ippm
Internet-Draft
Intended status: Standards Track
Expires: 24 October 2026

F. Brockners
Cisco
S. Bhandari
Databricks
T. Mizrahi
Huawei
J. Iurman
University of Liege
22 April 2026

Integrity Protection of In Situ Operations, Administration, and Maintenance (IOAM) Data Fields

draft-ietf-ippm-ioam-data-integrity-18

Abstract

In Situ Operations, Administration, and Maintenance (IOAM) records operational (including telemetry) information in packets while they traverse a path in the network. RFC 9197 specifies data fields for IOAM (a.k.a IOAM-Data-Fields) and associated data types. This document specifies integrity protection of IOAM-Data-Fields for Intra-IOAM-Domain use cases.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In Situ Operations, Administration, and Maintenance (IOAM) [RFC9197] records OAM information within packets while they traverse a network domain. The term "In Situ" refers to the fact that the OAM data is added to the data packets rather than being sent within packets specifically dedicated to OAM (a.k.a. In-Data-Packet OAM [I-D.ietf-opsawg-oam-characterization]). IOAM is used to complement OAM mechanisms such as Ping or Traceroute [RFC7276]. In terms of "active" or "passive" OAM, In Situ OAM can be considered a hybrid OAM type [RFC7799]. In Situ mechanisms do not require extra packets to be sent. IOAM adds information to the already available data packets and, therefore, cannot be considered passive. In terms of the classification given in [RFC7799], IOAM could be portrayed as Hybrid Type I. IOAM mechanisms can be leveraged where mechanisms using, e.g., ICMP do not apply or do not offer the desired results, such as verifying that a certain traffic flow takes a pre-defined forwarding path, Service Level Agreement (SLA) verification for the data traffic, detailed statistics on traffic distribution paths in networks that distribute traffic across multiple paths, or scenarios in which probe traffic is potentially handled differently from regular data traffic by the network devices.

IOAM is deployed inside an IOAM-Domain. An IOAM-Domain is a set of nodes that use IOAM and is bounded by its edges. It is expected that all nodes in an IOAM-Domain are managed by the same administrative entity, that has means to select, monitor, and control access to all the networking devices. Per [RFC9197], IOAM-Data-Fields are carried in the clear within packets and there are no protections against any device tampering with the data. IOAM-Data-Fields collected in an untrusted environment require at least integrity protection to support critical operational decisions. Refer to [RFC9197] for more details on IOAM-Domains.

Since arbitrary nodes can tamper with all packets data, including IOAM-Data-Fields, and the packets are (in general) processed by other intermediary nodes before they are delivered to a node that can verify the IOAM fields of the packet, there is little value in attempting to use cryptographic mechanisms to prevent such modifications to the IOAM fields in the packet. Instead, this document is limited to the "detectability problem", namely, to allow an endpoint to detect that such modification has occurred since the generation of the IOAM-Data-Fields. In addition, the following considerations and requirements are to be taken into account in constructing an IOAM integrity mechanism:

1. IOAM data is processed by the data plane, hence viability of any method to prove integrity of the IOAM-Data-Fields must be feasible at data plane processing/forwarding rates (IOAM may be applied to all traffic that a router forwards).

2. IOAM data is carried within packets. Additional space required to prove integrity of the IOAM-Data-Fields should be optimal, i.e., should not exceed the Maximum Transmission Unit (MTU) inside the IOAM-Domain or have adverse effect on packet processing. Specifically, fragmentation should be avoided.

3. Protection against replay of old IOAM data should be possible. Without replay protection, a rogue node can present the old IOAM data, masking any ongoing network issues/activity and making the IOAM-Data-Fields collection useless.

This document defines a method to protect the integrity of IOAM-Data-Fields for Intra-IOAM-Domain use cases, using the IOAM Option-Types specified in [RFC9197] as an example. The method will similarly apply to future IOAM Option-Types.

2. Conventions

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Terminology

This document uses terms such as IOAM-Data-Fields, IOAM-Domain, IOAM-Namespace, IOAM-Option-Type (or Option-Type), encapsulating node, transit node, and decapsulating node. Refer to [RFC9197] for definitions.

This document also introduces the term "Validator". Refer to Section 5.6 for the definition.

The following abbreviations are used in this document:

OAM: Operations, Administration, and Maintenance

IOAM: In Situ OAM

POT: Proof of Transit

E2E: Edge to Edge

MTU: Maximum Transmission Unit

GCM: Galois/Counter Mode

GMAC: Galois Message Authentication Code

IV: Initialization Vector

ICV: Integrity Check Value

AAD: Additional Authenticated Data

The following notation is used in this document:

A || B The concatenation of A and B, where the octets of B immediately follow the octets of A.

3. Threat Analysis

This section presents a threat analysis of integrity-related threats in the context of IOAM. The threats that are discussed are assumed to be independent of the lower layer protocols; it is assumed that threats at other layers are handled by security mechanisms that are deployed at those layers.

