Network Working Group
Internet-Draft
Obsoletes: 7298 (if approved)
Updates: 6126bis (if approved)
Intended status: Standards Track
Expires: May 18, 2019

C. Do
W. Kolodziejak
J. Chroboczek
IRIF, University of Paris-Diderot

November 14, 2018

HMAC authentication for the Babel routing protocol
draft-ietf-babel-hmac-01

Abstract

This document describes a cryptographic authentication for the Babel routing protocol that has provisions for replay avoidance. This document updates RFC 6126bis and obsoletes RFC 7298.

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 May 18, 2019.

Copyright Notice

Table of Contents

1. Introduction

By default, the Babel routing protocol trusts the information contained in every UDP packet it receives on the Babel port. An attacker can redirect traffic to itself or to a different node in the network, causing a variety of potential issues. In particular, an attacker might:

- spoof a Babel packet, and redirect traffic by announcing a smaller metric, a larger seqno, or a longer prefix;

- spoof a malformed packet, which could cause an insufficiently robust implementation to crash or interfere with the rest of the

network;

- replay a previously captured Babel packet, which could cause traffic to be redirected or otherwise interfere with the network.

Protecting a Babel network is challenging due to the fact that the Babel protocol uses both unicast and multicast communication. One possible approach, used notably by the Babel over Datagram Transport Layer Security (DTLS) protocol [I-D.ietf-babel-dtls], is to use unicast communication for all semantically significant communication, and then use a standard unicast security protocol to protect the Babel traffic. In this document, we take the opposite approach: we define a cryptographic extension to the Babel protocol that is able to protect both unicast and multicast traffic, and thus requires very few changes to the core protocol.

1.1. Applicability

The protocol defined in this document assumes that all interfaces on a given link are equally trusted and share a small set of symmetric keys (usually just one, and two during key rotation). The protocol is inapplicable in situations where asymmetric keying is required, where the trust relationship is partial, or where large numbers of trusted keys are provisioned on a single link at the same time.

This protocol supports incremental deployment (where an insecure Babel network is made secure with no service interruption), and it supports graceful key rotation (where the set of keys is changed with no service interruption).

This protocol does not require synchronised clocks, it does not require persistently monotonic clocks, and it does not require any form of persistent storage.

1.2. Assumptions and security properties

The correctness of the protocol relies on the following assumptions:

- that the Hashed Message Authentication Code (HMAC) being used is invulnerable to pre-image attacks, i.e., that an attacker is unable to generate a packet with a correct HMAC;

- that a node never generates the same index or nonce twice over the lifetime of a key.

The first assumption is a property of the HMAC being used. The second assumption can be met either by using a robust random number generator and sufficiently large indices and nonces, by using a reliable hardware clock, or by rekeying whenever a collision becomes likely.

If the assumptions above are met, the protocol described in this document has the following properties:

- it is invulnerable to spoofing: any packet accepted as authentic is the exact copy of a packet originally sent by an authorised node;

- locally to a single node, it is invulnerable to replay: if a node has previously accepted a given packet, then it will never again accept a copy of this packet or an earlier packet from the same sender;

- among different nodes, it is only vulnerable to immediate replay: if a node A has accepted a packet from C as valid, then a node B will only accept a copy of that packet as authentic if B has accepted an older packet from C and B has received no later packet from C.

While this protocol makes serious efforts to mitigate the effects of a denial of service attack, it does not fully protect against such attacks.

1.3. Specification of Requirements

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. Conceptual overview of the protocol

When a node B sends out a Babel packet through an interface that is configured for cryptographic protection, it computes one or more HMACs which it appends to the packet. When a node A receives a packet over an interface that requires cryptographic protection, it independently computes a set of HMACs and compares them to the HMACs appended to the packet; if there is no match, the packet is discarded.

