Abstract

This document defines Asymmetric Manifest-Based Integrity (AMBI). AMBI allows each receiver of a stream of multicast packets to check the integrity of the contents of each packet in the data stream. AMBI operates by passing cryptographically verifiable manifests for the data packets, over out-of-band communication channels.

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 https://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."

This Internet-Draft will expire on October 28, 2019.

Copyright Notice

Copyright (c) 2019 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 (https://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.
1. Introduction

Multicast transport poses security problems that are not easily addressed by the same security mechanisms used for unicast transport.

The "Introduction" sections of the documents describing TESLA [RFC4082], and TESLA in SRTP [RFC4383], and TESLA with ALC and NORM [RFC5776] present excellent overviews of the challenges unique to multicast authentication, briefly summarized here:

- A MAC based on a symmetric shared secret cannot be used because each packet has multiple receivers that do not trust each other.

- Asymmetric per-packet signatures can handle only very low bit-rates because of the computational overhead.

- An asymmetric signature of a larger message comprising multiple packets requires reliable receipt of all such packets, something that cannot be guaranteed in a timely manner even for protocols that do provide reliable delivery, and the retransmission of which
may anyway exceed the useful lifetime for data formats that can otherwise tolerate some degree of loss.

Asymmetric Manifest-Based Integrity (AMBI) specifies a method for receivers or middle boxes to cryptographically authenticate and verify the integrity of a stream of packets, by communicating packet "manifests" (described in Section 2.3) via an out-of-band communication channel that provides authentication and verifiable integrity.

Each manifest contains cryptographic hashes of packet payloads corresponding to specific packets in the authenticated data stream.

Each manifest MUST be delivered in a manner that provides cryptographic integrity of the manifest. For example, TLS or IPSec may be used to deliver a stream of manifests with unicast from a trusted sender to many receivers, or another mechanism that provides authentication for a multicast stream, such as a protocol which signs each packet, could be used to provide authentication for the manifests.

Upon successful verification of the contents of a manifest and receipt of any subset of the corresponding data packets, the receiver has proof of the integrity of the contents of the data packets listed in the manifest.

An "anchor message", described in Section 2.2, provides the link between an authenticated data stream and the out-of-band channel of manifests that authenticates it. This document defines a DNS-based method for a sender to advertise a URI that can be used to retrieve the anchor message over a secure transport. The anchor message MAY also be provided by other out-of-band mechanisms that provide integrity guarantees for the anchor message. Describing alternate methods is out of scope for this document.

Authenticating the integrity of the data packets depends on:

- authentication of the anchor message that provides the linkage between the manifest channel and the data stream; and

- the secrecy and cryptographic strength of private keys used for signing manifests, or the authentication of the secure channels used for transmitting manifests; and

- the difficulty of generating a collision for the packet hashes in the manifest.
1.1. Comparison with TESLA

AMBI and TESLA [RFC4082] and [RFC5776] attempt to achieve a similar goal of authenticating the integrity of streams of multicast packets. AMBI imposes a higher overhead, as measured in the amount of extra data required, than TESLA imposes. In exchange, AMBI provides non-repudiation (which TESLA does not), and relaxes the requirement for establishing an upper bound on clock synchronization between sender and receiver.

This tradeoff enables new capabilities for AMBI, relative to TESLA. In particular, when receiving multicast traffic from an untrusted transit network, AMBI can be used by a middle box to authenticate packets from a trusted source before forwarding traffic through the network, and the receiver also can separately authenticate the packets. (This use case is not possible with TESLA because the data packets can’t be authenticated until a key is disclosed, so either the middlebox has to forward data packets without first authenticating so that the receiver has them prior to key disclosure, or the middlebox has to hold packets until the key is disclosed, at which point the receiver can no longer establish their authenticity.)

1.2. Terminology

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 [RFC2119].

2. Protocol Specification

2.1. Packet Identifiers

2.1.1. Overview

Packet identifiers are a sequence number contained within the authenticated payload of the packet. This sequence number is used inside a manifest to associate each packet hash with a specific packet. Each authenticated packet MUST contain a packet identifier. See Section 4.1 for a discussion of the security implications.