This document is focused on integrity protection for IOAM-Data-Fields. Thus, the threat analysis includes threats that are related to or result from compromising the integrity of IOAM-Data-Fields. Other security aspects such as confidentiality are not within the scope of this document.

Throughout the analysis there is a distinction between on-path and off-path attackers. As discussed in [RFC9055], on-path attackers are located in a position that allows interception, modification, or dropping of in-flight protocol packets, whereas off-path attackers can only attack by generating protocol packets. Since IOAM operates within IOAM-Domains, this reduces potential attack vectors and should naturally mitigate off-path threats.

The analysis also includes the impact of each of the threats. Generally speaking, the impact of a successful attack on an OAM protocol [RFC7276] is an illusion of nonexistent failures (or disruption) or preventing the detection of actual ones; in both cases, the attack could result in Denial-of-Service (DoS). Furthermore, creating the illusion of a nonexistent issue could trigger unnecessary processing in some of the IOAM nodes along a forwarding path, and could cause more IOAM-related data to be exported to the management plane than is conventionally necessary. Beyond these general impacts, threat-specific impacts are discussed in each of the subsections below.

3.1. Modification: IOAM-Data-Fields

Applicability

On-path only.

Threat

An attacker can modify the IOAM-Data-Fields of in-transit packets. The modification can either be applied to all packets or selectively applied to a subset. Maliciously modified IOAM-Data-Fields can, for example, mislead network diagnostics, result in incorrect network performance metrics, or could misguide network optimization efforts.

Impact

By systematically modifying the IOAM-Data-Fields of some or all of the in-transit packets, an attacker can create a fake picture of the network status. Potential consequences include an impact on network performance, a change in the recorded forwarding path of packets, either based on fake node positions or fake data provided by the attacker to fool the system that ingests IOAM-Data-Fields.

3.2. Modification: IOAM Option-Type header

Applicability

On-path only.

Threat

An attacker can modify the fields of an IOAM Option-Type header to change or disrupt the behavior of nodes processing IOAM-Data-Fields along the path, or change the interpretation of IOAM-Data-Fields. The modification can either be applied to all packets or selectively applied to a subset.

Impact

Changing the fields of an IOAM Option-Type header might have several implications. The following list of examples is not exhaustive. An attacker could maliciously increase the processing overhead in nodes that process IOAM-Data-Fields and increase the on-the-wire overhead of IOAM-Data-Fields, by modifying the IOAM Trace-Type field ([RFC9197], Section 4.4.1) in the IOAM Trace Option-Type header. An attacker could also prevent some of the nodes that process IOAM-Data-Fields from incorporating IOAM-Data-Fields, by modifying the RemainingLen field ([RFC9197], Section 4.4.1) in the IOAM Trace Option-Type header. Another possibility for the attacker is to change the definition or interpretation of IOAM-Data-Fields by modifying the Namespace-ID field ([RFC9197], Section 4.3), which is common to all IOAM Option-Type headers. Without the right context (i.e., Namespace-ID), IOAM-Data-Fields cannot be reliably interpreted, just like data without metadata. An attacker could also cause a Denial-of-Service by setting the Loopback flag ([RFC9322], Section 3) in the IOAM Trace Option-Type header so that copies of packets are sent back by each node to the encapsulating node. Note that the modification of the header can cause impacts similar to those described in Section 3.1.

3.3. Injection: IOAM-Data-Fields

Applicability

On-path only.

Threat

An attacker can inject additional IOAM-Data-Fields into packets containing at least one IOAM Option-Type, thus falsifying the view of the actual network state. The injection can either be applied to all packets or selectively applied to a subset.

Impact

This attack causes impacts similar to those described in Section 3.1.

3.4. Injection: IOAM Option-Type header

Applicability

Both on-path and off-path.

Threat

An attacker can inject packets with IOAM Option-Type headers, thus manipulating other nodes that process IOAM-Data-Fields in the network.

Impact

This attack and its impacts are similar to those described in Section 3.2.

3.5. Deletion: IOAM-Data-Fields

Applicability

On-path only.

Threat

An attacker can remove IOAM-Data-Fields from packets containing at least one IOAM Option-Type, thus hiding the diagnosis of some nodes. The deletion can either be applied to all packets or selectively applied to a subset.

Impact

This attack causes impacts similar to those described in Section 3.1.

3.6. Deletion: IOAM Option-Type header

Applicability

On-path only.

Threat

An attacker can remove IOAM Option-Type headers from packets, thus preventing the use of IOAM to diagnose the network. The deletion can either be applied to all packets or selectively applied to a subset. The mechanisms in this document do not provide any mitigation against this threat.

Impact

By systematically removing IOAM Option-Type headers from some or all of the in-transit packets, an attacker can make telemetry recording incomplete or even impossible. As a consequence, network diagnosis could be incomplete or non-existent.

3.7. Replay

Applicability

Both on-path and off-path.

Threat

In addition to replaying old packets in general, an attacker can replay packets with IOAM-Data-Fields. Specifically, an attacker could replay a previously transmitted IOAM Option-Type header with a new data packet, therefore attaching old IOAM-Data-Fields to a fresh user packet.