In order to protect against replay B maintains a per-interface 32-bit integer known as the "packet counter" (PC). Whenever B sends a packet through the interface, it embeds the current value of the PC within the region of the packet that is protected by the HMACs and increases the PC by at least one. When A receives the packet, it compares the value of the PC with the one contained in the previous packet received from B, and unless it is strictly greater, the packet is discarded.

By itself, the PC mechanism is not sufficient to protect against replay. Consider a peer A that has no information about a pair B

(e.g., because it has recently rebooted). Suppose that A receives a packet ostensibly from B carrying a given PC; since A has no information about B, it has no way to determine whether the packet is freshly generated or a replay of a previously sent packet.

In this situation, A discards the packet and challenges B to prove that it knows the HMAC key. It sends a "challenge request", a TLV containing a unique nonce, a value that has never been used before and will never be used again. B replies to the challenge request with a "challenge reply", a TLV containing a copy of the nonce chosen by A, in a packet protected by HMAC and containing the new value of B's PC. Since the nonce has never been used before, B's reply proves B's knowledge of the HMAC key and the freshness of the PC.

By itself, this mechanism is safe against replay if B never resets its PC. In practice, however, this is difficult to ensure, as persistent storage is prone to failure, and hardware clocks, even when available, are occasionally reset. Suppose that B resets its PC to an earlier value, and sends a packet with a previously used PC n. A challenges B, B successfully responds to the challenge, and A accepts the PC equal to n + 1. At this point, an attacker C may send a replayed packet with PC equal to n + 2, which will be accepted by

A.

Another mechanism is needed to protect against this attack. In this protocol, every PC is tagged with an "index", an arbitrary string of octets. Whenever B resets its PC, or whenever B doesn't know whether its PC has been reset, it picks an index that it has never used before (either by drawing it randomly or by using a reliable hardware clock) and starts sending PCs with that index. Whenever A detects that B has changed its index, it challenges B again.

With this additional mechanism, this protocol is invulnerable to replay attacks (see Section 1.2 above).

3. Data Structures

3.1. The Interface Table

Every Babel node maintains an interface table, as described in [RFC6126bis] Section 3.2.3. This protocol extends the entries in this table with a set of HMAC keys, and a pair (Index, PC), where Index is an arbitrary string of bytes and PC is a 32-bit integer. The Index is initialised to a value that has never been used before (e.g., by choosing a random string of sufficient length).

