 0 as the Receiver may be subscribing to a media stream after the initial packet has been transmitted. The mechanism used by a Sender to know that a Receiver subscribed to the content stream is adaptation dependent.

This Technical Recommendation ensures that the AES-CTR mode cipher is used safely, providing requirements and techniques to guarantee the uniqueness of the **key** and **iv'_ctr** values of the privacy cipher. See the section 15 for more details.

The use of a 64-bit MAC is within the security guarantees of AES-CMAC as per NIST.SP.800-38b section A.2 “For most applications, a value for Tlen that is at least 64 should provide sufficient protection against guessing attacks.”

The Perfect Forward Secrecy ECDH mode allows further security for peer-to-peer configurations where only the associated Sender and Receiver can decrypt the content as the encryption key derives from both the ECDH, and the privacy encryption protocol schemes. In this mode the knowledge of the PSK is used to ensure that the peers are legitimate devices. Getting the shared ECDH secret key is not enough for decrypting the stream. The **key_pfs** is one of many pieces of information used for deriving the **privacy_key**.

DRAFT



CC BY-ND

# 19 Test vectors for key derivation (informative section)

This section provides test vectors for verifying the implementation of the Privacy Encryption Protocol.

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "ECDH_AES-128-CTR"
"ext_privacy_iv": "a48a7f235b37ebd6"
"ext_privacy_key_generator": "2a4ab04bd61219d37a91abf6f94ab124"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "a7938740"
"ext_privacy_ecdh_sender_public_key":
"04b575bec15a1c92e822a0b71175f7fed4d1f3f1a9518d38a9314e7882bd82b6"
"ext_privacy_ecdh_receiver_public_key":
"50f4cf59ab320e83c802c1fae250086943f9df13c605e843434b10098c630ebc"
"ext_privacy_ecdh_curve": "25519"
Privacy keyId 0001020304050607
Privacy keyGenerator = 2a4ab04bd61219d37a91abf6f94ab124
Privacy keyVersion = a7938740
Privacy PSK = 000102030405060708090a0b0c0d0e0f
Privacy PFS =
218f8b81501ea437e0bc2c21a8e9af2be7bee3b1c553f9ccaaf40e3dc19374c6
Privacy KEY = dee53f79ac29628644d01783b5b3c0b7
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "ECDH_AES-128-CTR"
"ext_privacy_iv": "6ca953833cba30fb"
"ext_privacy_key_generator": "2edf9023a68fb83c5d1f018d7cd3783e"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "cc2301ed"
"ext_privacy_ecdh_sender_public_key":
"04156ce3d53ba3e196377151b44ed0c7ad97467299a3a16bc6ae581f2369705f3c4dbe4c355
de00fecd20f3ac432aa87c1c5608c5addbef709dc5627063fbc1b99"
"ext_privacy_ecdh_receiver_public_key":
"0437a606a2a808d01f96d74404b5b8bfc85f2eb244c1fcf748e1f60b9b898cc3043cc92d81f
774071e365f67466083551f502616cfdd4590aa0fcb2abfa0aa58bd"
"ext_privacy_ecdh_curve": "secp256r1"
Privacy keyId 0001020304050607
Privacy keyGenerator = 2edf9023a68fb83c5d1f018d7cd3783e
Privacy keyVersion = cc2301ed
Privacy PSK = 000102030405060708090a0b0c0d0e0f
Privacy PFS =
dcf9d6b750d8c51419127f6e9ef9c91199bb99237d28e4054a6486f190b403d3
Privacy KEY = 12d376fa12f933780b1a68b9ebdb4187