Impact

This attack causes impacts similar to those described in Section 3.1.

3.8. Management and Exporting

Applicability

Both on-path and off-path.

Threat

Attacks that compromise the integrity of IOAM-Data-Fields can be applied at the management plane, e.g., by manipulating network management packets. Furthermore, the integrity of IOAM-Data-Fields that are exported to a receiving entity can also be compromised. Management plane attacks are not within the scope of this document; the network management protocol is expected to include inherent security capabilities. The integrity of exported data is also not within the scope of this document. It is expected that the specification of the export format will discuss the relevant security aspects.

Impact

Malicious manipulation of the management protocol can cause nodes that process IOAM-Data-Fields to malfunction, to be overloaded, or to incorporate unnecessary IOAM-Data-Fields into user packets. The impact of compromising the integrity of exported IOAM-Data-Fields is similar to the impacts of previous threats that were described in this section.

3.9. Delay

Applicability

On-path only.

Threat

An attacker might delay some or all of the in-transit packets that include IOAM-Data-Fields to create an illusion of congestion. Delay attacks are well known in the context of deterministic networks [RFC9055] and time synchronization [RFC7384], and could be somewhat mitigated in these environments by using redundant paths in a way that is resilient to an attack along one of the paths. This approach does not address the threat in the context of IOAM, as it does not meet the requirement to measure a specific path or to detect a problem along the path. Note that the mechanisms in this document do not attempt to provide any mitigation against this threat.

Impact

Since IOAM can be applied to a fraction of the traffic, an attacker can detect and delay only the packets that include IOAM-Data-Fields, thus preventing the authenticity of delay and load measurements.

3.10. Threat Summary

Threat	Document Scope: In	Document Scope: Out
Modification: IOAM-Data-Fields	X	
Modification: IOAM Option-Type header	X	
Injection: IOAM-Data-Fields	X	
Injection: IOAM Option-Type header	X	
Deletion: IOAM-Data-Fields	X	
Deletion: IOAM Option-Type header		X
Replay	X	
Management and Exporting		X
Delay		X

Figure 1: Threat Analysis Summary

4. Integrity-Protected Option-Types

This section defines new IOAM Option-Types to carry IOAM-Data-Fields with integrity protection. For each IOAM Option-Type defined in [RFC9197], a corresponding Integrity-Protected Option-Type is defined as follows:

IOAM Integrity-Protected Pre-allocated Trace Option-Type: corresponds to the IOAM Pre-allocated Trace Option-Type ([RFC9197], Section 4.4) with integrity protection.

IOAM Integrity-Protected Incremental Trace Option-Type: corresponds to the IOAM Incremental Trace Option-Type ([RFC9197], Section 4.4) with integrity protection.

IOAM Integrity-Protected POT Option-Type: corresponds to the IOAM POT Option-Type ([RFC9197], Section 4.5) with integrity protection.

IOAM Integrity-Protected E2E Option-Type: corresponds to the IOAM E2E Option-Type ([RFC9197], Section 4.6) with integrity protection.

The Direct Export (DEX) Option-Type [RFC9326] is not covered by the Integrity Protection Method defined in Section 5. This document focuses on the integrity protection of IOAM-Data-Fields, whereas DEX does not carry IOAM-Data-Fields by definition. As a consequence, DEX and similar (i.e., any IOAM Option-Type with no IOAM-Data-Fields) are considered out of scope and MUST NOT use the Integrity Protection Method defined in this document.

The Integrity Protection header sits between an IOAM Option-Type header and its IOAM-Data-Fields, forming an equivalent Integrity-Protected Option-Type. The Integrity Protection header is defined as shown in Figure 2.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Method ID   |  Nonce Length |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                             Nonce                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                  Integrity Check Value (ICV)                  ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Integrity Protection header

Method ID: 8-bit unsigned integer, see Section 6.2. It defines the Integrity Protection Method to compute the Integrity Check Value (ICV) field. If a node encounters an unknown value, it MUST NOT change the contents of the Integrity Protection header and MUST NOT change the contents of the IOAM-Data-Fields. In other words, the node must not process the IOAM Option-Type.

Nonce Length: 8-bit unsigned integer. It defines the length of the Nonce field, in octets.

Reserved: 16-bit Reserved field. It MUST be set to zero upon transmission and ignored upon receipt.

Nonce: Variable length field. Its size depends on the Nonce Length field.

Integrity Check Value (ICV): Variable length field. Its size depends on the Method ID field.

In order to keep IOAM-Data-Fields aligned ([RFC9197], Section 4.4.2), the total length of the Integrity Protection header MUST be a multiple of 4 octets.

4.1. Integrity-Protected Trace Option-Types

Both the IOAM Pre-allocated Trace Option-Type header and the IOAM Incremental Trace Option-Type header are identical, as defined in Section 4.4 of [RFC9197]. When followed by the Integrity Protection header, they respectively form the IOAM Integrity-Protected Pre-allocated Trace Option-Type and the IOAM Integrity-Protected Incremental Trace Option-Type (Figure 3). The definitions of fields that are not part of the Integrity Protection header are the same as in [RFC9197].