3.2. The Neighbour table

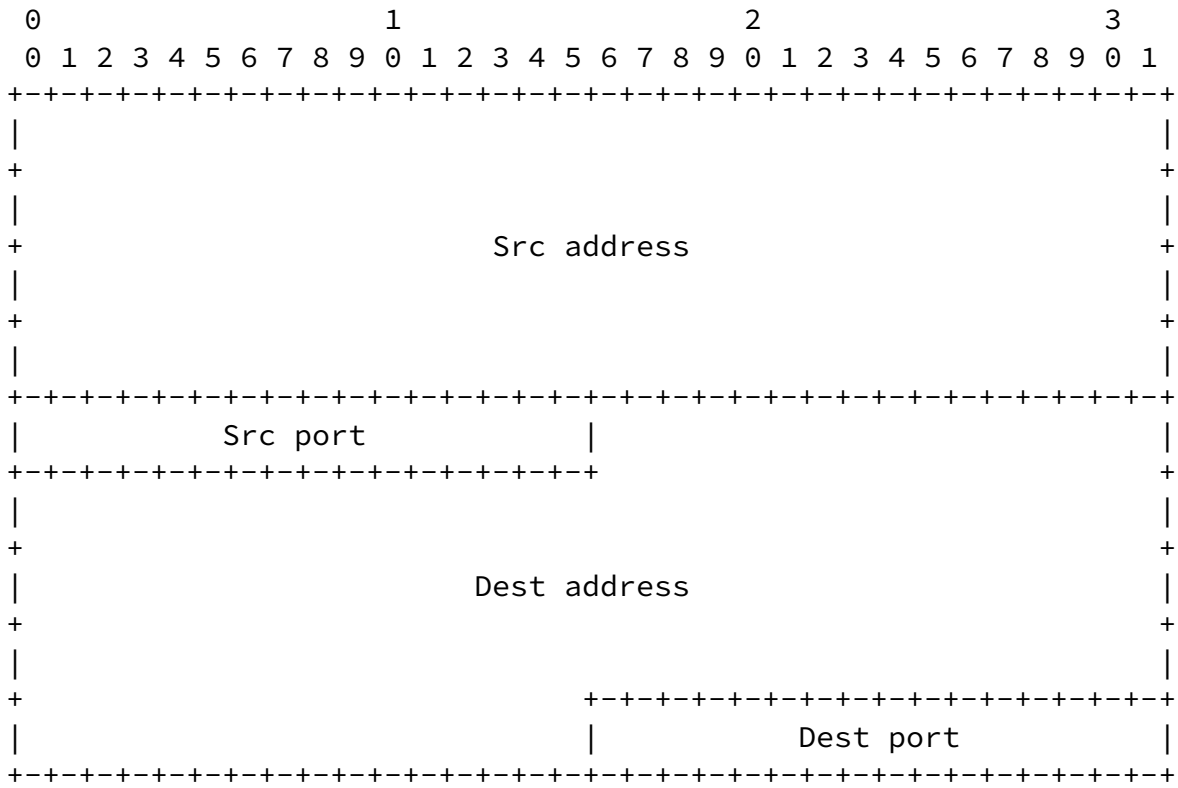
Every Babel node maintains a neighbour table, as described in [RFC6126bis] Section 3.2.4. This protocol extends the entries in this table with a pair (Index, PC), as well as a nonce (an arbitrary string of bytes) and a challenge expiry timer. The Index and PC are initially undefined, and are managed as described in Section 4.3. The Nonce and expiry timer are initially undefined and used as described in Section 4.3.1.1.

4. Protocol Operation

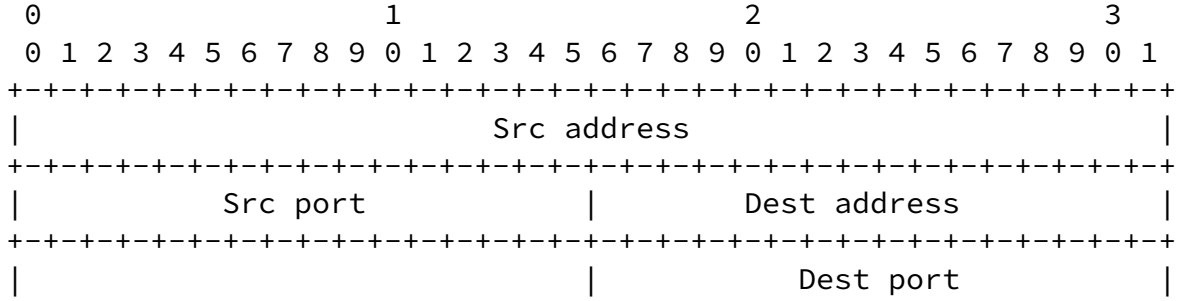
4.1. HMAC computation

A Babel node computes an HMAC as follows.

First, the node builds a pseudo-header that will participate in HMAC computation but will not be sent. If the packet was carried over IPv6, the pseudo-header has the following format:



If the packet was carried over IPv4, the pseudo-header has the following format:



Fields :

Src address The source IP address of the packet.

Src port The source UDP port number of the packet.

Dest address The destination IP address of the packet.

Src port The destination UDP port number of the packet.

The node takes the concatenation of the pseudo-header and the packet including the packet header but excluding the packet trailer (from octet 0 inclusive up to Body Length + 4 exclusive) and computes an HMAC as defined in Section 2 of [RFC2104] with one of the implemented hash algorithms. Every implementation MUST implement HMAC-SHA256 [RFC6234], and MAY implement other HMAC algorithms.