This document defines a new UDP option in Section 2.1.4 for use as a packet identifier.

Some multicast-capable transport protocols have a sequence number embedded in data packets in the protocol. The sequence numbers in these protocols MAY be used instead of the new UDP option, to avoid introducing extra overhead in the authenticated data packets.
In Section 2.1.2, Section 2.1.3, and Section 2.1.4, this document defines some sample ways to specify packet identifiers based on such sequence numbers embedded in existing protocols.

Other appropriate sequence number systems may exist, such as the anti-replay Sequence Number field in Section 3.1 of [RFC6584], when NORM or FLUTE operates with an authentication profile that uses it (however, since that example already provides authentication, it is not added as an option in this document). The AMBI anchor message format can be extended in future documents to support those or other suitable schemes by adding values to the registry defined in Section 3.

In some deployments, in contrast to using the new UDP option, the approach of using an existing sequence number may carry a benefit because it requires no change to the stream of packets being authenticated, enabling interoperability with existing unmodified sending and receiving applications.

2.1.2. RTP Sequence Number

Sequence number from Section 5.1 of [RFC3550].

TBD: discussion of security consequences of using 16 bits—recommend a bigger hash in manifests for this case?

2.1.3. SRTP Sequence Number

Packet Index from Section 3.3.1 of [RFC3711].