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Namespace-ID          | NodeLen | Flags | RemainingLen|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               IOAM Trace-Type                 |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Method ID   |  Nonce Length |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                             Nonce                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                  Integrity Check Value (ICV)                  ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        IOAM-Data-Fields                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Integrity-Protected Trace Option-Types header

4.2. Integrity-Protected POT Option-Type

The IOAM POT Option-Type header is defined in Section 4.5 of [RFC9197]. When followed by the Integrity Protection header, it forms the IOAM Integrity-Protected POT Option-Type (Figure 4). The definitions of fields that are not part of the Integrity Protection header are the same as in [RFC9197].

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Namespace-ID          | IOAM-POT-Type | IOAM-POT-Flags|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Method ID   |  Nonce Length |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                             Nonce                             ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                  Integrity Check Value (ICV)                  ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        IOAM-Data-Fields                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Integrity-Protected POT Option-Type header

4.3. Integrity-Protected E2E Option-Type

The IOAM E2E Option-Type header is defined in Section 4.6 of [RFC9197]. When followed by the Integrity Protection header, it forms the IOAM Integrity-Protected E2E Option-Type (Figure 5). The definitions of fields that are not part of the Integrity Protection header are the same as in [RFC9197].

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Namespace-ID         |         IOAM-E2E-Type         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   Method ID   |  Nonce Length |           Reserved            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                             Nonce                             ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                  Integrity Check Value (ICV)                  ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                        IOAM-Data-Fields                       ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Integrity-Protected E2E Option-Type header

5. Integrity Protection Method

The Integrity Protection Method defined in this document leverages symmetric keys for the integrity protection of IOAM-Data-Fields. This method uses AES-GMAC [AES] [NIST.800-38D], a block cipher mode of operation providing data origin authentication, which is also a specialization of the Galois/Counter Mode (GCM). The GCM authenticated encryption operation has four inputs: a secret key, an Initialization Vector (IV), a plaintext, and Additional Authenticated Data (AAD). It has two outputs: a ciphertext whose length is identical to the plaintext and an Authentication Tag. GMAC is the special case of GCM in which the plaintext has a length of zero. Therefore, the empty ciphertext output is ignored, and the only output is the Authentication Tag. For this method, the Authentication Tag MUST NOT be truncated, meaning its size MUST always be 16 octets (i.e., a full Authentication Tag). Below, the GMAC Initialization Vector (IV) is referred to as the nonce, the GMAC Authentication Tag is referred to as the Integrity Check Value (ICV), and the GMAC Additional Authenticated Data (AAD) is referred to as the AAD.

5.1. Key and Nonce Management

In order to use this method and apply integrity protection, it is REQUIRED that each IOAM node that updates the ICV (in the Integrity Protection header) has its own unique symmetric key. Although GMAC supports all AES key sizes (i.e., 128, 192, and 256 bits), it is RECOMMENDED to use the longest key size when possible. Each key MUST be securely generated and fresh. Also, each key MUST be securely distributed to only the corresponding IOAM nodes and any Validator that needs to validate messages protected by that key. Except key rotation requirements, the details of key generation and distribution are outside the scope of this document.

In addition to key management, per-message nonces used with GMAC MUST be managed to prevent reuse of a key-nonce pair. Since reuse of a nonce with a given key allows forgery of arbitrary ciphertexts with valid authentication tag, it is extremely important to have high confidence in nonce non-reuse.

For this method, the size of the nonce MUST always be 12 octets. If a node receives a Nonce Length value other than 12, it MUST NOT change the contents of the Integrity Protection header and MUST NOT change the contents of the IOAM-Data-Fields. In other words, the node must not process the IOAM Option-Type. A nonce MUST NOT be reused with the same key. The nonce is based on the "Deterministic Construction" [NIST.800-38D] and has the format shown in Figure 6.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Key ID     |             Encapsulating Node ID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                        Counter (64 bits)                      |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Structure of the Nonce field (12 octets)

Key ID: 8-bit unsigned integer. It identifies the key used by the corresponding Encapsulating Node ID to compute the ICV with GMAC.

Encapsulating Node ID: 24-bit unsigned integer. It identifies an IOAM encapsulating node that generated the nonce. It MUST be unique for each IOAM node in the IOAM-Domain, which means a reasonable limit of 2^24 distinct IOAM nodes in total.

Counter: 64-bit unsigned integer. It is an incrementing counter managed by the corresponding encapsulating node (i.e., the one whose ID is the same as the Encapsulating Node ID field). The counter MUST start at 0 for each Key ID and MUST be unique for each packet, which means a limit of 2^64 packets per Key ID on the corresponding Encapsulating Node ID. Since GMAC does not require a nonce to be secret or unpredictable, it is secure to use a counter.

