Network Working Group
Internet-Draft
Intended status: Experimental
Expires: January 23, 2016

A. Farrel
Juniper Networks
S. Farrell
Trinity College, Dublin

July 23, 2015

Opportunistic Security in MPLS Networks
draft-ietf-mpls-opportunistic-encrypt-00.txt

Abstract

This document describes a way to apply opportunistic security between adjacent nodes on an MPLS Label Switched Path (LSP) or between end points of an LSP. It explains how keys may be agreed to enable encryption, and how key identifiers are exchanged in encrypted MPLS packets. Finally, this document describes the applicability of this approach to opportunistic security in MPLS networks with an indication of the level of improved security as well as the continued vulnerabilities.

This document does not describe security for MPLS control plane protocols.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Table of Contents

1. Introduction 3
1.1. Experimental Status 4
1.2. Existing Security Tools for MPLS Data 5
2. Principles of Opportunistic Security 5
2.1. Why Do We Need Opportunistic Security? 5
2.2. Opportunistic Security at 10,000ft 7
2.3. What about a Man-in-the-Middle? 8
2.4. OS in MPLS Overview 9
3. MPLS Packet Encryption 11
3.1. MPLS Encryption Label 14
3.2. Control Word 14
3.3. Considerations for ECMP 15
3.4. Backward Compatibility 16
3.5. MTU Considerations 17
3.6. Recursive Encryption 17
4. Key Exchange For Opportunistic Security in MPLS 17
4.1. Initiating MPLS-OS 18
4.2. MPLS G-ACh Advertisement Protocol for Key Exchange 19
4.3. Key Exchange Protocol 19
4.4. Indicating the Return Path 24
4.5. Protecting the Key Exchange Protocol Messages 25
5. Applicability of MPLS Opportunistic Security 25
5.1. Tunnel MPLS Packets 27
5.2. Penultimate Hop Popping 28
6. Security Considerations 29
6.1. Security Improvements 29
6.2. Applicability 29
6.3. Continued Vulnerabilities 29
6.4. New Security Considerations 29
7. Manageability Considerations 30
7.1. MITM Detection 30
8. IANA Considerations 30
8.1. GAP Key Exchange TLV 30
8.2. Key Derivation Functions and Symmetric Algorithms 31
9. Acknowledgements 31
10. References 31
10.1. Normative References 31
10.2. Informative References 32

Authors' Addresses 34

1. Introduction

MPLS is an established data plane protocol in the Internet. It is found in the majority of core service provider networks and most end-to-end traffic in the Internet will be carried over MPLS at some point in its path. The MPLS data plane is defined by [RFC3031] and [RFC3032].

Data security (e.g., confidentiality) in MPLS has previously relied on just two features:

- Physical isolation of MPLS networks has been used to ensure that interception of MPLS traffic was not possible.

- Higher-layer protocol security (such as IPsec [RFC4302], [RFC4303]) has been used whenever a particular flow has determined that security was desirable.

These features have a number of significant vulnerabilities:

- Networks are increasingly easily compromised physically such that "taps" may be inserted in links between routers [RFC7258].

- Routers may be compromised either in their entirety or through the management/control plane (or misconfiguration). This may result in packets being diverted to transit inspection points on their way to their destination.

- The increased support for point-to-multipoint (P2MP) MPLS means that routers can easily be configured (or misconfigured) to make a copy of data and to send it to an additional destination.

- End-to-end payload security may be hard to manage and operate and is not turned on by default by many users. While this form of security is desirable, the network should also improve the security of data transfer that it offers.

The concept of Opportunistic Security (OS) is introduced in [RFC7435]. This document describes an OS design pattern for the MPLS data plane. It shows what part of an MPLS packet may be encrypted and provides a way to indicate that the packet is encrypted as well as to carry a key identifier with each packet.

MPLS opportunistic security can be achieved between adjacent Label Switching Routers (LSRs) on an MPLS Label Switched Path (LSP), and also between end points of an LSP.

This document also provides a mechanism for keys to be exchanged to

facilitate encryption. Finally, this document describes the applicability of OS in MPLS networks with an indication of the level of improved security as well as the continued vulnerabilities.

This document does not describe security for MPLS control plane protocols.

Please note that a discussion of the applicability of MPLS opportunistic security is provided in Section 5.

1.1. Experimental Status

This document is presented as experimental. Before advancing this work on the IETF's Standards Track, it is important to get experience of the practicality of the mechanisms described. In particular whether it is practical to achieve these mechanisms in existing hardware, and whether the imposition of additional MPLS labels is acceptable in the MPLS data plane. Additionally, the consequences of the reduced MTU caused by inserting the additional MPLS label and control word as well as the fact that the encrypted packet will be larger than the unencrypted packet need to be investigated.

It is currently believed that MPLS OS can be deployed progressively without the need to negotiate capabilities outside the key exchange mechanisms described here. This means that no specific walled garden needs to be described in this document.

Experimentation and further investigation of the security aspects of these mechanisms are encouraged especially with regard to mitigation of man-in-the-middle attacks. Consideration of the impact of MPLS OS on MPLS Operations, Administration, and Management (OAM) and other MPLS management techniques also needs further exploration.

The key functions of MPLS OS described in Section 2.4 are based on an initial set of choices that may be adequate for MPLS OS. However, security knowledge is evolving and it may be advisable to "upgrade" for example to Elliptic Curve Diffie-Hellman (ECDH) [RFC6239], using NIST curves or new curves (such as 25519). Furthermore, alternative key derivation functions could be chosen, or symmetric cipher mode could be used. Note that changing to a symmetric cipher that is faster in software, but less likely to be available in hardware would not be a good change.

Section 2.4 also describes the frequency with which keys should be changed. The values described here should be subject to more research and experimentation since key change is fundamental to the actual security of the encryption.

Section 4.3.3 defines the input parameters to the key derivation function and includes the LSP identifier. This identifier is only needed if the scope of the key is per LSP. This document is written on that assumption because of the need to rotate the key after a certain number of packets have been transmitted. However, this could be the subject of some investigation since dropping the LSP identifier would simplify the TLV and the computation. It would also address the issue of identifying the LSP in the case of LDP.

Section 4.3.3 also specifies that the alt is not used. Further investigation is needed to see whether this input parameter would add value.

Note that this experiment uses a special-purpose MPLS label. Since this document is experimental it makes use of an extended special-purpose label from the experimental range. If this work is moved to be published on the standards track, it will be possible to achieve the same function using a simple special-purpose label rather than an extended special-purpose label.

1.2. Existing Security Tools for MPLS Data

This section is a placeholder for text that needs to be added to describe existing security tools for securing MPLS Data. The text needs to describe the use of IPsec used for the payload of MPLS LSPs, and should also cover the use of link layer security (such as MACsec).

>>> TBD

2. Principles of Opportunistic Security

This section provides an overview of opportunistic security in the context of MPLS. Readers are advised to familiarize themselves with some of the attack vectors discussed in [RFC7258] and with the more general description of opportunistic security as described in [RFC7435]. The text here is intended for the consumption of MPLS experts who may not have a background in security: it is, therefore, tutorial and simplistic in nature.

2.1. Why Do We Need Opportunistic Security?

To introduce this discussion we start from a basic view of how encryption is typically used in IETF protocols.