4.2. Packet Transmission

A Babel node may delay actually sending TLVs by a small amount, in order to aggregate multiple TLVs in a single packet up to the interface MTU (Section 4 of [RFC6126bis]). For an interface on which HMAC protection is configured, the TLV aggregation logic MUST take into account the overhead due to PC TLVs (one in each packet) and HMAC TLVs (one per configured key).

Before sending a packet, the following actions are performed:

- a PC TLV containing the PC and Index associated with the outgoing interface is appended to the packet body; the PC is incremented by a strictly positive amount (typically just 1); if the PC overflows, a new index is generated;

- for each key configured on the interface, an HMAC is computed as specified in Section 4.1 above, and an HMAC TLV is appended to the packet trailer (see Section 4.2 of [RFC6126bis]).

4.3. Packet Reception

When a packet is received on an interface that is configured for HMAC protection, the following steps are performed before the packet is passed to normal processing:

- First, the receiver checks whether the trailer of the received packet carries at least one HMAC TLV; if not, the packet is immediately dropped and processing stops. Then, for each key configured on the receiving interface, the implementation computes the HMAC of the packet. It then compares every generated HMAC against every HMAC included in the packet; if there is at least one match, the packet passes the HMAC test; if there is none, the packet is silently dropped and processing stops at this point. In order to avoid memory exhaustion attacks, an entry in the Neighbour Table MUST NOT be created before the HMAC test has passed successfully. The HMAC of the packet MUST NOT be computed for each HMAC TLV contained in the packet, but only once for each configured key.

- The packet body is then parsed a first time. During this "preparse" phase, the packet body is traversed and all TLVs are ignored except PC TLVs, Challenge Requests and Challenge Replies. When a PC TLV is encountered, the enclosed PC and Index are saved for later processing; if multiple PCs are found, only the first one is processed, the remaining ones are silently ignored. If a Challenge Request is encountered, a Challenge Reply is scheduled, as described in Section 4.3.1.2, and if a Challenge Reply is encountered, it is tested for validity as described in Section 4.3.1.3 and a note is made of the result of the test.

- The preparse phase above has yielded two pieces of data: the PC and Index from the first PC TLV, and a bit indicating whether the packet contains a successful Challenge Reply. If the packet does not contain a PC TLV, the packet is dropped and processing stops at this point. If the packet contains a successful Challenge Reply, then the PC and Index contained in the PC TLV are stored in the Neighbour Table entry corresponding to the sender (which may need to be created at this stage).

- If there is no entry in the Neighbour Table corresponding to the sender, or if such an entry exists but contains no Index, or if the Index it contains is different from the Index contained in the PC TLV, then a challenge is sent as described in Section 4.3.1.1, processing stops at this stage, and the packet is dropped.

- At this stage, the Index contained in the PC TLV is equal to the Index in the Neighbour Table entry corresponding to the sender.

The receiver compares the received PC with the PC contained in the Neighbour Table; if the received PC smaller or equal than the PC contained in the Neighbour Table, the packet is silently dropped and processing stops (no challenge is sent in this case, since the mismatch might be caused by harmless packet reordering on the link). Otherwise, the PC contained in the Neighbour Table entry is set to the received PC, and the packet is accepted.

After the packet has been accepted, it is processed as normal, except that any PC, Challenge Request and Challenge Reply TLVs that it contains are silently ignored.

4.3.1. Challenge Requests and Replies

During the preparse stage, the receiver might encounter a mismatched Index, to which it will react by scheduling a Challenge Request. It might encounter a Challenge Request TLV, to which it will reply with a Challenge Reply TLV. Finally, it might encounter a Challenge Reply TLV, which it will attempt to match with a previously sent Challenge Request TLV in order to update the Neighbour Table entry corresponding to the sender of the packet.

4.3.1.1. Sending challenges

When it encounters a mismatched Index during the preparse phase, a node picks a nonce that it has never used before, for example by drawing a sufficiently large random string of bytes or by consulting a strictly monotonic hardware clock. It stores the nonce in the entry of the Neighbour Table of the neighbour (the entry might need to be created at this stage), initialises the neighbour's challenge expiry timer to 30 seconds, and sends a Challenge Request TLV to the unicast address corresponding to the neighbour.