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "ECDH_AES-128-CTR"
"ext_privacy_iv": "58461a11c3fc1337"
"ext_privacy_key_generator": "a7ebcd7bef2b32abc008e1d0d0c777a0"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "5c436e9d"
"ext_privacy_ecdh_sender_public_key":
"0400d4c60c971b147796b35074a2795c82721f00aa5c76e487bcd963d07d5a32ad12f4f4285
426ff563b729b8146558e671a4893f9f7b544a00b7f75d75c22657221f0004e4d41ad92c32c3
```

DRAFT



BY-ND

| |
|---|
| b86421eba93a491339e67bfa22120497e99220625400f9efbe63db77a71f77e95c5efb8dbef9<br>a1d17a20bd5e7954d55c7ea8000472a9b28debe"<br>"ext_privacy_ecdh_receiver_public_key":<br>"04008788c452f0c25a34f25df671af02b35e18f61e8e2f4aed3c275556d9ff38e699f3c963a<br>f7046d775fcb0019318ad7aa2e5485f91011f0f246e2319d1c5bb497ec700545de08600571f1<br>6aed75cf2dd230c9db5fac67accc56eb3c6f97e1d49781692ca79524d719f257b02ce96df5d9<br>062ed7c3a0d7ac26411940b323183f565fe5bea"<br>"ext_privacy_ecdh_curve": "secp521r1"<br>Privacy keyId 0001020304050607<br>Privacy keyGenerator = a7ebcd7bef2b32abc008e1d0d0c777a0<br>Privacy keyVersion = 5c436e9d<br>Privacy PSK = 000102030405060708090a0b0c0d0e0f<br>Privacy PFS =<br>015df637be34bb2edc8f493d3cdbb4ba05371b894cf20adf899ad5a1cbbba4c26acaf1342b37<br>66e5f686b00537d810372fb840b28c4a3587bba07cf12721cff37846<br>Privacy KEY = 56afadf373fccef80e70a755fe0a1588 |
| "ext_privacy_protocol": "RTP"<br>"ext_privacy_mode": "ECDH_AES-256-CTR"<br>"ext_privacy_iv": "85cce5edc2dbbf91"<br>"ext_privacy_key_generator": "a208336568863d5cf6ee704837340d79"<br>"ext_privacy_key_id": "0001020304050607"<br>"ext_privacy_key_version": "84f03939"<br>"ext_privacy_ecdh_sender_public_key":<br>"062acdfb142ef1944a26b934030f28e5715f0aef6f3d674efc4c3949ed2e3913"<br>"ext_privacy_ecdh_receiver_public_key":<br>"07356a625c072988b1cb69369861f0d52cf56c7f80f2cded3c3155d58c2fa44d"<br>"ext_privacy_ecdh_curve": "25519"<br>Privacy keyId 0001020304050607<br>Privacy keyGenerator = a208336568863d5cf6ee704837340d79<br>Privacy keyVersion = 84f03939<br>Privacy PSK = 000102030405060708090a0b0c0d0e0f<br>Privacy PFS =<br>79a44729b1f4d9f52a4e210a5b4e776de4f511837798b88beafd5aaa41eb0700<br>Privacy KEY =<br>f78d42babb85119405b13bb1199a80bdd5557cc64a596d97abe9bf945079d81a |
| "ext_privacy_protocol": "RTP"<br>"ext_privacy_mode": "ECDH_AES-256-CTR"<br>"ext_privacy_iv": "d03c3806f952a5c0"<br>"ext_privacy_key_generator": "51fa624b4c62a2125e45424c2f185cb9"<br>"ext_privacy_key_id": "0001020304050607"<br>"ext_privacy_key_version": "2b7a8223"<br>"ext_privacy_ecdh_sender_public_key":<br>"043634e9782c0ea61112ca7d5907a99fa942e1f1ea080591558433d39a39a30cba478178347<br>03e8bd9918e39fc0704d102d277f28856931844be4dc46412c1a8fd"<br>"ext_privacy_ecdh_receiver_public_key":<br>"045ec33bda2ae2feea2931de5bc134db4d40635f85d9a29d3005f86ceac67b07db7bafadbbb<br>b7f40e9812518a909db3f77e0168fd73cad5ae903f467f0a4b18112"<br>"ext_privacy_ecdh_curve": "secp256r1"<br>Privacy keyId 0001020304050607<br>Privacy keyGenerator = 51fa624b4c62a2125e45424c2f185cb9<br>Privacy keyVersion = 2b7a8223<br>Privacy PSK = 000102030405060708090a0b0c0d0e0f |

DRAFT



CC BY-ND