Say we have two protocol entities, Alice and Bob, and they would like some message "M" sent from Alice to Bob to have confidentiality. Alice needs to send M encrypted with algorithm "E" under some

symmetric (e.g., AES) key, "k". Thus Alice wants to send Bob "E(k,M)", but for Bob to be able to understand (i.e., decrypt) the message Alice and Bob both need to agree on the key that will be used: this is called their shared secret.

In many IETF protocols, such as the common usage of Transport Layer Security (TLS) S/MIME Cryptographic Message Syntax (CMS) or Pretty Good Privacy (PGP), Alice simply invents a random key "k" and then encrypts that under Bob's public key "Pub-b" and sends Bob both E(Pub-b,k) and E(k,M). (There are lots of other details and other options for how this can be handled, but we ignore those for now.) In such cases, before Alice can send "E(k,M)", she needs to acquire Bob's public key and she needs to be certain that it really is Bob's public key and not Charlie's. That knowledge requires some long-term key management, which is often done using a Public Key Infrastructure (PKI) so that Alice actually stores the public key (Pub-ca) of a Certification Authority (CA), and Bob gets his public key (Pub-b) "certified" by the CA, which means the CA creates a digitally signed data structure "Cert(Pub-ca,Pub-b)". The crucial thing is that Alice, Bob, and a CA need to co-ordinate before Alice and Bob can agree on a key "k", and that process imposes a key-management burden.

Doing such key management is clearly quite possible, since TLS and IPsec and other well-deployed technologies depend on it. But, in the case of HTTP/TLS on the public web, we see that only roughly 30% of web sites actually take on this burden, even though the software required is ubiquitous and, at least for 2nd level DNS domains in .com for example, there are CAs who offer free domain-validated certificates. While some of the 70% who don't set up certificates might not actually want confidentiality, there are certainly some who would and arguably many that would benefit from confidentiality, if it just happened out of the box, without an administrator having to do anything. And there are also arguably many other protocols where the same is true.

An alternative to the PKI is manual configuration of keys at Alice and Bob. Manual configuration is used in a large number of cases in deployments, however it has a set of issues that make it problematic. These issues include:

- the scale of configuration that is needed for a full set of Security Associations (SAs) between all communicating parties
- the likelihood of configuration errors
- the security vulnerabilities associated with manual keying and unsecured exchange of keys.

Opportunistic Security (OS) is a protocol design pattern to achieve encryption between Alice and Bob without requiring key-management through CAs and without relying on manual configuration of keys.

2.2. Opportunistic Security at 10,000ft

Instead of the "key transport" mechanisms described in Section 2.1, OS aims to use "key agreement". With key management, Alice invents "k" and safely transports it to Bob encrypted with Bob's public key as "E(Pub-b,k)". With key agreement, both Alice and Bob contribute to calculating "k" as follows.

Assume that Alice and Bob are using some protocol where they can exchange a few messages in order to agree on the key "k" to use. With a Diffie-Hellman key agreement ("D-H") both Alice and Bob have public and private values, where the private value can be randomly generated, perhaps even once per message "M". They swap the public values, and can then, thanks to the "magic" of Diffie-Hellman, derive a key "k" that nobody else can know.

In this way Alice sends Bob "Pub-a" and Bob sends Alice "Pub-b" and at that point both of them can safely calculate a shared secret "k" from those values. And after that Alice can send Bob "E(k,M)".

From here on, we change the terminology slightly and refer to Alice as the initiator, with private key "i" and Bob as the recipient, with private key "r" so that our notation is closer to that used in IPsec's Internet Key Exchange Protocol (IKE) on which we model our use of OS.

D-H works as follows: Let "p" be well-known large prime number that we use for all modular arithmetic (meaning that "a^b" is actually "(a^b) mod p"), and let "g" be another well-known value (called a generator for the group determined by "p"). Also let Alice and Bob's private values be "i" and "r" respectively. Now, if Alice sends Bob "g^i" as her public value, and Bob similarly sends Alice "g^r" then both of them can easily calculate "g^(i*r)" or "g^ir" but nobody else can, since calculating "x" when only given "g^x" is a computationally hard problem for any "x". Once both Alice and Bob have the value "g^ir" in hand, they can easily derive a value "k" from that using any of a number of well-known key derivation functions (KDFs) such that k = f(g^ir) for a KDF "f".

As you can see from the above, Alice and Bob do not need to pre-arrange anything other than "g", "p" and "f", and those can be public information that is used by everyone everywhere (or at least by all participants in a particular deployment). Yet, Alice and Bob have managed to derive a common and private value for a key "k" that they can use to encrypt (and decrypt) "M".

This method of using the OS pattern provides strong confidentiality and can be built into any protocol that allows Alice and Bob to

occasionally exchange public values.

There are also additional advantages to key agreement when compared to key transport. The most important of those is that with key agreement we can easily ensure that k has a property called Perfect Forward Secrecy (PFS). That means that an attacker has to separately attack each key k. In contrast, if we use the key transport approach, then an attacker who somehow accesses Bob's private key "Priv-b" can record lots of traffic and later go back and decrypt all the "E(Pub-b,k)" values that all Alices have ever sent to Bob. With key agreement as described, since both Alice and Bob contribute to the value k, and since Alice and Bob will typically periodically generate new private values i and r (perhaps even for every single M), compromise of one party is far less catastrophic, and an attacker who gets access to one private value gets far less benefit.

2.3. What about a Man-in-the-Middle?

OS as described so far is vulnerable to Man-in-the-Middle (MITM) attacks. The problem is that Alice does not know that it was really Bob's public value that she received; it could have been Charlie's public value sent by Charlie. And Charlie could also send Bob his public value pretending to be Alice. Now Charlie can share a key with Alice and a key with Bob so that Charlie can sit between Alice and Bob decrypting what he gets from Alice and then re-encrypting it to send to Bob. Neither Alice nor Bob can tell that Charlie is present as a "Man-in-the-Middle" and both Alice and Bob think they are safely exchanging encrypted messages.

A MITM attack like that is bad and making a protocol proof against such attacks comes at the cost of the key-management burden described in Section 2.1. Most IETF protocols to date require that such MITM attacks not be feasible.

However, despite its potential vulnerability to MITM attacks, OS still has value. This value arises because of the difficulty of inserting a MITM actor, and the cost of processing for the MITM in the case of a very large number of relationships. In particular, where the choice is between no encryption (as has been the case for MPLS to date) and OS, it is clear that using OS offers better (although not the best) security.

Consider the case where an attacker taps a link on the path between Alice and Bob. In this case, the attacker can capture every packet between the two parties, and if there is no encryption, can read every message. Furthermore, consider that the attacker could tap a fiber in the core of the network and so capture every packet between

a large number of Alices and their corresponding Bobs. In these cases, Charlie can operate as a "passive MITM" since all he has to do is watch the packets.

With OS in use, Charlie is forced to be an "active MITM". That is he must engage in the D-H exchange between each pair of Alices and Bobs, and he must must decrypt and encrypt each packet he wants to inspect. This imposes a higher cost and is especially burdensome if he is attempting to do it in parallel for lots of Alice/Bob pairs using lots of different keys and communication sessions.