A node MAY aggregate a Challenge Request with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the sender of the neighbour, it MAY send the buffered TLVs in the same packet as the Challenge Request. However, it MUST arrange for the Challenge Request to be sent in a timely manner, as any packets received from that neighbour will be silently ignored until the challenge completes.

Since a challenge may be prompted by a replayed packet, a node MUST impose a rate limitation to the challenges it sends; a limit of one challenge every 300ms for each neighbour is suggested.

4.3.1.2. Replying to challenges

When it encounters a Challenge Request during the preparse phase, a node constructs a Challenge Reply TLV by copying the Nonce from the Challenge Request into the Challenge Reply. It sends the Challenge Reply to the unicast address of the sender of the Challenge Reply.

A node MAY aggregate a Challenge Reply with other TLVs; in other words, if it has already buffered TLVs to be sent to the unicast address of the sender of the Challenge Request, it MAY send the buffered TLVs in the same packet as the Challenge Reply. However, it MUST arrange for the Challenge Reply to be sent in a timely manner (within a few seconds), and SHOULD NOT send any other packets over the same interface before sending the Challenge Reply, as those would be dropped by the challenger.

A challenge sent to a multicast address MUST be silently ignored.

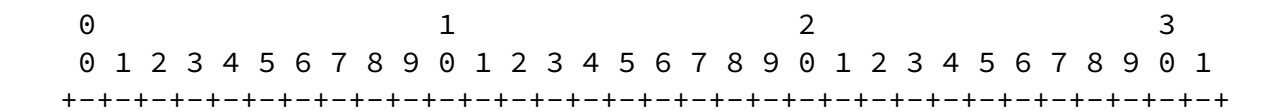
4.3.1.3. Receiving challenge replies

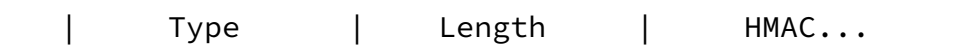
When it encounters a Challenge Reply during the preparse phase, a node consults the Neighbour Table entry corresponding to the neighbour that sent the Challenge Reply. If no challenge is in progress, i.e., if there is no Nonce stored in the Neighbour Table entry or the Challenge timer has expired, the Challenge Reply is silently ignored and the challenge has failed.

Otherwise, the node compares the Nonce contained in the Challenge Reply with the Nonce contained in the Neighbour Table entry. If the two are equal (they have the same length and content), then the challenge has succeeded; otherwise, the challenge has failed.

5. Packet Format

5.1. HMAC TLV





Fields :

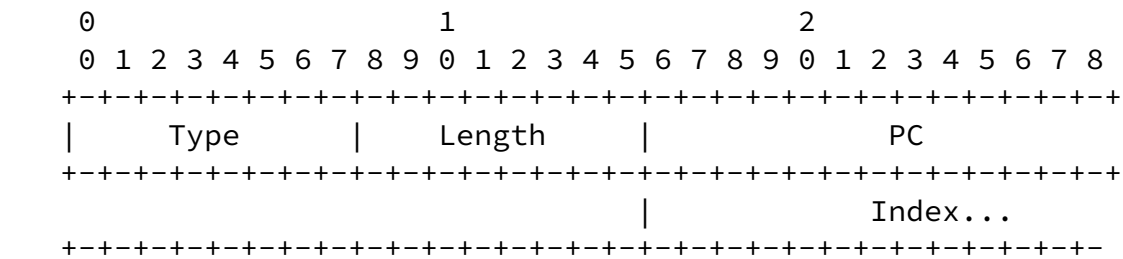
Type Set to TBD to indicate an HMAC TLV.

Length The length of the body, exclusive of the Type and Length fields. The length of the body depends on the hash function used.