When the key-specific counter of an encapsulating node reaches the maximum value (i.e., 2^64), the encapsulating node MUST stop applying integrity protection with that key and start using a new one. Ideally, before that limit is reached, the key management system SHOULD rotate the key and notify any Validator that needs to validate messages protected by that key. The Key ID field provides an easy way to avoid conflicts during key rotations with other IOAM nodes that apply integrity protection. When an encapsulating node reaches its maximum use of 256 distinct keys (i.e., see the Key ID field) and is also about to reach the limit of its key-specific counter, the key management system MUST rotate the keys of all IOAM nodes in the IOAM-Domain and notify any Validator with the new keys. In addition, the internal integrity protection state of all IOAM nodes, which is useful to detect and prevent key-nonce reuse, MUST be reset. Overall, the key management system MUST NOT (ever) redistribute an old key to any of the IOAM nodes in the IOAM-Domain.

An encapsulating node that participates in the integrity protection of IOAM-Data-Fields MAY retain on persistent storage the current key in use and the corresponding key-specific counter, such that a power interruption or system crash (or reboot in general) does not lead to nonce reuse. If it is not possible for some reason, the encapsulating node MUST NOT reuse the key and the key management system MUST rotate the key before the encapsulating node resumes integrity protection. In case of a power interruption or system crash (or reboot in general) on any transit node that participates in the integrity protection of IOAM-Data-Fields, the transit node MUST NOT reuse the key and the key management system MUST rotate the key before the transit node resumes integrity protection.

To illustrate with an example, it would take 7 years for a single key of an encapsulating node to reach the limit of 2^64 1500-byte packets on a 1 Pbps (Petabits per second) link at line rate, while it would take approximately 143 days with 64-byte packets. For 256 distinct keys of an encapsulating node, it would take 1792 and 100 years respectively. This is a worst-case scenario, as such link capacity is not common and IOAM might not be applied to all packets.

5.2. Integrity Protection of Header Fields

The main objective of the Integrity Protection Method defined in this document is to provide integrity protection of IOAM-Data-Fields. However, some Option-Type header fields are crucial for IOAM-Data-Fields (e.g., the IOAM Namespace-ID, which is common to all IOAM Option-Types). Without such fields, IOAM-Data-Fields cannot be reliably interpreted. As a consequence, the integrity of any immutable Option-Type header field MUST be protected. As for mutable Option-Type header fields, they do not benefit from any integrity protection since their values can change.

With GMAC, the bit length of the AAD MUST be a multiple of 8. In other words, the AAD is made up of whole octets. Masking specific bits of Option-Type header fields allows this constraint to be respected, even for fields that are single bits (e.g., flags) or not aligned on natural boundaries. The list of Option-Type header fields and their corresponding integrity protection masks to be applied (i.e., a logical AND operation) are defined in Section 6.2.

For interoperability, having a dynamic registry to specify which header fields participate in integrity protection might not seem optimal at first glance. However, from a vendor perspective, integrity protection is simply a feature built on top of IOAM. Therefore, if two different vendors providing IOAM integrity protection are not interoperable, it probably means that their core IOAM implementations are not interoperable in the first place. Since IOAM has no versions, it does not make sense to introduce versions in the integrity protection feature either. This is the reason why it is acceptable to have a dynamic registry for this purpose and to update it accordingly over time.

5.3. Encapsulating Node

The encapsulating node MUST follow the instructions in Section 5.1 to generate a new nonce, which is then stored in the Nonce field of the Integrity Protection header (the Nonce Length field is set accordingly). The Method ID field MUST be set to 0, as defined in Section 6.2.

The ICV is the result of a GMAC operation over a list of immutable header fields as defined in Section 5.2 (depending on the Integrity-Protected Option-Type) and immutable IOAM-Data-Fields added by the encapsulating node. In the case the Integrity-Protected Pre-allocated Trace Option-Type is used, the encapsulating node includes the IOAM-Data-Fields that correspond to itself, i.e., "node data list [n]" ([RFC9197], Section 4.4). The encapsulating node performs the following operations:

* AAD = (Header Fields || IOAM-Data-Fields)

* ICV = GMAC(Key, Nonce, AAD)

Fields in the AAD are encoded in network byte order and in the same sequence as they appear on the wire. The encapsulating node then stores the ICV in the corresponding field of the Integrity Protection header.

5.4. Transit Node

A transit node MUST NOT generate a new nonce. Instead, it MUST use the received Nonce field for its own GMAC operation. As a consequence, a transit node MUST be able to detect nonces already used with its current key to prevent key-nonce pair reuse, which is critical for GMAC. Details on how this detection is implemented are outside the scope of this document. If a transit node receives a Nonce field that has already been used with its current key, it MUST NOT change the contents of the Integrity Protection header and MUST NOT change the contents of the IOAM-Data-Fields. In other words, the transit node must not process the IOAM Integrity-Protected Option-Type.