2.1.4. UDP Option

Define a new UDP option [I-D.ietf-.tsvwg-udp-options] (TBD2).

```
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| UDP Kind=TBD2 |    Length=6   |      32-bit sequence number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2.2. Anchor Message
2.2.1.  Overview

An anchor message provides the information that makes it possible to associate the manifests with the data packets they authenticate. ID values that appear as text integers in the anchor message also appear in the manifest binary data, with the anchor message providing context on how to interpret the values.

An anchor message MAY be discovered and transmitted by any means which provides adequate source authentication and data integrity to meet the security needs of the receiver.

In order to support middle-box authentication, it is RECOMMENDED that senders arrange to distribute anchor messages according to the method outlined in Section 2.2.2.

2.2.2.  DNS-based Anchor URI Bootstrap

This document defines a new DNS resource record (RR) to communicate a URI for an AMBI anchor message to remote receivers of the sender's traffic, so they can use it to authenticate traffic from the sender.

The sender is the owner of the RR, and configures the zone so that it contains an RR that provides a URI that can provide secure delivery of the an anchor message appropriate to authenticate all the sender's multicast traffic.

This mechanism only works for source-specific multicast (SSM) channels. The source address of the (S,G) is reversed and used as an index into one of the reverse mapping trees (in-addr.arpa for IPv4, as described in Section 3.5 of [RFC1035], or ip6.arpa for IPv6, as described in Section 2.5 of [RFC3596]).

When a middle box or receiver processes a join for a new source, if it is configured to perform authentication on SSM multicast channels it can forward, the middle box can discover a URI to obtain the anchor message by issuing a DNS request for the AMBI record of the reverse IP of the source of the (S,G), then fetch the contents of the resulting URI, validate it, and use it to authenticate traffic from the source.

TBD: consider breaking up anchor message to avoid large, frequently changing anchors for sources with many groups.

TBD: consider graceful rollover for anchors, instead of synchronized update of anchor hash.
2.2.3. Anchor Message YANG model

The anchor message is composed of a YANG instance object that validates against the YANG model below.

<CODE BEGINS> file "ietf-ambi-anchor.yang"
module ietf-ambi-anchor {
    yang-version 1.1;

    namespace "urn:ietf:params:xml:ns:yang:ietf-ambi-anchor";
    prefix "ambi";

    import ietf-yang-types {
        prefix "yang";
        reference "RFC6991 Section 3";
    }

    import ietf-inet-types {
        prefix "inet";
        reference "RFC6991 Section 4";
    }

    import ietf-routing-types {
        prefix "rt-types";
        reference "RFC8294";
    }

    organization "IETF";

    contact
        "Author: Jake Holland
         <mailto:jholland@akamai.com>
        Author: Kyle Rose
         <mailto:krose@akamai.com>
    ";

    description
        "This module contains the definition for the AMBI anchor message data type.

        Copyright (c) 2018 IETF Trust and the persons identified as authors of the code. All rights reserved.

        Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents"
This version of this YANG module is part of
draft-jholland-mboned-ambi, [TBD: change]
see the internet draft itself for full legal notices."

revision 2018-06-27 {
  description "Initial revision.";
  reference "";
}

/* TBD: copied some from https://tools.ietf.org/html/rfc8177,
   but the model doesn't seem to match what I want. is there another
   I can import instead of making these here? Or a registry
to reference? */
identity crypto-hash {
  description
    "Base identity of cryptographic hash options. ";
}

identity sha-256 {
  base crypto-hash;
  description
    "The SHA-256 algorithm.";
}

identity blake2b {
  base crypto-hash;
  description
    "The BLAKE2b algorithm.";
}

identity crypto-signature {
  description
    "Base identity of cryptographic asymmetric signature
    options.";
}

identity ed25519 {
  base crypto-signature;
  description
    "The Ed25519 algorithm.";
}

identity rsa {
  base crypto-signature;
  description
    "Base identity of cryptographic asymmetric signature
    options.";
}
"The RSA algorithm.");
}

identity sequence-type {
    description
    "Base identity for sequence number type options.";
}

identity rtp {
    base sequence-type;
    description
    "The sequence number from RTP.";
}

identity srtp {
    base sequence-type;
    description
    "The sequence number from SRTP.";
}

identity udp {