HMAC The body contains the HMAC of the whole packet plus the pseudo header.

This TLV is allowed in the packet trailer (see Section 4.2 of [RFC6126bis]), and MUST be ignored if it is found in the packet body.

5.2. PC TLV



Fields :

Type Set to TBD to indicate a PC TLV.

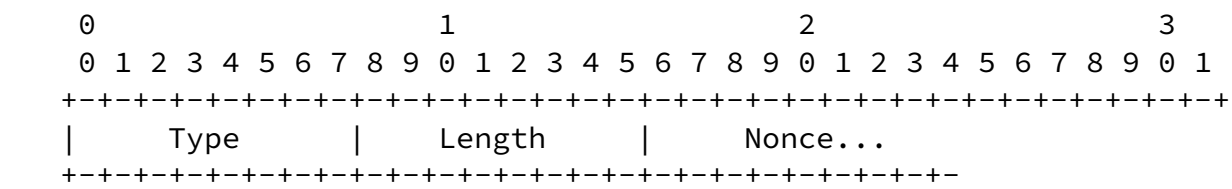
Length The length of the body, exclusive of the Type and Length fields.

PC The Packet Counter (PC), which is increased with every packet sent over this interface. A new index MUST be generated whenever the PC overflows.

Index The sender's Index.

Indices are limited to a size of 32 octets: a node MUST NOT send a TLV with an index of size strictly larger than 32 octets, and a node MAY silently ignore a PC TLV with an index of size strictly larger than 32 octets.

5.3. Challenge Request TLV



Fields :

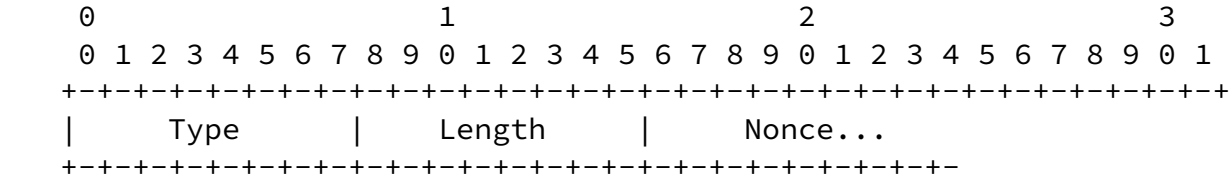
Type Set to TBD to indicate a Challenge Request TLV.

Length The length of the body, exclusive of the Type and Length fields.

Nonce The nonce uniquely identifying the challenge.

Nonces are limited to a size of 192 octets: a node MUST NOT send a Challenge Request TLV with a nonce of size strictly larger than 192 octets, and a node MAY ignore a nonce that is of size strictly larger than 192 octets.

5.4. Challenge Reply TLV



Fields :

Type Set to TBD to indicate a Challenge Reply TLV.

Length The length of the body, exclusive of the Type and Length

fields. The length of the body is set to the same size as the challenge request TLV length received.

Nonce A copy of the nonce contained in the corresponding challenge request.

6. Security Considerations

This document defines a mechanism that provides basic security properties for the Babel routing protocol. The scope of this protocol is strictly limited: it only provides authentication, on the assumption that routing information is not confidential, only symmetric keying, and only allows for the use of a small number of symmetric keys on each link. Deployments that need more features, e.g., confidentiality or asymmetric keying, should use a more featureful security mechanism such as the one described in [I-D.ietf-babel-dtls].

This mechanism relies on two assumptions, as described in Section 1.2. First, it assumes that the hash being used is invulnerable to pre-image attacks (Section 1.1 of [RFC6039]); at the time of writing, SHA-256, which is mandatory to implement (Section 4.1), is believed to be safe against practical attacks.

Second, it assumes that indices and nonces are generated uniquely over the lifetime of a key used for HMAC computation. This property can be satisfied either by using a cryptographically secure random number generator to generate indices and nonces that contain enough entropy (64-bit values are believed to be large enough for all practical applications), or by using a reliably monotonic hardware clock. If unicity cannot be guaranteed (e.g., because a hardware clock has been reset), then rekeying is necessary.