The ICV is the result of a GMAC operation over the received ICV field and immutable IOAM-Data-Fields added by the transit node. In the case the Integrity-Protected Pre-allocated Trace Option-Type is used, the transit node includes the IOAM-Data-Fields that correspond to itself, i.e., "node data list [n-i]" ([RFC9197], Section 4.4). The transit node performs the following operations:

* AAD = (ICV || IOAM-Data-Fields)

* ICV = GMAC(Key, Nonce, AAD)

Fields in the AAD are encoded in network byte order and in the same sequence as they appear on the wire. The transit node then updates the ICV field accordingly in the Integrity Protection header.

If the transit node does not add any immutable IOAM-Data-Fields (e.g., it only modifies mutable IOAM-Data-Fields or does nothing), and if the transit node, in case the Integrity-Protected Pre-allocated Trace Option-Type is used, does not update the "node data list" array, then the transit node MUST NOT update the ICV field in the Integrity Protection header.

A transit node MUST NOT add or remove the Integrity Protection header. Also, a transit node MUST NOT modify the Method ID field, the Nonce Length field, or the Nonce field in the Integrity Protection header.

5.5. Decapsulating Node

The decapsulating node MAY perform the function of the Validator. If it does, please refer to Section 5.6.

If the decapsulating node does not perform the function of the Validator, which is an alternative to put any Validator out of the forwarding path in case of performance concerns, the decapsulating node MUST send the entire IOAM Integrity-Protected Option-Type to a Validator. The method to send it to a Validator is out of scope for this document. Before that, the decapsulating node updates the ICV field in the Integrity Protection header. The ICV is the result of a GMAC operation over the received ICV field and immutable IOAM-Data-Fields added by the decapsulating node. In the case the Integrity-Protected Pre-allocated Trace Option-Type is used, the decapsulating node includes the IOAM-Data-Fields that correspond to itself, i.e., "node data list [n-i]" ([RFC9197], Section 4.4). Since the decapsulating node MUST use the received Nonce field for its own GMAC operation, it MUST be able to detect nonces already used with its current key to prevent key-nonce pair reuse, which is critical for GMAC. Details on how this detection is implemented are outside the scope of this document. If a decapsulating node receives a Nonce field that has already been used with its current key, it MUST NOT change the contents of the Integrity Protection header and MUST NOT change the contents of the IOAM-Data-Fields. In other words, the decapsulating node must not process the IOAM Integrity-Protected Option-Type. Otherwise, the decapsulating node performs the following operations:

* AAD = (ICV || IOAM-Data-Fields)

* ICV = GMAC(Key, Nonce, AAD)

Fields in the AAD are encoded in network byte order and in the same sequence as they appear on the wire.

If the decapsulating node does not add any immutable IOAM-Data-Fields (e.g., it only modifies mutable IOAM-Data-Fields or does nothing), and if the decapsulating node, in case the Integrity-Protected Pre-allocated Trace Option-Type is used, does not update the "node data list" array, then the decapsulating node MUST NOT update the ICV field in the Integrity Protection header.

The decapsulating node MUST NOT add the Integrity Protection header. Also, the decapsulating node MUST NOT modify the Method ID field, the Nonce Length field, or the Nonce field in the Integrity Protection header.

5.6. Validator

An IOAM node that performs the validation of the integrity protection is referred to as a Validator. Any Validator is a key trusted entity in this system, as it has access to all of the keying material in use and makes the final determination of whether an ICV is valid.

A validator is not vulnerable to key-nonce reuse, since the computed ICV remains internal. However, a protection against replay attacks in general (more specifically, replay of old IOAM-Data-Fields) is still needed. To this end, a Validator MUST be able to detect nonces already used with specific keys during validation to prevent replays. Details on how this detection is implemented are outside the scope of this document. If a Validator receives a Nonce field that has already been used with a specific key during validation, it MUST consider the ICV as invalid and ignore the next steps.

To validate an ICV, a Validator MUST recompute it by iteratively following the previous steps (in the same order) of this Integrity Protection Method, using the respective symmetric keys received previously. The recomputed ICV is then compared to the received ICV field. As a result, a Validator can perform an integrity check on the IOAM-Data-Fields and report whether any modification is detected. The validation is one-step in some cases (e.g., with POT Type-0 or E2E), where only the encapsulating node updates the ICV according to the definition of this method. For other cases where transit nodes also update the ICV (e.g., with Trace Option-Types), a Validator MUST identify these transit nodes to look up their respective keys. For that, a unique identifier of the node, such as the "node_id" ([RFC9197], Section 4.4.2) for Trace Option-Types, MUST be included in IOAM-Data-Fields. Regardless of the Option-Type, the Nonce field allows the encapsulating node to be identified (Section 5.1). Details on how the mapping between those identifiers and keys is implemented on a Validator are outside the scope of this document.