```
Privacy PFS =
3e1e0e9836bd01b38a9f18fac02da9d5a545f1ca8149f076917d6f3e3a8b94eb
Privacy KEY =
a3ba0f316f10fb6866bbeb3d6841b346505a1c1f5ec3e36c626721637c0c5aaa
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "ECDH_AES-256-CTR"
"ext_privacy_iv": "b50bf96330c3a250"
"ext_privacy_key_generator": "8623b4b1e6fa7067be1f5952ad6299b8"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "2af1988d"
"ext_privacy_ecdh_sender_public_key":
"040044e3343f79a69672c5b6618abe1b0971fab4726015ad0802435519fae84998a4c337865
f734d9acca251e23b5508987afdfeb22c641a4e533707ea136d1e543df000fb33412a195894c
75298aa36152a007d87df36dd3269dd74d328416323c294b79bde17752732fd086f34ce3d2b4
1c9563f55646a86b4d7d2b12f110048681a3bee"
"ext_privacy_ecdh_receiver_public_key":
"040023d3cfca8624e5c26ef74d523a5cc0a932dad292b6bf6c0efebab0328ef823f6d3a5192
c158cdb0bd637146110021dbc68e8ff54b1f73dc9471e0c78de3bc95a7600bc2d28700a228a5
93c3e38368835d7bd5da0a148e6bf28c18d6a6559b35c59a1fdadd67f3baf6b3561a5af48365
dc4c28a8432ab48304d2ffdc25a0d567775636c"
"ext_privacy_ecdh_curve": "secp521r1"
Privacy keyId 0001020304050607
Privacy keyGenerator = 8623b4b1e6fa7067be1f5952ad6299b8
Privacy keyVersion = 2af1988d
Privacy PSK = 000102030405060708090a0b0c0d0e0f
Privacy PFS =
00c25350af2ccf296cd60e055b8d70c66a40db98eccb179103c0208700df96ba41d144abd187
5128824a659ae133e394ace2d3e898d95f8f895e96e3a4593a570cf4
Privacy KEY =
3b99a7d6eca76f53600084aec2ce920c5a73391b650b95fc285d00b6286e28d9
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "AES-128-CTR"
"ext_privacy_iv": "f86c85e76cc45e50"
"ext_privacy_key_generator": "52bbbea2b2cdc7ddbb18c23becd3c753"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "007c84b5"
"ext_privacy_ecdh_sender_public_key":
"033a90030205226c28b5da20aef1519c7df0aa28980bfbbf506e5063729955f8"
"ext_privacy_ecdh_receiver_public_key": "00"
"ext_privacy_ecdh_curve": "25519"
Privacy keyId 0001020304050607
Privacy keyGenerator = 52bbbea2b2cdc7ddbb18c23becd3c753
Privacy keyVersion = 007c84b5
Privacy PSK = 000102030405060708090a0b0c0d0e0f
Privacy PFS =
Privacy KEY = 650132d60b2700cd2aa3e25f24aa8980
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "AES-256-CTR"
"ext_privacy_iv": "f86c85e76cc45e50"
"ext_privacy_key_generator": "52bbbea2b2cdc7ddbb18c23becd3c753"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "007c84b5"
```

DRAFT



CC BY-ND

```
"ext_privacy_ecdh_sender_public_key":
"1ae3c42e4280f5080b7f6ffa01a56d2bbb71c63ede9f718f31f2f92dabacf67b"
"ext_privacy_ecdh_receiver_public_key": "00"
"ext_privacy_ecdh_curve": "25519"
Privacy keyId 0001020304050607
Privacy keyGenerator = 52bbbea2b2cdc7ddbb18c23becd3c753
Privacy keyVersion = 007c84b5
Privacy PSK = 000102030405060708090a0b0c0d0e0f
Privacy PFS =
Privacy KEY =
650132d60b2700cd2aa3e25f24aa8980cafd1d993e2e2a36640b7795579c089a
```