Furthermore, when D-H is in use for OS, management tools can be used to detect the presence of Charlie as a MITM. This is because Charlie has to agree one key "kA" with Alice, and a different key "kB" with Bob. As far as we know, Charlie cannot arrange that kA equals kB because both sides contribute to the key value in the D-H key agreement. That means that if Alice and Bob can check with each other what value of "k" they are using and the values do not match, then they know that Charlie is present. What is more, Alice and Bob can make this check on the value of "k" for any of the "E(k,M)" they ever exchanged.

Thus, in the case of a fiber tap where many Alice/Bob pairs are being monitored, it only takes one Alice and Bob to detect the MITM attack for all Alice/Bob pairs to be alerted to the problem. In such cases the cost of detection for Charlie may be even greater than the cost of performing the MITM attack.

Hence we conclude that OS can have considerable value when used in MPLS networks.

2.4. OS in MPLS Overview

The basic requirement for MPLS-OS is that we want to provide a way for two MPLS nodes to do a key exchange and to derive a session key from that to use in MPLS packet encryption.

To do that we use a Diffie-Hellman key exchange as outlined in Section 2.2. We model this on IKE [RFC7296] using essentially the same parameters. We feed the shared Diffie-Hellman value, which is g^ir, into a standard KDF that also takes as input an LSP identifier (LSP ID) together with the sending and receiving LSR IDs - where the sending LSR is the point of encryption and the receiving LSR is the point of decryption such that the pair of LSRs define the SA. These additional inputs are used to ensure that we end up with different keys on an LSP even if the same g^i and g^r values are re-used, however it is RECOMMENDED that fresh values of i and r are used each time [RFC4086]. The KDF to be used here is as defined in [RFC5869].

The D-H values used MUST be of at least 2048-bits. Implementations MUST support the 2048-bit modular exponentiation (MODP) group from Section 3 of [RFC3526] and SHOULD support the larger MODP groups from [RFC3526].

This document also defines the mechanism used to derive an identifier for a key (the key-id) from the shared Diffie-Hellman value, which is also based on the KDF output. The key will be used with a symmetric encryption algorithm, such as AEAD_AES_GCM_128 (the default, following [RFC5116]).

As with any symmetric block cipher, one should not use the same key for too long. The nonce defined for these keys is derived using a 96 bit counter incremented by one for each encrypted packet. It is critical for security that nonce values MUST NOT be re-used with a given key. (This is an inherent issue with how AES-GCM or any counter mode achieves high performance.)

Accordingly, implementations MUST support mechanisms for key change.

To support key change, this document defines a way for two LSRs using a key on an LSP to agree a new key and to switch over to using that key when desired. That means that implementations MUST be able to handle at least two keys (old and new) for a given LSP. Once a new key has been agreed then it should be used for sending packets; once encrypted data packets protected with the new key have been successfully received, the old key SHOULD be discarded. Section 4 describes how two LSRs agree keys: to agree a new key two LSRs simply run the same key agreement exchange, but this time protected with the old session key as described in Section 4.5. This process can, of course, be repeated any number of times for the same LSP. It is RECOMMENDED that the key on an LSP be changed at least once every day or every 10^6 packets whichever is sooner, and MUST change keys before encrypting 2^64 packets. For an LSP running over a fully-busy 100Gbe interface, we might assume that means roughly 160 million packets per second, or roughly 2^44 packets per day. The 2^64 limit therefore means changing keys daily in the busiest cases of some of the largest current links capacities.

In the event of a key agreement exchange or decryption failure, an alarm MUST be raised to the operator. Default (i.e., node-wide) and per-LSP behavior SHOULD be configurable in this case: actions may include reverting to non-encrypted traffic, re-attempting key exchange, or tearing down the LSP. Note that a simple attack on OS is to tamper with key agreement exchange messages or encrypted packets so that OS fails. Such attacks may be intended to cause the LSP to operate without encryption, so an operator should consider this when setting the behavior in this case.

Section 7.1 also discusses a mechanism that allows a pair of LSRs using OS on an LSP to detect that a MITM attack has happened. For this, we simply define a function of the shared secret, which can be logged and later compared. Note that logging a sample of these "witness" values will likely be sufficient to detect pervasive MITM attacks [RFC7258]. As with the key-id, we base this on the same KDF output.

We might want to consider deriving the witness value from a separate invocation of the KDF that does not depend on the LSP-specific inputs. The benefit from that would be that the same MITM-detection infrastructure could be used for many protocols. However, that would require standardizing a generic D-H MITM-detection protocol, or at least formats, in order to be useful. We also need to consider what additional information needs to be logged with the witness value so that comparisons can easily be made at scale but without creating new privacy-invasive meta-data. That last is not much of an issue for MPLS-OS, but could be elsewhere. At present we do not intend to go for the generic MITM-detection approach, but it is worth considering.

An additional discussion of the applicability of MPLS-OS is found in Section 5.

3. MPLS Packet Encryption

MPLS packets are encrypted according to the mechanisms described in this section.

When an MPLS packet is encrypted, this is indicated by the insertion of a new extended special-purpose label [RFC7274] in the label stack. This is referred to as the MPLS Encryption Label (MEL). The format of the MEL is described in Section 3.1.

The MEL MUST have the bottom of stack bit (the S bit) set and MUST be followed by a pseudowire control word [RFC4385]. The format of the control word is described in Section 3.2.

The remainder of the MPLS packet is encrypted and cannot be parsed without decryption. It needs to be understood, therefore, that the phrase "bottom of stack" refers to the parsable label stack (i.e., those label stack entries that have not been encrypted) and does not indicate the full label stack of the unencrypted packet. Figures 1 and 2 should make this point clear.

Implementations MUST support the AEAD_AES_GCM_128 encryption algorithm, as specified in Section 5.1 of [RFC5116], which is the default algorithm as described in Section 4.3 of this document.

Note that it is critical that a new nonce is used for every encryption. The nonce is an implicit packet counter. The initial nonce value is derived from the HMAC-based Key Derivation Function (HKDF) output (see Section 4.3.2) at key agreement time and the counter is incremented by one for each packet encrypted on the sending side and by one for each packet successfully decrypted on the receiver side.

Although the nonce is not transmitted with the packets, a 16-bit counter carried in the control Word indicates the nonce value modulo 65536. This feature allows a receiving node to quickly spot that a packet has been dropped and resynch its own counter in order to be able to continue to decrypt received packets. In the event that the counter cannot be resynchronized or that more than 65536 packet are lost in one batch the receiver will encounter a decryption error. In this case the receiver may report a general decryption error or may attempt to resynchronize by advancing its own counter in units of 65536 according to the modulo value in the received packet. Note that incrementing the counter in order to test for decryption failure does generate a potential DoS if, e.g., an attacker decrements the nonce-mod-65536 value. Implementations that do such tests SHOULD maintain a small maximum window size beyond which they will cease attempting to decrypt. It could be that throwing an error might be the more effective response if the packet loss rates are expected to be low enough.

It should also be noted that the output from encryption will be 16 octets longer than the input.

The bottom of stack bit is set in the MEL to stop implementations continuing to search down the label stack (which is encrypted) and attempting to use the data as though it was a valid label stack. The control word is needed because many implementations that find the bottom of stack expect the next bytes to be a control word or protocol indicator.