    base sequence-type;
    description
    "The sequence number from UDP.";
}

typedef key-identifier {
    type uint16 {
        range 1..65535;
    }
    description "Key identifier within a manifest";
}

typedef bitrate {
    type string {
        pattern '[1-9][0-9]*[GMK]?bps';
    }
    description "Bit-rate of a data stream";
}

typedef packetrate {
    type string {
        pattern '[1-9][0-9]*[GMK]?pps';
    }
    description "Packet rate of a data stream";
}

typedef manifest-transport {
    type union {

type leafref {
    path "/anchor/data_stream/id";
}  

type inet:uri;

}  

description "Transport method for a manifest stream";
}

container anchor {
    container self {
        presence "An anchor message exists";
        description "Self-referential properties about the anchor message";
        leaf uri {
            type inet:uri;
            mandatory true;
            description "The canonical URI for this anchor message.";
        }
        leaf version {
            type uint16;
            mandatory true;
            description "The version number for this anchor message.";
        }
        leaf hash_algorithm {
            type identityref {
                base crypto-hash;
            }
            mandatory true;
            description "The algorithm for the anchor message hash provided in a manifest.";
        }
        leaf hash_bits {
            type uint16;
            mandatory true;
            description "The number of bits for the anchor’s hash provided in a manifest.";
        }
        leaf expires {
            type yang:date-and-time;
            mandatory true;
            description "The expiration time for this anchor message.";
        }
    }
}
description "Anchor message for AMBI"

list public_key {
  key id;
  description "Public key for ALTI signatures.";
  leaf id {
    type key-identifier;
    mandatory true;
    description "The key identifier referenced in a manifest.";
  }
  leaf algorithm {
    type identityref {
      base crypto-signature;
    }
    mandatory true;
    description "The signature algorithm for use with this key.";
  }
  leaf signature_bits {
    type uint16;
    mandatory true;
    description "The length of the signature provided in manifests
    signed with this key.";
  }
  leaf value {
    type string;
    mandatory true;
    description "The base64-encoded value of the public key.";
  }
}

list data_stream {
  key id;
  unique "source destination port";
  description "Stream of data packets to be authenticated";
  leaf id {
    type uint16;
    mandatory true;
    description "The datastream_id referenced by a
    manifest_stream.";
  }
  leaf source {
    type inet:ip-address;
    mandatory true;
description
  "The source IP address of the authenticated data stream.";
}
leaf destination {
  type rt-types:ip-multicast-group-address;
  mandatory true;
  description
    "The destination group IP address of the authenticated data stream.";
}
leaf port {
  type uint16;
  mandatory true;
  description
    "The destination UDP port of the authenticated data stream.";
}
leaf max_bitrate {
  type bitrate;
  mandatory true;
  description
    "The maximum bitrate expected for this data stream.";
}
leaf max_packetrate {
  type packetrate;
  mandatory true;
  description
    "The maximum packetrate expected for this data stream.";
}
list authenticator {
  key manifest_id;
  description
    "A manifest stream that authenticates this data";
  leaf manifest_id {
    type leafref {
      path "/anchor/manifest_stream/id";
    }
    mandatory true;
    description
      "The ID of a manifest stream that provides authentication for this data stream.";
  }
}
}
list manifest_stream {
    key id;
    description "Stream of manifests";
    leaf id {
        type uint16;
        mandatory true;
        description "The Manifest ID referenced in a manifest.";
    }
    leaf transport {
        type manifest-transport;
        mandatory true;
        description "The ID of the data stream that carries this manifest stream or a uri that provides a websocket with the stream of manifests.";
    }
    leaf hash_algorithm {
        type identityref {
            base crypto-hash;
        }
        mandatory true;
        description "The hash algorithm for the packet hashes within manifests in this stream.";
    }
    leaf hash_bits {
        type uint16;
        mandatory true;
        description "The number of bits of hash provided for packet hashes.";
    }
    leaf sequence_type {
        type identityref {
            base sequence-type;
        }
        mandatory true;
        description "The linkage to the data packet sequence numbers in the manifest.";
    }
}

Figure 1: Anchor Message YANG model
2.2.4. Example Anchor Message