```
"ext_privacy_protocol": "RTP",
"ext_privacy_mode": "AES-256-CTR"
"ext_privacy_iv": "aa68f9206ddee5e9"
"ext_privacy_key_generator": "f99067d1f5f72363d3b0e009ab34c36b"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "7251c65d"
"ext_privacy_ecdh_sender_public_key":
"3af38bbbab6dc1e60b1d82680416caf2645091f7e1d191e98d434bad162fedd6"
"ext_privacy_ecdh_receiver_public_key": "00"
"ext_privacy_ecdh_curve": "25519"
Privacy keyId 0001020304050607
Privacy keyGenerator = f99067d1f5f72363d3b0e009ab34c36b
Privacy keyVersion = 7251c65d
Privacy PSK =
000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f
Privacy PFS =
Privacy KEY =
e9ceff8c8aa6aa6680c1928a5427fb71351ce3c9c507c92a9fba3bcbd65681f3
```

```
"ext_privacy_protocol": "RTP"
"ext_privacy_mode": "AES-256-CTR"
"ext_privacy_iv": "7eee1d6607035871"
"ext_privacy_key_generator": "1927a9d6914eb5579edd30712a081f84"
"ext_privacy_key_id": "0001020304050607"
"ext_privacy_key_version": "c5f4a28d"
"ext_privacy_ecdh_sender_public_key":
"044b4b973dd122d3e2571e568f9fa594009ece14724acec2802e36323d494336"
"ext_privacy_ecdh_receiver_public_key": "00"
"ext_privacy_ecdh_curve": "25519"
Privacy keyId 0001020304050607
Privacy keyGenerator = 1927a9d6914eb5579edd30712a081f84
Privacy keyVersion = c5f4a28d
Privacy PSK =
000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f000102030405
060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f
Privacy PFS =
Privacy KEY =
2e4edd15087fa6d4fef2f5c16ee0d474fec93823c12099a47d00bd5cd54d87e6
```

Table 2 - Test Vectors

DRAFT



(cc) BY-ND

## 20 RTP transport protocol adaptations

The RTP protocol adaptation shall be supported by all implementations of the Privacy Encryption Protocol using the RTP protocol. The RTP_KV protocol adaptation is optional.

For RTP, this Technical Recommendation assumes full control over the encryption process. There is no encryption/authentication of the RTCP messages.

*Note*: This Technical Recommendation does not describe privacy encryption of RTCP messages.

The **protocol** parameter shall be RTP or RTP_KV.

- A Sender / Receiver shall support the RTP **protocol**.
- A Sender using the RTP_KV **protocol** shall transmit the **key_version** along with the ciphered content in the dynamic_key_version field of the Full RTP Extension Header. When using the RTP **protocol,** the dynamic_key_version field of the Full RTP Extension Header shall be set to 0 unless the extension header field is being used for other purposes outside the scope of this Technical Recommendation. A Receiver using the RTP **protocol** shall ignore the dynamic_key_version field of the Full RTP Extension Header. When using the RTP_KV **protocol** the Receiver shall monitor it.

The **mode** parameter shall be one of AES-128-CTR, AES-256-CTR, AES-128-CTR_CMAC-64, AES-256-CTR_CMAC-64, AES-128-CTR_CMAC-64-AAD, AES-256-CTR_CMAC-64-AAD, ECDH_AES-128-CTR, ECDH_AES-256-CTR, ECDH_AES-128-CTR_CMAC-64, ECDH_AES-256-CTR_CMAC-64, ECDH_AES-128-CTR_CMAC-64-AAD, ECDH_AES-256-CTR_CMAC-64-AAD.

- A Sender / Receiver shall support the AES-128-CTR **mode**. Support for all other modes is optional. For modes of operations with authentication, a 64-bit truncated MAC shall be supported. The CMAC function used for authentication shall use the privacy cipher **key**. When the key is 128 bits the CMAC function shall use the AES-128 block cipher. When the key is 256 bits the CMAC function shall use the AES-256 block cipher.

*Note*: `The modes AES-128-CTR_CMAC-64 and AES-256-CTR_CMAC-64 are similar in their structure to the CCM mode specified in NIST SP 800-38c with the AES-CBC-MAC function replaced by the more robust AES-CMAC function.`