The position of the MEL and control word depend on whether hop-by-hop or end-to-end encryption is being applied.

Figure 1 illustrates the format of an example MPLS packet before and after hop-by-hop encryption. The left hand part of the figure shows a normal MPLS packet with a label stack and payload. The bottom label in the stack has the S bit set. The payload is the data carried by the MPLS packet (such as IP) and may be prefixed by a control word.

The right hand part of Figure 1 shows the same packet after it has been encrypted. The top of stack is a label with value 15 that

indicates that an extended special-purpose label follows. Next comes the MEL with the S bit set. The label value of the MEL is from the experimental range 240-255 and is selected according to the scope of the MPLS OS experiment being run. The MEL is followed by a control word. Everything that follows the control word is the entire original MPLS packet encrypted.

```
 ----------- .      -----------
| Top Label | .    | Label 15  |
+-----------+  .   +-----------+
|   Label   |   .  | MEL   S=1 |
+-----------+    . +-----------+
| Label S=1 |     .| Ctrl Word |
+-----------+      +-----------+
|           |      |           |
~  Payload  ~      ~ Encrypted ~
|           |      |           |
 -----------........-----------
```

Figure 1 : The Use of the MEL for Hop-by-Hop Encryption

Figure 2 illustrates the format of an example MPLS packet before and after end-to-end encryption. The left hand part of the figure shows a normal MPLS packet with a label stack and payload. The bottom label in the stack has the S bit set and the payload may be prefixed by a control word. The right hand part of the figure shows how the top two labels (or however many labels are needed for end-to-end delivery) remain at the top of the label stack. Then follows label 15 to indicate that an extended special-purpose label follows, then

```
 -----------        -----------
| Top Label |      | Top Label |
+-----------+      +-----------+
|   Label   |      |   Label   |
+-----------+.     +-----------+
|   Label   | .    | Label 15  |
+-----------+  .   +-----------+
|   Label   |   .  | MEL   S=1 |
+-----------+    . +-----------+
| Label S=1 |     .| Ctrl Word |
+-----------+      +-----------+
|           |      |           |
~  Payload  ~      ~ Encrypted ~
|           |      |           |
 -----------........-----------
```

Figure 2 : The Use of the MEL for End-to-End Encryption

comes the MEL with S bit set, and a control word. The remainder of the packet is encrypted and contains the rest of the label stack and the payload.

3.1. MPLS Encryption Label

The MPLS Encryption Label (MEL) is a normal label stack entry carrying an extended special-purpose label with a value from the experimental range 240-255. The format of the label stack entry is defined in [RFC3032] and shown in Figure 3.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Label                  | TC  |S|       TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3 : Format of the MEL Label Stack Entry

Label: The value of MEL for this experiment

TC: For end-to-end encryption, the value of the TC field SHOULD be set to the value of the unencrypted label stack entry that immediately precedes the MEL. In the case of hop-by-hop encryption, the value of the TC SHOULD be copied from the TC of the first encrypted label if there is a label stack present. Otherwise this field SHOULD be set to all zero (0b000).

S: MUST be set to one.

TTL: SHOULD be set to two to prevent encrypted packets being accidentally forwarded too far beyond the point of intended decryption. Note that setting to zero might cause a receiver to discard the packet when the MEL becomes top of stack, and setting to one might cause the packet to be sent to the slow path when the MEL becomes the top of the stack even though decryption should be a fast-path function.

The sending LSR MAY choose different values for the TTL and TC fields if it is known that label 15 or the MEL will not be exposed as the top label at any point along the LSP (for example, by penultimate hop popping - but see Section 5 for a discussion of MPLS tunnels and penultimate hop popping).

3.2. Control Word

The control word is inserted after the MEL as described in Section 3. The S bit set to one in the MEL and the presence of the control word helps protect against transit nodes that may perform hashing or

inspection of the label stack and payload packet headers when forwarding MPLS packets (for example, to enable ECMP). The control word indicates that the payload is not a protocol that can be meaningfully hashed or inspected.

The format of the control word is defined in [RFC4385] and shown in Figure 4.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0| Flags |FRG|  Length   | Sequence Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Control Word for Encrypted MPLS

Flags: The Flags field is treated as a four-bit number. It contains the key-id that identifies the algorithm and key as established through configuration or dynamic key exchange as described in Section 4.

FRG: Must be sent as 0, and ignored on receipt. Fragmentation is not used.

Length: MUST be sent as 0, and ignored on receipt.

Sequence Number: This field contains the packet counter (nonce) for the encryption algorithm and key currently in use modulo 65536. It can be used by a receiver to quickly check that the value of the nonce being used for decryption is likely to be correct as described in Section 3.

3.3. Considerations for ECMP

As previously stated, the S bit set in the MEL and the presence of the control word prevent implementations from attempting to use the encrypted MPLS packet and its payload to determine a hash value for uses such as ECMP. However, the resultant label stack shown in Figure 2 will probably not provide sufficient entropy for ECMP purposes.

In order to increase the entropy, an implementation that inserts an MEL and MEL MAY also insert an Entropy Label Indicator (ELI) and Entropy Label (EL) as defined in [RFC6790] ELI and EL are positioned in the label stack before the MEL as shown in Figure 5. The setting of the fields in the ELI and EL label stack entries are as described in [RFC6790].

The ELI and EL will normally occur immediately before the label 15 and MEL pair, but they MAY be placed higher up the label stack.

```
                              -----------
                             | Top Label |
                             +-----------+
                             |   Label   |
           -----------       +-----------+
          | Top Label |      |    ELI    |
          +-----------+      +-----------+
          |   Label   |      |    EL     |
          +-----------+.     +-----------+
          |   Label   | .    | Label 15  |
          +-----------+  .   +-----------+
          |   Label   |   .  | MEL   S=1 |
          +-----------+    . +-----------+
          | Label S=1 |     .| Ctrl Word |
          +-----------+      +-----------+
          |           |      |           |
          ~  Payload  ~      ~ Encrypted ~
          |           |      |           |
           -----------.......-----------
```

Figure 5 : The Use of ELI and EL with MEL

3.4. Backward Compatibility

Keys and encryption algorithms may be configured manually or exchanged dynamically as described in Section 4. These mechanisms provide a preliminary way to protect against a sender encrypting data that the receiver cannot decrypt, however, misconfiguration may lead to a sender using the MEL when the receiver does not support encryption.

When a node finds an unknown label at the top of the label stack it must discard the packet as described in [RFC3031]. Therefore, when a receiver discovers label 15 and does not support extended special-purpose labels it will discard the packet. Similarly when a receiver that supports extended special-purpose labels, but does not support the MEL (i.e., does not support encryption) it will discard the packet. (Note that care must be taken if multiple experiments are being carried out in the same network since a different extended special-purpose label must be used for each experiment.) The net result is that when a sender uses encryption in error, all packets that it sends on the LSP will be discarded by the receiver. Note that in this discussion, "the receiver" may be the next hop if single hop encryption is used, or may be the end of the LSP if end-to-end encryption is used.

Transit nodes that are not actively participating in the encryption will not inspect the MEL except potentially as part of an ECMP hash,

and it should be noted that the use of Special Purpose Labels in hashing is strongly discouraged (see Section 2.4.5.1 of [RFC7325]). This means that transit nodes will not encounter the MEL during normal packet processing and will not discard packets.