While it is probably not possible to be immune against denial of service (DoS) attacks in general, this protocol includes a number of mechanisms designed to mitigate such attacks. In particular, reception of a packet with no correct HMAC creates no local state whatsoever (Section 4.3). Reception of a replayed packet with correct hash, on the other hand, causes a challenge to be sent; this is mitigated somewhat by requiring that challenges be rate-limited.

At first sight, sending a challenge requires retaining enough

information to validate the challenge reply. However, the nonce included in a challenge request and echoed in the challenge reply can be fairly large (up to 192 octets), which should in principle permit encoding the per-challenge state as a secure "cookie" within the nonce itself.

7. IANA Considerations

IANA is instructed to allocate the following values in the Babel TLV Numbers registry:

Type	Name	Reference
TBD	HMAC	this document
TBD	PC	this document
TBD	Challenge Request	this document
TBD	Challenge Reply	this document

8. Acknowledgments

The protocol described in this document is based on the original HMAC protocol defined by Denis Ovsienko [RFC7298]. The use of a pseudo-header was suggested by David Schinazi. The use of an index to avoid replay was suggested by Markus Stenberg. The authors are also indebted to Florian Horn and Toke Hoyland-Jorgensen.

9. References

9.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

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

DOI 10.17487/RFC2119, March 1997.

[RFC6039] Manral, V., Bhatia, M., Jaeggli, J., and R. White, "Issues with Existing Cryptographic Protection Methods for Routing Protocols", RFC 6039, DOI 10.17487/RFC6039, October 2010, <https://www.rfc-editor.org/info/rfc6039>.

[RFC6126bis] Chroboczek, J. and D. Schinazi, "The Babel Routing Protocol", draft-ietf-babel-rfc6126bis-06 (work in progress), October 2018.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

9.2. Informational References

[I-D.ietf-babel-dtls] Decimo, A., Schinazi, D., and J. Chroboczek, "Babel Routing Protocol over Datagram Transport Layer Security", draft-ietf-babel-dtls-01 (work in progress), October 2018.

[RFC7298] Ovsienko, D., "Babel Hashed Message Authentication Code (HMAC) Cryptographic Authentication", RFC 7298, DOI 10.17487/RFC7298, July 2014, <https://www.rfc-editor.org/info/rfc7298>.

Appendix A. Incremental deployment and key rotation

This protocol supports incremental deployment (transitioning from an insecure network to a secured network with no service interruption)

and key rotation (transitioning from a set of keys to a different set of keys).

In order to perform incremental deployment, the nodes in the network are first configured in a mode where packets are sent with authentication but not checked on reception. Once all the nodes in the network are configured to send authenticated packets, nodes are reconfigured to reject unauthenticated packets.

In order to perform key rotation, the new key is added to all the nodes; once this is done, both the old and the new key are sent in all packets, and packets are accepted if they are properly signed by either of the keys. At that point, the old key is removed.

In order to support incremental deployment and key rotation, implementations SHOULD support an interface configuration in which they send authenticated packets but accept all packets, and SHOULD allow changing the set of keys associated with an interface without a restart.

Appendix B. Changes from previous versions

B.1. Changes since draft-ietf-babel-hmac-00

- Changed the title.
- Removed the appendix about the packet trailer, this is now in rfc6126bis.
- Removed the appendix with implicit indices.
- Clarified the definitions of acronyms.
- Limited the size of nonces and indices.

Authors' Addresses

Clara Do
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France

Email: clarado_perso@yahoo.fr

Weronika Kolodziejak
IRIF, University of Paris-Diderot
75205 Paris Cedex 13
France

Email: weronika.kolodziejak@gmail.com

Juliusz Chroboczek
IRIF, University of Paris-Diderot
Case 7014
75205 Paris Cedex 13
France

Email: jch@irif.fr