The **key** shall correspond to the **privacy_key** defined in the Privacy Key Derivation section.

The **iv'_ctr** value shall correspond to **iv'** || **ctr**. The **iv'** value shall be a 64-bit Octet String in binary form. It shall derive from the **iv** parameter of the stream associated SDP transport file and/or NMOS transport parameters. The **iv'** value shall correspond to the base **iv** value for a


BY-ND

stand-alone unidirectional stream or the sum of the base **iv** and a sub-stream index in the range [0, 1023] for a multiplexed and/or bidirectional stream.

The **ctr** value shall be a 64-bit Octet String in binary form. It shall be transmitted by the Sender along with the ciphered content. This 64-bit value shall be transmitted in the ctr_high and ctr_low fields of the Full RTP Extension Header. The least significant 24 bits shall be transmitted in the ctr_short field of the Short RTP Extension Header. The **ctr** value shall start at 0 and increment by 1 modulo $2^{64}$ at every slice being encrypted

The **ctr** shall start at 0 for a new key and may continue counting for a given **key** if it is known that the **ctr** value cannot wrap-around during the active time of the Sender/ Receiver.

A Receiver shall recover the full **ctr** value from the ctr_low and ctr_high fields of the Full RTP Extension Header as follow:

$ctr = ctr\_high_0 || ctr\_high_1 || ctr\_high_2 || ctr\_high_3 || ctr\_low_0 || ctr\_low_1 || ctr\_low_2 || ctr\_low_3$

A Receiver shall recover the full **ctr** value from the ctr_short field of the Short RTP Extension Header as follow:

$prev24 = ctr_5 || ctr_6 || ctr_7$

$new24 = ctr\_short_0 || ctr\_short_1 || ctr\_short_2$

If the value corresponding to prev24 is smaller than the value corresponding to new24; then the recovered **ctr** is $ctr_0 || ctr_1 || ctr_2 || ctr_3 || ctr_4 || ctr\_short_0 || ctr\_short_1 || ctr\_short_2$; else the recovered **ctr** is $ctr_0 || ctr_1 || ctr_2 || ctr_3 || ctr_4 || ctr\_short_0 || ctr\_short_1 || ctr\_short_2 + 00 || 00 || 00 || 00 || 01 || 00 || 00 || 00$.

***Note***: The else clause is adding 1 to the value corresponding to $ctr_0 || ctr_1 || ctr_2 || ctr_3 || ctr_4$

RTP Header, RTP Header Extensions and RTP Payload Header shall not be encrypted. As per RFC 8088 (How to Write an RTP Payload Format) The RTP Payload Header is defined as follows: "*RTP payload formats often need to include metadata relating to the payload data being transported. Such metadata is sent as a payload header, at the start of the payload section of the RTP packet. The RTP packet also includes space for a header extension [RFC5285]; this can be used to transport payload format independent metadata, for example, an SMPTE time code for the packet [RFC5484]. The RTP header extensions are not intended to carry headers that relate to a particular payload format, and must not contain information needed in order to decode the payload*."

Depending on the RTP Payload Format, and as specified in the associated RFC specification, the first N bytes of the RTP Payload may contain an RTP Payload Header, as defined in the RFC payload format specification, and as such, shall not be encrypted. Otherwise, if the RFC payload

DRAFT



BY-ND

format specification does not define an RTP Payload Header, the complete RTP Payload shall be encrypted.

The mechanism used by a Sender or Receiver to detect which portion of the RTP Payload is encrypted is outside the scope of this Technical Recommendation. It is expected that an AMWA NMOS message and/or an SDP transport file associated with a Sender media stream would provide the necessary payload format information to a Receiver.

Modes of operation using AAD shall calculate the MAC over the following aad_full Octet String when the Full RTP Extension Header is used, and aad_short Octet String when the Short RTP Extension Header is used, prior to being calculated over the payload data. The Octet String are in binary form.

RTP_KV protocol adaptation:

aad_full = 00000000 || dynamic_key_version || ctr_high || ctr_low

RTP protocol adaptation:

aad_full = 00000000 || 00000000 || ctr_high || ctr_low

RTP and RTP_KV protocol adaptation:

aad_short = 0000000000000000|| 0000000000 || ctr_short

***Note***: `Without AAD the MAC applies to the RTP Payload, precisely to the RTP Payload content that is to be encrypted. As described before, RTP Header, RTP Header Extensions and RTP Payload Header are not encrypted and hence are not subject to the MAC. When using AAD additional data is subject to the MAC while still keeping the objective of excluding RTP/UDP/IP specific information.`

***Note***: `In order to obtain a fully secure MAC of the encrypted payload a mac-then-encrypt scheme is used with the CMAC function. By using the mac-then-encrypt instead of an encrypt-then-mac, one minimizes the required key material, using the same key for both encryption and authentication. The encrypt-then-mac construction would require using different keys for the encryption and authentication.`

## 21.1 RTP Header Extensions

The following RTP Extension Headers should be declared in the SDP transport file as per RFC 8285.

The following URN shall be used to associate the ID field of the header extensions for the full and short flavors: "urn:ietf:params:rtp-hdrext:PEP-Full-IV-Counter", "urn:ietf:params:rtp-hdrext:PEP-Short-IV-Counter".



CC BY-ND

DRAFT

The a=extmap attribute shall be used to declare the RTP Extension Headers in the SDP transport file using "sendonly" as the direction parameter.

### 1.1.1 CTR Full RTP Extension Header

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 0xBE            | 0xDE            | length=4                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ID     | L=14   |0| RESERVED                                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| dynamic_key_version all Octet of the key_version Octet String |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ctr_high first 4 Octet of the ctr Octet String                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ctr_low last 4 Octet of the ctr Octet String                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Table 3- CTR Full RTP Extension Header

- All the field are in big-endian unless otherwise specified
- All the Octet String are binary form
- bit 8 of the second 32-bit word shall be set to 0
- RESERVED bits of the second 32-bit word must be set to 0
- ctr_high corresponds to the most significant bits of **ctr** (first 4 Octet of **ctr** Octet String)
    - $ctr_0 \| ctr_1 \| ctr_2 \| ctr_3$
- ctr_low corresponds to the least significant bits of **ctr** (last 4 Octet of **ctr** Octet String)
    - $ctr_4 \| ctr_5 \| ctr_6 \| ctr_7$

DRAFT

### 2.1.1 CTR Short RTP Extension Header

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 0xBE            | 0xDE            | length=1                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ID     | L=2    | ctr_short last 3 Octet of the ctr Octet String|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Table 4 - CTR Short RTP Extension Header

- All the field are in big-endian unless otherwise specified
- All the Octet String are binary form
- ctr_short corresponds to the least significant bits of **ctr** (last 3 Octet of **ctr** Octet String)
    - $ctr_5 \| ctr_6 \| ctr_7$

## 21.2 RTP Payload format



BY-ND