3.5. MTU Considerations

Adding label 15, the MEL, and the Control Word as described above will reduce the available data size by 12 octets. Furthermore, as described in Section 3, the output of the encryption algorithm is at least 16 octets longer than the input. Therefore, the use of encryption reduces the available MTU by at least 28 octets. Other encryption algorithms may result in even greater reductions in the available MTU.

When end-to-end encryption is in use this can be considered by the ingress LSR, however, when single-hop encryption is in use the participating LSRs need to advertise this reduction in link MTU so that the packets do not overflow. MPLS packets MUST NOT be fragmented as a result of encryption.

3.6. Recursive Encryption

The use of MEL and control word described in Section 3 may be applied recursively. That is, the payload of an encrypted MPLS packet may, itself be an encrypted MPLS packet. This may be particularly useful in the case where an MPLS VPN has native MPLS traffic.

There are no special considerations except to note that encryption and decryption processing may be burdensome if an LSP and its payload LSP have encryption applied at the same LSR. Additionally, it should be noted that, as described in Section 3.6, each recursive encryption reduces the MTU by 28 octets.

4. Key Exchange For Opportunistic Security in MPLS

For encryption to be useful both ends of an encrypted session must know which algorithm is in use and which key to use. The mechanism described in Section 3 provides a way to indicate an index into a table of algorithms and keys that can be used to decrypt an encrypted MPLS packet.

It is possible that this table has been manually configured or set up using a key exchange protocol such as Internet Key Exchange version 2 (IKEv2) [RFC7296]. However, such a process implies a stable security association between encrypter and decrypter of MPLS packets. While such a stable association is entirely consistent with the concept of OS, OS nonetheless calls for a more dynamic key agreement method.

This section provides a mechanism for adjacent MPLS LSRs, or for a pair of LSRs at opposite ends of an MPLS LSP, to dynamically exchange keys and algorithm identifiers so that encryption may be applied opportunistically.

The mechanism uses message exchanges in the MPLS Generic Associated Channel (G-ACh) [RFC5586] as part of the MPLS Generic Associated Channel (G-ACh) Advertisement Protocol (GAP) [RFC7212]. This channel is in-band with an LSP and may be used to carry messages between neighbors or between the end points of the LSP. A type field within the common message header, the Associated Channel Header (ACH), is used to indicate the type of message carried.

Nodes that receive G-ACh messages and do not understand them, or nodes that understand the G-ACh but do not recognize the ACH message type drop the packets as described in [RFC5586].

Note that this mechanism may benefit from encryption that is already in use on an LSP. Thus key changes using this mechanism can be made using encrypted messages.

4.1. Initiating MPLS-OS

This document assumes that the use of MPLS-OS is initiated by the upstream of a pair of LSRs (either a pair of adjacent LSRs on an LSP, or a pair of LSP end points). That is, the upstream LSR send the first G-ACh that initiates key exchange. The key that is generated after the exchange is then used to encrypt traffic travelling in the direction between initiating and responding LSRs: that is, from upstream to downstream LSR.

In the case of a bidirectional LSP, each direction is treated separately. That is, "upstream" refers to the direction of traffic flow, and not to any signaling that is used to establish the LSP. Thus, it is possible that a bidirectional LSP uses MPLS-OS on none, either one, or both of the directions of traffic flow for the LSP. But it is important to note that the keys used are different in each direction, each being generated and exchanged through a separate instance of the procedures described in this document. Note that the input parameters for key derivation listed in Section 4.3.3 show LSP identifier, initiator LSR identifier, and responder LSR identifier as three of the ordered list of pieces of information used by the key derivation function. In the case of a bidirectional LSP, the LSP identifier will be the same in each direction, and the two LSR identifiers will be the same, but the LSR identifiers will be used in the reverse order at the two end points of the MPLS-OS exchange and this will reduce the chance of the same key being used in each direction.

Note also that in the case of a pair of unidirectional LSPs in reverse directions between a pair of LSRs there should be no relationship between the keys used on each LSP even if there is a tight coupling between the LSPs such as might be the case for associated bidirectional LSPs [RFC7551]. The key derivation function will use different LSP identifiers as well as using the LSR identifiers in a different order.

4.2. MPLS G-ACh Advertisement Protocol for Key Exchange

GAP defines messages exchanged in G-ACh on a common Associated Channel Type code point (0x0059) [RFC7212]. The application for which the messages are exchanged is defined by the Application ID field carried in the Applications Data Block (ADB). MPLS OS capability notification and key exchange uses the GAP Application ID (0x0000) defined by [RFC7212] and a new ADB TLV for MPLS OS.

Implementations that do not support GAP will discard received packets with this Associated Channel Type as described in [RFC5586]. Implementations that support GAP but that do not support key exchange will discard received packets with this ADB TLV as described in [RFC7212]. Either of these discards will result in no dynamic key exchange, but other key definitions are still supported (such as manual configuration) and may be used to construct a table of algorithms and keys that can be used to achieve MPLS encryption using the mechanisms described in Section 3.

4.3. Key Exchange Protocol

4.3.1. Communication Channels

The key exchange protocol described in this document uses a D-H exchange that assumes a bidirectional communication channel. GAP is designed to run over a unidirectional channel and uses normal IP forwarding for return path messages with an optimization to use the return path of a bidirectional LSP. However, LSPs in packet networks are usually unidirectional. That means that, while the key exchange messages can be sent on the LSP in one direction, a channel needs to be established for the return messages.

This document uses a process similar to that defined for MPLS LSP Ping [RFC4379] and [RFC7110], and that described to indicate a return path for MPLS performance measurement in [I-D.ietf-mpls-pm-udp-return]. That is, the forward message is sent on the LSP and includes the identity of the return path communication channel. The return path may be indicated as a UDP communication over IP, as an LSP running in the opposite direction, or as the reverse direction of a bidirectional LSP.

Note that the GAP messages defined in [RFC7212] include a TLV that enables authentication. This feature SHOULD be used if possible, but it is in the nature of opportunistic security that the necessary security association might not exist. In this case the ability to tamper with the instructions that select a return path may provide a mechanism that makes MITM attacks easier. An implementation that initiates key exchange for MPLS Opportunistic Security MUST verify that the response messages are received on the expected return path channel and SHOULD raise an operator alert if the channel is unexpected.

4.3.2. Key Exchange Messages

The format of a GAP message is described in [RFC7212]. When used for key exchange the GAP message includes an ADB with the fields set as follows.

Application ID is set to 0x0000.

Element Length is set to the total length in octets of this ADB including the Application ID and this field.

Lifetime field SHOULD be set to zero and MUST be ignored.

A key exchange ADB MUST include a Key Exchange TLV as shown in Section 4.3.3. The ADB and MAY also include an Authentication TLV as described in [RFC7212] to provide authentication and integrity validation for a GAP message (see Section 4.5). Additionally, the ADB MAY include a Source TLV as described in [RFC7212] and discussed in Section 4.4.

4.3.3. Key Exchange TLV

A session key is to be established between an initiator (Alice) and a recipient (Bob). The D-H public value for Alice is g^i and for Bob, g^r. The shared Diffie-Hellman value is g^ir. g^ir is represented as a string of octets in big endian order padded with zeros if necessary to make it the length of the modulus. Both g^i and g^r will be 2048 bits long, if the Diffie-Hellman modulus is 2048 bits long.