6. IANA Considerations

6.1. IOAM Option-Types

IANA is requested to add the following new code points in the "IOAM Option-Type" registry available at [IANA-IOAM]:

Code Point: (suggested) 64

Name: IOAM Integrity-Protected Pre-allocated Trace Option-Type

Description: Pre-allocated Trace with Integrity Protection

Reference: This document, Section 4

Code Point: (suggested) 65

Name: IOAM Integrity-Protected Incremental Trace Option-Type

Description: Incremental Trace with Integrity Protection

Reference: This document, Section 4

Code Point: (suggested) 66

Name: IOAM Integrity-Protected POT Option-Type

Description: POT with Integrity Protection

Reference: This document, Section 4

Code Point: (suggested) 67

Name: IOAM Integrity-Protected E2E Option-Type

Description: E2E with Integrity Protection

Reference: This document, Section 4

New IOAM Integrity-Protected Option-Types that intend to use the Integrity Protection Method defined in this document will update the "IOAM Integrity Protection Methods" registry (Section 6.2), more specifically the "Protected Header Fields" column of this Integrity Protection Method, to specify the list of corresponding Option-Type header fields that participate in the integrity protection of IOAM-Data-Fields. Section 5.2 discusses the motivations and choices for protecting the integrity of Option-Type header fields in addition to IOAM-Data-Fields.

6.2. IOAM Integrity Protection Methods

IANA is requested to define a new registry named "IOAM Integrity Protection Methods", under the "In Situ OAM (IOAM)" registry group [IANA-IOAM].

This new registry defines 256 code points to identify different IOAM Integrity Protection Methods. The following initial code points are defined:

ID	Description	Protected Header Fields	Reference
0x00	AES-GMAC, 16-octet (full) Authentication Tag, 12-octet Initialization Vector.	Pre-allocated Trace and Incremental Trace: - Namespace-ID (mask = 0xffff) - NodeLen + Flags + RemainingLen (mask = 0xfb00) - IOAM Trace-Type (mask = 0xffffff) - Reserved (mask = 0x00) POT: - Namespace-ID (mask = 0xffff) - IOAM-POT-Type (mask = 0xff) IOAM-POT-Flags (mask = 0x00) E2E: - Namespace-ID (mask = 0xffff) - IOAM-E2E-Type (mask = 0xffff)	This document, Section 5
0x01 - 0xFE	Unassigned		
0xFF	Reserved		This document

Figure 7: IOAM Integrity Protection Methods

Code points 1-254 are available for assignment via the "IETF Review" process, as per [RFC8126].

New registration requests must use the following template: the value of the requested code point, a description of the Integrity Protection Method, the list of header fields with integrity protection masks for all supported Option-Types, and a reference to the document (and, optionally, the section) that defines the new Integrity Protection Method.

7. Operational Considerations

7.1. Manageability

[I-D.ietf-ippm-ioam-integrity-yang] specifies a YANG module to manage IOAM profiles with integrity protection.

7.2. Scalability and Performance

There is an additional per-packet processing for each node that uses the Integrity Protection Method defined in this document, in particular for any Validator with Integrity-Protected Option-Types where transit nodes participate in integrity protection (e.g., Integrity-Protected Trace Option-Types). Inappropriate use of this Integrity Protection Method could overload nodes and cause service degradation or failure. Therefore, relevant metrics (e.g., CPU and memory utilization) SHOULD be monitored to detect misbehaving implementations. Operators deploying IOAM with this Integrity Protection Method MUST ensure that such overload situations are avoided. This might, for example, be achieved by applying IOAM only to a subset of the entire traffic, keeping in mind that only that IOAM subset would be integrity protected. In addition, integrity protection workload could be distributed across multiple Validators, enabling validation jobs to be parallelized and the processing load to be shared.

7.3. Migration

Option-Types defined in [RFC9197] and Integrity-Protected Option-Types defined in this document are not mutually exclusive. They can both coexist inside an IOAM-Domain, as integrity protection is seen as a feature running on top of IOAM. For example, an implementation might support the simultaneous configuration of an IOAM-Namespace with the Pre-allocated Trace Option-Type and another IOAM-Namespace with the Integrity-Protected Pre-allocated Trace Option-Type.

7.4. MTU

[RFC9197] discusses MTU considerations, as IOAM data is carried within packets. Similarly, additional space required to prove integrity of the IOAM-Data-Fields SHOULD be optimal, i.e., should not exceed the MTU inside the IOAM-Domain or have adverse effect on packet processing. Specifically, fragmentation should be avoided.

8. Security Considerations

Section 3 provides a threat analysis of integrity-related threats in the context of IOAM.

The Integrity Protection Method defined in this document (Section 5) leverages symmetric keys and uses AES-GMAC [AES] [NIST.800-38D]. Security considerations regarding key and nonce management are discussed in Section 5.1.

Packet reordering or duplication does not compromise the safety of the Integrity Protection Method defined in this document, as key-nonce reuse is not allowed. However, reordered or duplicated packets are not considered for integrity protection, resulting in no IOAM data being inserted for those packets. This behavior depends on the replay protection window implemented by a node, which determines the tolerance for packet reordering.