```
{
  "ietf-ambi-anchor:anchor": {
    "self": {
      "uri": "https://example.com/ambi/anchor/example_1.json",
      "version": 1,
      "hash_algorithm": "blake2b",
      "hash_bits": 256,
      "expires": "2018-03-05T23:59:59Z"
    },
    "public_key": [
      {
        "id": 1,
        "algorithm": "ed25519",
        "signature_bits": 256,
        "value": "VGhpcyBpcyBub3QgYSBnb29kIGtleSB0byB1c2UuLi4NCg=="
      }
    ],
    "data_stream": [
      {
        "id": 10,
        "source": "192.0.2.10",
        "destination": "232.10.10.1",
        "port": 18001,
        "max_bitrate": "10Mbps",
        "max_packetrate": "1Kpps",
        "authenticator": [
          {
            "manifest_id": 1
          }
        ]
      },
      {
        "id": 20,
        "source": "192.0.2.10",
        "destination": "232.10.10.1",
        "port": 18002,
        "max_bitrate": "400Kbps",
        "max_packetrate": "40ppps",
        "authenticator": [
          {
            "manifest_id": 2
          }
        ]
      }
    ],
    "manifest_stream": [
```
2.3. Manifests

2.3.1. Overview

A manifest cannot be interpreted except in context of a known anchor message.

In order for a manifest to be considered as potentially authenticating a set of packets, the Anchor Version MUST match the value in a known unexpired anchor message, and the Anchor Hash MUST match the hash of the contents of that anchor message, according to the /anchor/self/hash_algorithm and /anchor/self/hash_bits fields, in order for a manifest to be accepted for use as evidence of authenticity and integrity.

A manifest also MUST NOT be accepted unless it has verified authenticity and integrity, either because it contains a cryptographic signature, or because it appeared in a secured unicast stream, or because another verified manifest has provided the packet hash for a packet containing this manifest.

2.3.2. Manifest Layout
2.3.2.1. Ver (Protocol Version)

MUST be set to 0 by senders, and if a nonzero value is received this message MUST NOT be accepted or processed as a manifest.

For manifests streams which were authenticated by a means other than cryptographic signature, it is RECOMMENDED that authenticators stop following this manifest stream and refresh the anchor if they receive an invalid version.

For manifest streams authenticated by signature, it is RECOMMENDED that authenticators remain joined to this stream and ignore this packet, as the manifest MAY have been sent maliciously.

An authenticator MAY implement a rate limit on invalid manifests and drop the stream if the rate is exceeded.

2.3.2.2. Reserved

MUST be set to 0 by senders and MUST be ignored by receivers.

2.3.2.3. P ("Purge" bit)

If this bit is 1, the anchor message this manifest specifies MUST be purged by authenticators who accept this manifest, so that it cannot be used to authenticate future manifests unless it was re-fetched.
2.3.2.4. S ("Step" bit)

If this bit is 1, the "Packet Identifier Step" field is present in this manifest, else it is not.

2.3.2.5. Anchor Version

The value from the "/anchor/self/version" field in the anchor message. If no unexpired anchor message with this version is known to the authenticator, this manifest MUST NOT be accepted.

2.3.2.6. Anchor Hash

The hash of the anchor message, using the algorithm indicated by the "/anchor/self/hash_algorithm" field, using the first bits from the hash up to the number of bits indicated by the "/anchor/self/hash_bits" field in the anchor message.

If the hash of an anchor message with this version does not match the bits in this field, this manifest MUST NOT be accepted.

This field is padded at the end with 0-bits until the end is 4-octet aligned.

2.3.2.7. Manifest ID

The value from "/anchor/manifest_stream/id" in the anchor message corresponding to the manifest stream this manifest is a part of.

2.3.2.8. First Packet Hash Identifier

The packet number corresponding to the first packet hash that’s contained in this manifest. This refers to a value in a data packet described by the "/anchor/manifest_stream/sequence_type" field for this manifest stream.

2.3.2.9. Packet Identifier Step

If the "S bit" is 0, this field is not present in the manifest.

If the "S bit" is 1, this field is repeatedly added to the First Packet Hash Identifier using 32-bit signed arithmetic to determine the packet number of subsequent hashes.
2.3.2.10. Packet Hash Count

The number of hashes contained in the Packet Hashes section.

2.3.2.11. Packet Hashes

Concatenated Hashes of the data packets authenticated by this manifest. The hash covers the IP payload of the packet, it is calculated with the algorithm indicated by the "/anchor/manifest_stream" object from the anchor message, with an "id" field matching the "Manifest ID" field, with the algorithm and number of bits equal to the "hash_algorithm" and "hash_bits" field from that object. The hashes are concatenated without padding, except the last octet is padded with 0 if necessary.

3. IANA Considerations


TBD2: Add a new entry to the "UDP Option Kind" numbers registry: https://tools.ietf.org/html/draft-ietf-tsvwg-udp-options-02#section-14

TBD: check guidelines in https://tools.ietf.org/html/rfc5226 and remove this paragraph

Example from: https://tools.ietf.org/html/rfc5226#section-5.1

TBD: new Resource Record type with a URI in the RRData? Or should this be done as a NAPTR + URI chain?

[TO BE REMOVED: Please add the yang model in Section 2.2.3 to: https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml


4. Security Considerations

4.1. Packet Identifiers

TBD: explain attack from generating malicious packets and then looking for collisions, as opposed to having to generate a collision including a sequence number and then hitting a match.
TBD: DNSSEC vis-a-vis anchor url discovery. (we need a diagram about for middle-box handling of a revers-path propagated join?) Explain why malicious DNS could deny service, but cannot cause accepting attack packets.

TBD: Is the purge bit sufficient to cover when a key is found to be leaked?

TBD: follow the rest of the guidelines: https://tools.ietf.org/html/rfc3552

5. References

5.1. Normative References


5.2. Informative References


Authors' Addresses

Jake Holland
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02144
United States of America

Email: jakeholland.net@gmail.com

Kyle Rose
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02144
United States of America

Email: krose@krose.org