The Key Exchange TLV is modelled on that from Section 3.4 of [RFC7296] with the addition of information to identify the LSP and its return path, and is encoded as shown in Figure 6.

```
                     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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |    Reserved   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|D|Rsvd | Return|              Path Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           LSP-ID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Algorithm   |   Group Num   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
~                       D-H Public Value                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6 - Key Exchange Message TLV

Type is set to TBD1 to indicate that this is a Key Exchange TLV

The Reserved and Length fields are defined in [RFC7212].

The flag D denotes the direction of the message, '0' indicates a message from initiator (Alice) to recipient. '1' indicates the reverse direction.

The Rsvd bits are reserved. They SHOULD be set to zero and ignored on receipt.

The Return field is used on a message from the initiator to indicate the type of return path to be used for messages from the responder. The Path Identifier field is interpreted in this context. Possible values are as follows:

0 The reverse path of a bidirectional LSP is to be used for the response. Used on a message from an initiator.
1 The reverse path messages are to be sent encapsulated in UDP. Used on a message from an initiator.
2 Any LSP between the recipient and the initiator may be used.
3 Any LSP between the recipient and the initiator that is already using MPLS-OS may be used.
4 The reverse path messages are to be sent on a specific LSP.

All other values are undefined and MUST be processed as an encoding error as described in Section 4.3.4. Similarly, if the value zero is used on a unidirectional LSP then it MUST be handled as an encoding error.

The Path Identifier is interpreted in the context of the Return field. The field only has meaning on messages from the initiator and SHOULD be ignored on responses. If the Return is set to the following values, the Path Identifier has the following meaning:

0 In this case the Path Identifier field has no meaning and SHOULD be ignored.
1 The Path Identifier field contains a UDP port number from the dynamic port range that the initiator will listen on for a response.
2 In this case the Path Identifier field has no meaning and SHOULD be ignored.
3 In this case the Path Identifier field has no meaning and SHOULD be ignored.
4 The Path Identifier field contains an LSP-ID that must be used for reverse path messages.

See Section 4.4 for more discussion of return paths.

The LSP-ID parameter indicates the LSP to which this key exchange applies. On messages from initiator to recipient this field MUST be set to the LSP on which the message flows and any mismatch MUST be treated as an encoding error (Section 4.3.4). On messages from recipient to initiator, this value MUST be copied from the received message and an initiator that cannot match the message and LSP-ID to a message that it previously sent MUST treat the situation as an encoding error.

The Algorithm field is a one octet field that specifies both the KDF to use and the symmetric algorithm to be used for data packet encryption. A registry for values of this field is defined in Section 8.2. The value 0 is used to indicate the default KDF and symmetric encryption mode. An implementation receiving a value for an Algorithm it does not support MUST treat the case as an encoding error as described in Section 4.3.4. All implementations MUST support the default KDF. Note that since implementation of encryption and decryption is likely to be implemented in hardware for reasons of data throughput performance, the introduction of new algorithms may be bound by firmware or even hardware upgrades.

The Diffie-Hellman Group Num is from [RFC3526], so the group number for 2048 MODP is decimal 14. Note that this is a one octet field, but is two octets in the [RFC7296] equivalent. This is not an issue because there are only 30 MODP groups defined at present and new groups are not added frequently.

The D-H public value will contain g^i or g^r depending on the direction (i.e., the setting of the D flag) and is in big endian

order.

The length of the Diffie-Hellman public value for MODP groups MUST be equal to the length of the prime modulus over which the exponentiation was performed, prepending zero bits to the value if necessary.

Once both sides have derived g^ir they need to feed that and the other inputs described in Section 2.4 into the KDF indicated by the algorithm field. With the default algorithm (value zero), the KDF to be used is HKDF as specified in [RFC5869].

The parameters for the use of HKDF are:

Hash: SHA-256

Salt: Not used

Skip: Do not skip

Info: The catenation of a fixed string indicating use of MPLS-OS, with the value "MPLS-OS", the first 32 bits of the key exchange message, with the D flag set to 0, plus the LSP ID and the sender and receiver LSR IDs in that order. That is:

```
MPLS-OS||0||payloadLen||alg||group Num||LSP-ID||i-LSR-ID||r-LSR-ID
```

L: The output length in bits is 272.

The fixed string "MPLS-OS" is used as an input here to prevent potential cross-protocol attacks. Those might otherwise be possible if this mechanism were to be copied in other protocols. (If copying this mechanism for any reason, then a different fixed string value should be used.)

LSP-ID is a unique identifier shared between the initiator and receiver (Alice and Bob) that uniquely identifies the LSP.

[[If RSVP-TE is used for signaling, the LSP-ID is known along the LSP and at the two end points. Similarly, the LSP-ID is known if the LSP is manually configured. It is not so clear how the LSP-ID is known for LSPs established using LDP, although possibly we could use the FEC as defined for RFC 4379 and its extensions.]]

i-LSR-ID and r-LSR-ID are the LSR-IDs of the initiator and receiver respectively (Alice and Bob), where an LSR-ID is the 32 bit, globally unique identifier of the LSR as described in [RFC5036] and [RFC4990].

superior security.

The default encryption algorithm, AEAD_AES_GCM_128, specified in Section 3, requires a 128 bit session key.

The 272-bit HKDF output is the catenation of the session key, the key-id, the witness value and the high-order 16 bits of the initial nonce value in that order. That is the session key is the leftmost 128 bits of the HKDF output. The key-id is the next 4 bits, the witness value is the next 124 bits and the last 16 bits are the 16 most significant bits of the initial nonce value. The low order 64 bits of the initial nonce value are set to zero before the first call to the AES-GCM encryption function. The key-id is carried in encrypted packets as described in Section 3.2.

Note that a 4 bit key-id is adequate in a system where, for any one LSP there is one active key and one new or replaced key. There might also be more than one algorithm, and it is possible that new keys need to be pipelined if roll-over is frequent. In the case that a newly-generated key-id is already in use, the key-id value is repeatedly incremented (modulo 16) until an unused value is found. If all 16 values are already in use, the key derivation function should not be executed.

4.3.4. Encoding Errors

Unknown values in received key Exchange TLVs MUST be treated as encoding errors. All messages that constitute encoding errors MUST be silently discarded. That is, such errors MUST NOT cause response messages to be sent since those messages could be used as part of an attack to determine the capabilities of an LSR.

An LSR SHOULD log such errors and notify the operator. However, care is needed even in these actions since they may be externally visible.

4.4. Indicating the Return Path

The key exchange for MPLS-OS requires a two-way exchange of messages. The Return field of the Key Exchange TLV indicates the reverse path to use for key exchange messages relevant to a particular LSP.

Whenever the LSP being secured is bidirectional, the same LSP SHOULD be used for reverse path messages. Otherwise, the initiator selects the communication channel as described in Section 4.3.3.

If UDP is being used and it may be unclear to what address the messages should be sent, the initiator MUST include a Source Address TLV [RFC7212] to provide this information.

Operators should consider the security implications of the return path. The use of an already-secured LSP (Return type 3) may provide

Implementations MUST make the choice of return path request sent by an initiator available as a configuration option. As noted in Section 4.3.1, the fact that the the initial GAP messages might not be protected means that there is the potential to tamper with the instructions that select a return path. This could be used as a vector for MITM attacks. To protect against this, an implementation that initiates key exchange for MPLS Opportunistic Security MUST verify that the response messages are received on the expected return path channel and SHOULD raise an operator alert if the channel is unexpected. In these circumstances an implementation MAY be configured to abort establishment of MPLS-OS although, since that in itself is an attack vector, it is RECOMMENDED that implementations continue toward the use of MPLS-OS while notifying the operator.

4.5. Protecting the Key Exchange Protocol Messages

GAP includes an Authentication TLV that can be used to protect GAP messages as described in [RFC7212]. If there is already an SA between the initiator and recipient this TLV SHOULD be used. However, it is probable with MPLS-OS that no such SA exists and the point of the mechanisms described in this document is to exchange keys in that case, therefore, it is quite likely that the Authentication TLV cannot be used on the first GAP exchanges.

As described in Section 2.4, once one key exchange has been successfully completed, further key exchanges should be protected using a previous key. This is simply achieved since key exchange messages are, themselves, carried in MPLS packets on the LSP and may be subject to encryption exactly as any other packet.

Furthermore, once keys have been established, they may also be used in the GAP Authentication TLV.

5. Applicability of MPLS Opportunistic Security

MPLS-OS provides another tool in the security and privacy toolkit. It is not a panacea and does not solve (nor is it intended to solve) all security or privacy problems. In particular, the use of MPLS-OS does not protect user-data end-to-end that might be better secured using encryption at the IP layer or at higher layers.

As noted throughout this document, the intention of OS in MPLS is to allow one LSR to enable encryption between itself and its neighbor, or between itself and the other end of an LSP, in a dynamic and un-planned way. This can have benefits in a number of scenarios where

the network that generates MPLS traffic transmits it over another network (for example, carrier's carrier, or some deployments of enterprise network). Additionally, the use of MPLS-OS might allow a service provider to offer a secure edge-to-edge service for a variety of applications ranging from VPNs through pseudowires and where the payload traffic might not always be IP. Lastly, in some non-traditional carriers the user data belongs to the operator or is the direct responsibility of the operator (for example, in data centers, or in large-scale private networks).

As with all security mechanisms, there is a trade-off between a number of factors. On one side is the completeness of the security of the user-data, and on the other side is the complexity of configuring and managing the necessary security associations. Furthermore, while mechanisms closer to the end-user than MPLS-OS (for example, TLS and IPsec in tunnel mode) provide better security for user-data by virtue of not transmitting the data across any network hops without it being encrypted, such mechanisms often expose more metadata for inspection by snoopers within the network.

Additionally, while a variety of per-link encryption mechanisms exist and could be used to guard against attacks such as fiber taps, those approaches do not protect against subverted nodes (i.e., routers) on the path since, by definition, per-link encryption does not protect packets once they come off the link. MPLS-OS in the end-to-end LSP mode protects packets on the links and as they cross transit routers.

Nevertheless, it is not the purpose of this document to recommend the use of MPLS-OS to the exclusion of all other encryption techniques. As already mentioned, MPLS-OS is offered as another tool in the tool kit and users as well as network operators are strongly advised to consider using a variety of tools to achieve the level of security and privacy that they desire.

Note that, in order that OS can be used, one end of a peering (neighbor or LSP end) must decide to attempt OS and the other end must support it. This can be determined by the message exchanges described in Section 4.3 since if one peer does not send a key exchange message then encryption will not be used, and if the other peer does not respond then it is unwilling or unable to decrypt messages.

MPLS-OS should be applicable to all forms of MPLS. That is, it should be possible to use it in RSVP-TE systems, in LDP systems, and in MPLS-TP systems (by which we mean those that have manually configured LSPs). Equally, it should work for point-to-point (P2P) and multipoint-to-point (MP2P) uses of MPLS because there is a simple relationship between the sender (encrypter) and the receiver

(decrypter) in both cases. In the MP2P case, the sender's identity can be extracted from the key identifier and there are considered to be enough key identifiers to allow an arbitrary number of senders on the LSP. There will, however, be the need for the receiver to hold OE state (keys, packet counters) for each sender which may be a significant amount of data for an MP2P LSP (although no more than if the same LSP were replaced by multiple P2P LSPs). Additionally, it should be noted that not only will each sender on an MP2P LSP have a different key, but each may separately decide whether to encrypt data or not.

At this time it is not certain whether MPLS-OS can be applied to a point-to-multipoint (P2MP) or a multipoint-to-multipoint LSP in its entirety because packet replication cannot handle the necessary key conversions for each receiver. However, MPLS-OS can certainly be applied to individual hops on these LSPs. Further work is needed to determine whether non-branching multi-hop segments of P2MP and MP2P LSPs can also be protected using MPLS-OS.

5.1. Tunnel MPLS Packets

Note that in the case of tunneling of MPLS packets in another technology (such as MPLS-in-UDP [RFC7510]) there are two approaches that are viable:

- The payload of the encapsulation (i.e., the entire MPLS packet) can be encypted using the mechanisms described in this document without any changes. Any payload identifier in the encapsulation header can remain set to "MPLS" since the encrypted packet is always just an MPLS packet.

- The encryption mechanisms present in the encapsulating technology can be used without any need to use the mechanisms described in this document.