A compromised transit node could remove the Integrity Protection header and replace the IOAM Integrity-Protected Option-Type with the unprotected analogue IOAM Option-Type, in order to be able to modify IOAM-Data-Fields and bypass the Validator. A compromised IOAM transit node could also reinitialize both the Nonce and ICV fields in the Integrity Protection header, in order to pretend to be an encapsulating node and fool the Validator. To avoid such situations, any Validator MUST know all IOAM Namespace-IDs for which the integrity protection is enabled. For each of them, any Validator MUST know the corresponding encapsulating nodes and, for each encapsulating node, which Option-Types are added. When enabled, the integrity protection MUST be applied to the entire corresponding IOAM set, not a subset. Implementation details are outside the scope of this document. Also, as discussed in Section 3.6, a compromised transit node could entirely remove IOAM Option-Types. This document does not provide any mitigation against this threat as it is out of scope, and such situations SHOULD be handled by the security mechanisms of another layer.

A compromised Validator could use specific keys to forge or modify IOAM-Data-Fields, as if it passed through the encapsulating or transit nodes in question. It could also render incorrect assessments of an ICV’s (in)validity. Since a Validator is a key trusted entity in this integrity protection system, there is no recourse to prevent such cases. In contrast, compromised encapsulating or transit nodes could forge or drop packets they process but cannot impersonate other IOAM nodes or modify integrity protected IOAM-Data-Fields produced by other nodes without being detected by a Validator. In particular, a transit node is limited in what forgery can be made without detection because a Validator will validate the encapsulating node’s ICV as part of validating the final ICV, thus modification to content protected by the encapsulating node would be detected at the time of validation.

The Integrity Protection Method defined in this document is intended for Intra-IOAM-Domain use cases (i.e., no confidentiality, integrity protection only). For Inter-IOAM-Domain use cases, operators can use IPSec to securely transfer IOAM-Data-Fields between IOAM-Domains.

9. Acknowledgements

The authors would like to thank Santhosh N, Rakesh Kandula, Saiprasad Muchala, Al Morton, Greg Mirsky, Benjamin Kaduk, Mehmet Beyaz, Martin Duke, Tianran Zhou, Giuseppe Fioccola, Mohamed Boucadair, and Yoshifumi Nishida for their comments and advice.

10. References

10.1. Normative References

[AES] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[NIST.800-38D] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, 2007, <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9197] Brockners, F., Ed., Bhandari, S., Ed., and T. Mizrahi, Ed., "Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9197, DOI 10.17487/RFC9197, May 2022, <https://www.rfc-editor.org/info/rfc9197>.

10.2. Informative References

[I-D.ietf-ippm-ioam-integrity-yang] Iurman, J. and T. Zhou, "A YANG Data Model for In Situ Operations, Administration, and Maintenance (IOAM) Integrity-Protected Options", Work in Progress, Internet-Draft, draft-ietf-ippm-ioam-integrity-yang-07, 22 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ioam-integrity-yang-07>.

[I-D.ietf-opsawg-oam-characterization] Pignataro, C., Farrel, A., and T. Mizrahi, "Guidelines for Characterizing the Term "OAM"", Work in Progress, Internet-Draft, draft-ietf-opsawg-oam-characterization-17, 28 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-oam-characterization-17>.

[IANA-IOAM] IANA, "In Situ OAM (IOAM)", <https://www.iana.org/assignments/ioam/ioam.xhtml>.

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, DOI 10.17487/RFC7276, June 2014, <https://www.rfc-editor.org/info/rfc7276>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <https://www.rfc-editor.org/info/rfc7799>.

[RFC9055] Grossman, E., Ed., Mizrahi, T., and A. Hacker, "Deterministic Networking (DetNet) Security Considerations", RFC 9055, DOI 10.17487/RFC9055, June 2021, <https://www.rfc-editor.org/info/rfc9055>.

[RFC9322] Mizrahi, T., Brockners, F., Bhandari, S., Gafni, B., and M. Spiegel, "In Situ Operations, Administration, and Maintenance (IOAM) Loopback and Active Flags", RFC 9322, DOI 10.17487/RFC9322, November 2022, <https://www.rfc-editor.org/info/rfc9322>.

[RFC9326] Song, H., Gafni, B., Brockners, F., Bhandari, S., and T. Mizrahi, "In Situ Operations, Administration, and Maintenance (IOAM) Direct Exporting", RFC 9326, DOI 10.17487/RFC9326, November 2022, <https://www.rfc-editor.org/info/rfc9326>.

Authors’ Addresses

Frank Brockners
Cisco Systems, Inc.
Hansaallee 249, 3rd Floor
40549 DUESSELDORF
Germany
Email: fbrockne@cisco.com

Shwetha Bhandari
Databricks
Angkor West Building, Bagmane Capital Tech Park, Ferns City
Doddanekkundi, Mahadevpura, Bengaluru, Karnataka 560048
India
Email: shwetha.bhandari@databricks.com

Tal Mizrahi
Huawei
8-2 Matam
Haifa 3190501
Israel
Email: tal.mizrahi.phd@gmail.com

Justin Iurman
University of Liege
10, Allee de la decouverte (B28)
4000 Sart-Tilman
Belgium
Email: justin.iurman@uliege.be