```
+-+-+-+-+-+-+-+-+-+-+-+-+ ... +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RTP Headers |RTP Header Extensions  | RTP Payload Header  |0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+ ... +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|3 4 5 6 7 8 9 A B C D E F|0 1 2 3 4 5 6 7 8 9 A B C D E F|0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|4 5 6 7 8 9 A B C D E F|0 1 2 3 4 5|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Table 5 - RTP Payload format

After the unencrypted RTP Header, RTP Header Extensions and RTP Payload Header the encrypted part of the RTP Payload shall be encoded as a big-endian sequence of bytes subdivided into zero or more complete data slices of 16 bytes, that may be terminated by a partial data slice of less than 16 bytes. Partial data slices shall be assumed to be zero-filled to complete a big-endian data slice of 16 bytes by the AES encryption/decryption internal process. The provided bytes of the partial data slice correspond to the most significant bytes of the big-endian data slice. The zero filled bytes shall be ignored/discarded and not be considered as being part of the RTP Payload.

***Note***: Any RTP Payload is allowed to terminate with a partial data slice of less than 16 bytes.

A CTR Full RTP Extension Header shall be used in the first RTP packet of a video frame/field, a video frame/field slice, and an audio frame/packet. A CTR Full RTP Extension Header shall also be used in every RTP packet that is not categorized as video or audio. The usage of CTR Full RTP Extension Headers shall ensure that the distance between the associated **ctr** values of two consecutive Full RTP Extension Headers is less than $2^{24}$ units. When privacy encrypting an HDCP encrypted RTP stream in-place, and the HDCP encrypted RTP stream uses a Full RTP Extension Header, a CTR Full RTP Extension Header shall be used. A CTR Short RTP Extension Header shall be used when a CTR Full RTP Extension Header is not used.

The RTP packets after the first one, if any, completing a video frame/field, a video frame/field slice or an audio frame/packet should use a CTR Short RTP Extension Header. The concept of "frame" is used for uncompressed and compressed audio and video. The concept of "field" is used for uncompressed and compressed video. The concept of "packet" is used for uncompressed and compressed audio.

***Note***: A video frame/field may be segmented into slices. Each such slice uses a CTR Full RTP Extension Header in the first RTP packet.

***Note***: An audio "packet" represents the generic grouping of a number of audio samples into a processing unit.

When a message authentication algorithm is used along with the cipher, the MAC shall be stored as the last N bytes of the RTP Payload. These last N bytes shall be used only by the encryption/decryption process and are not part of the effective RTP payload before/after



BY-ND

DRAFT

encryption/decryption. The RTP payload to encrypt shall be N bytes less than the maximum payload length value possible for a given RTP payload format in order to allow including the N bytes in the length value after encryption.

## 21.3 Dynamic key_version

A Sender/Receiver configured for in-band dynamic changes of the **key_version** may change the **key_version** value dynamically at natural boundaries of the media content (frame, field or GOP boundary for video and ancillary data, packet boundary for audio and generic data) to change the Privacy Cipher encryption key. The current value of the **key_version** shall be transmitted in clear to the peer through the dynamic_key_version field of the CTR Full RTP Extension Header. The dynamic_key_version value shall correspond to the **key_version** value used for deriving the encryption key of the associated RTP Payload.

When a Receiver configured for in-band dynamic changes of the **key_version** becomes active it shall select an initial **key_version** value. Subsequently it may increment the selected **key_version** value by 1 modulo $2^{32}$ to change the associated **privacy_key** during the activation. The **key_version** may be shared by a number of streams / sub-streams encrypted by a Receiver Device. A Sender/Receiver configured for in-band dynamic changes of the **key_version** shall use the **key_version** received in clear from the peer through the dynamic_key_version field of the CTR Full RTP Extension Header to derive the Privacy Cipher decryption key of the associated RTP Payload.

## 21.4 IPMX integration with HDCP support

The privacy encryption protocol RTP adaptation is compatible with IPMX devices already supporting HDCP in AES-128-CTR mode. An HDCP encrypted content may be privacy encrypted in-place using the same HDCP CTR Full/Short RTP Extension Headers.

- The FRZ control signal of the HDCP CTR Full RTP Extension Header shall be ignored by PEP. It shall be set to 0 in the HDCP CTR Full RTP Extension Header such as to always maintain HDCP encryption and the transmission of HDCP CTR Full/Short RTP Extension Header with appropriate values.

- The StreamCtr value of the HDCP CTR Full RTP Extension Header shall be ignored by PEP.

- The InputCtr value of the HDCP CTR Full/Short RTP Extension Header shall be used by PEP as the ctr_high, ctr_low and ctr_short values.

The privacy cipher described in this document is compatible with the HDCP cipher when $\mathbf{lc_{128}}$ and **streamCtr** at the cipher level are forced to 0. The $\mathbf{r_{iv}}$ value derives from the **iv'** value. The $\mathbf{k_s}$ value is the **key** derived from **PSK**, **key_generator, key_version** and **key_pfs** values.



CC BY-ND

## 21 Other transport protocol adaptations

For transport protocols other than RTP, this Technical Recommendation assumes no control over the encryption process.

The **key** derives from the **privacy_key** defined in the Privacy Key Derivation section ( key = f(privacy_key) ) using a process that is specific to the transport protocol adaptation.

The effective value of **iv'_ctr** is protocol specific.

The sections of the stream that are encrypted and/or authenticated are protocol specific.

DRAFT



cc BY-ND