In some cases that processing of one label on the label stack depends on the values contained in the previous label stack entry. For example, in the "Pipe Model" [RFC3270], the Diff-Serv treatment of the packet that is forwarded beyond the end of the tunnel depends on the setting of the TC field in the previous label stack entry. This requires that when a label is popped, the value of the TC field in the label stack entry is cached for use while forwarding. In the case that the next label on the stack is the MEL, decryption of the rest of the packet is required, and this caching would be a little more complicated to implement. This situation is mitigated by setting the TC field of the label stack entry that contains the MEL to the value from the preceding label stack entry as described in Section 3.1.

The "Short Pipe Model" [RFC3270] can be handled using a combination of the above technique and the procedures described in the next section.

5.2. Penultimate Hop Popping

In penultimate hop popping (PHP) a label is removed from the label stack of a packet one hop before the end of the LSP. The packet is forwarded as though it was still carried on the LSP, but the label stack entry for the LSP is removed. Sometimes we say that packet uses the "implicit null label".

When there are additional subsequent labels on the label stack, this has no impact on the use of the mechanisms described in this document. It is possible that after PHP the MEL will become the top label in the stack meaning that the received packet may encounter the MEL as te top label. This has implications for the setting of the TC and TTL fields in the MEL label stack entry as described in Section 3.1.

However, in some cases of PHP the popped label is the bottom of the label stack and the packet after the popped label is some non-MPLS payload protocol (such as IPv6). PHP is used specifically because the receiving interface does not have MPLS capabilities in the forwarding plane. In this situation the packet is identified within the link encapsulation on the final hop by its payload protocol type (such as IPv6). If MPLS-OS is used this situation will change because even when the final label is stripped using PHP there will remain a MEL entry in the label stack. Therefore the packet will need to be identified as "MPLS" in the link encapsulation on the final hop, yet the receiver might not be capable of handling MPLS packets.

This problem can be approached in two ways:

- The penultimate hop may note the presence of the MEL during PHP and attempt to remove the MEL as well. This is unlikely to be successful as the encryption negotiation has been conducted between the end points of the LSP and the penultimate hop is not aware of the keys or algorithms needed for decryption.

  Furthermore, this approach would leave the packet unencrypted on its final hop which may be counter to the intent of the LSP end points.

- The end point of the LSP should recognize that it cannot have both MPLS-OS and PHP. Indeed, in agreeing to the use of MPLS-OS the end point is making a statement about its ability to handle the MEL and

so it can choose:

- to request PHP and allow the penultimate hop to set the payload indicator of the link encapsulation header to "MPLS"; or

- to not request PHP.

6. Security Considerations

6.1. Security Improvements

See Section 2.1.

6.2. Applicability

See Section 5.

6.3. Continued Vulnerabilities

The mechanisms described in this document do not provide protection against certain types of MITM attacks. For example, the key exchange protocol in Section 4.3 will not detect if key exchange messages or their responses are intercepted and discarded such that the initiating peer believes that encryption is not supported. Similarly, those messages may be tampered with such that a receiver cannot determine the correct mapping of table index to algorithm and key when an encrypted packet is received. Furthermore, the MEL in an MPLS packet is not protected and may be overwritten such that a receiver is unable to decrypt the packet.

See Section 7.1 for a discussion of how active MITM attacks can be detected.

6.4. New Security Considerations

If a pair of LSRs do not do the key exchange before sending any data packets on the LSP then those first packets will not be protected by OS and hence will be available to a monitor.

If a MITM can prevent the OS key exchange from completing, e.g. via deleting messages or changing bits in messages, and if the LSRs continue to send data regardless then those data packets will be available to a monitor. That is, in simple terms, a MITM attacker is able to prevent OS from being used through a very simple attack, and the only options for the end points in this situation are to send no data or to send data in the clear. Again, it should be pointed out that this occurrence is not worse than not running OS at all, but has the benefit of being detectable by end points. See Section 2.4 and

Section 7.1 for a description of how alarms should be raised in these circumstances. Furthermore, Section 4.3.1 and Section 4 describe how the return path for key exchange messages might be hijacked to better facilitate MITM attacks and indicates how the initiator of MPLS-OS can detect this and what actions it should take.

Thus, as been previously noted, OS is not a cure for all ills or a prevention against all attacks, but it does offer a way to increase security in some circumstances.

7. Manageability Considerations

As described in Section 2.4 node-wide and per-LSP behavior SHOULD be configurable to describe the action where key agreement exchange or packet decryption fails. In any case, such events MUST trigger alarms to the operator.

7.1. MITM Detection

Section 2.4 introduces the concept of a function of the shared secret that can be compared by two LSRs that are using OS to see whether they are victims of an active MITM attack.

Section 4.3 describes how a witness value is derived for the default KDF, HKDF.

The participating LSRs can simply log this value plus the LSP and LSR IDs from time to time and a management application can compare the values. If they are different for the same LSP ID, then an active MITM attack has taken place.

It needs to be carefully noted that the management channel used to log or otherwise compare the witness values from the two LSRs MUST be secure. It is likely that routers use relatively high security management channels for configuration and other management operations.

8. IANA Considerations

8.1. GAP Key Exchange TLV

IANA maintains a registry called "Generic Associated Channel (G-ACh) Parameters" with a sub-registry called "G-ACh Advertisement Protocol Application Registry" from which new assignments may be made through the "IETF review" allocation policy [RFC5226]. IANA is requested to make a new allocation as follows:

Value	Description	Reference
TBD1	Opportunistic Key Exchange Protocol for MPLS	[This.ID]

8.2. Key Derivation Functions and Symmetric Algorithms

IANA maintains a registry called "Generic Associated Channel (G-ACh) Parameters". IANA is requested to create a new sub-registry called "G-ACh Advertisement Protocol: MPLS Encryption Algorithms Registry" with new values to be assigned through "IETF Review" as defined in [RFC5226].

The available range is 0 - 255.

IANA is requested to record the following information and create an initial entry as follows:

Value	Key Derivation Function	Symmetric Algorithm	Reference
0	HKDF	AEAD_AES_GCM_128	[This.I-D]
1-255	Unassigned		

9. Acknowledgements

Many thanks to Alia Atlas for detailed discussion of the implications and mechanisms of MPLS opportunistic security. Thanks also to Ron Bonica for encouraging this work, to Sean Turner and Stewart Bryant for early review, and to Jeff Haas, Eric Rosen, and Ross Callon for discussions. Thanks for MPLS Review Team comments from Mach Chen and Lizhong Jin, and to Charlie Kaufman for review comments.

Additional thanks to Andy Malis and Danny McPherson for advice about the use of the Control Word.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3526] Kivinen, T., and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC4385] Bryant, S., Swallow, G., Martini, L., and D. McPherson, "Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN", RFC 4385, February 2006.

[RFC5116] D. McGrew, "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5586] Bocci, M., Vigoureux, M., and S. Bryant, "MPLS Generic Associated Channel", RFC 5586, June 2009.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

[RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and L. Yong, "The Use of Entropy Labels in MPLS Forwarding", RFC 6790, November 2012.

[RFC7212] Frost, D., Bryant, S., and M. Bocci, "MPLS Generic Associated Channel (G-ACh) Advertisement Protocol", RFC 7212, June 2014.

[RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels" RFC 7274, June 2014.

10.2. Informative References

[I-D.ietf-mpls-pm-udp-return]
Bryant, S., Sivabalan, S., and S. Soni, "MPLS Performance Measurement UDP Return Path", draft-ietf-mpls-pm-udp-return, work in progress.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, January 2001.

[RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, January 2001.

[RFC3270] Le Faucheur, F. (Ed), "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services", RFC 3207, May 2002.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4379] Kompella, K. and G. Swallow, "Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures" RFC 4379, February 2006.

[RFC4990] Shiomoto, K., Papneja, R., and R. Rabbat, "Use of Addresses in Generalized Multiprotocol Label Switching (GMPLS) Networks", RFC 4990, September 2007.

[RFC5036] Andersson, L., Minei, I., and B. Thomas, "LDP Specification", RFC 5036, October 2007.

[RFC6239] K. Igoe, "Suite B Cryptographic Suites for Secure Shell (SSH)", RFC 6239, May 2011.

[RFC7110] Chen, M., Cao, W., Ning, S., Jounay, F., and Delord, S., "Return Path Specified Label Switched Path (LSP) Ping", RFC 7110, January 2014.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, October 2014.

[RFC7325] C. Villamizar, Ed., "MPLS Forwarding Compliance and Performance Requirements", RFC 7325, August 2014.

[RFC7435] V. Dukhovni, "Opportunistic Security: Some Protection Most of the Time", RFC 7435, December 2014.

[RFC7510] X. Xu et al., "Encapsulating MPLS in UDP", RFC 7510, April 2015.

[RFC7551] Zhang, F., Jing, R., and R. Gandhi, "RSVP-TE Extensions for Associated Bidirectional Label Switched Paths (LSPs)", RFC 7551, May 2015.

Authors' Addresses

Adrian Farrel
Juniper Networks

EMail: adrian@olddog.co.uk

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie