

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: January 11, 2021

E. Birrane
S. Heiner
JHU/APL
July 10, 2020

Security Context Template
draft-birrane-dtn-scot-00

Abstract

This document defines a standard template for security contexts written for the Bundle Protocol Security Protocol (BPSec).

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 January 11, 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1. Introduction	2
1.1. Specification Scope	3
1.2. Related Documents	3
1.3. Terminology	4
2. System Overview	4
2.1. Methods of Establishing Context	4
2.1.1. Out-Of-Band Mode	4
2.1.2. In-Band Mode	5
2.1.3. Hybrid Mode	6
2.2. Security Policy Roles	7
2.3. Common Events and Actions	8
2.3.1. Bundle Protocol Reason Codes	8
2.3.2. Event Codes	10
2.3.3. Processing Actions	12
2.4. Security Policy Considerations	14
2.5. Security Context Template	15
2.5.1. Overview	16
2.5.2. Interfaces	16
2.5.3. Definitions	17
2.5.4. Canonicalization Algorithms	17
2.5.5. Processing	18
2.5.6. Policy	18
2.5.7. IANA Considerations	18
3. Normative References	18
Authors' Addresses	18

1. Introduction

The Bundle Protocol (BP) may operate in environments that prevent reliable synchronization of session information, to include negotiated cryptographic material. To accommodate for this possibility, BP bundles may use extension blocks to carry annotative information. The Bundle Protocol Security Protocol (BPsec) defines security-related extension blocks (security blocks) to carry cryptographic material, to optionally include material that might otherwise be expected resident at communication endpoints. To accomplish these, BPsec security blocks specify a "security context" comprising the material generated for, carried with, and/or otherwise utilized by the block.

Where BPsec security blocks traverse networks for which a synchronized security mechanism exists, a security context can be defined which minimally identifies endpoint information. For example, in networks that support TLS, a security context can be defined to carry TLS record information after a successful TLS handshake has been used to synchronize state at the endpoints of the

exchange. Alternatively, where no such synchronizing security mechanisms exist, a security context can be defined which carries necessary configuration, cryptographic material, and policy.

The diversity of networks in which BP, and thus BPsec, may be deployed implies a diversity of network contexts in which security may be applied. For this reason, it is expected that multiple security contexts will be defined for BPsec security blocks, such that BPsec agents may select the most suitable context. This document defines a template for the documentation of security contexts. This template includes Bundle Protocol (BP) reason codes, discrete events in the life-cycle of a security block, and policy actions that may be taken by a bundle node when processing a security block.

1.1. Specification Scope

This document defines the information that must be addressed in the definition of any BPsec security context specification. Specifically, this document details the following information.

- o Data specification requirements.
- o Definitions and handling requirements for standard BP reason codes.
- o Definitions and handling requirements for standard error codes.
- o Guidance for specifying context-specific parameters and results.

The SCoT addresses only that information necessary for the proper specification, interpretation, and processing of BPsec security blocks. Any specific security protocol, cipher suite, or enumeration set other than those defined by BP and BPsec are not considered part of this document.

1.2. Related Documents

This document is best read and understood within the context of the following other DTN documents:

The Concise Binary Object Representation (CBOR) format [RFC7049] defines a data format that allows for small code size, fairly small message size, and extensibility without version negotiation. The block-specific-data associated with BPsec security blocks are encoded in this data format.

The Bundle Protocol [I-D.ietf-dtn-bpbis] defines the format and processing of bundles, defines the extension block format used to represent BPsec security blocks, and defines the canonical block structure used by this specification.

The Bundle Security Protocol [I-D.ietf-dtn-bpsec] defines the BP extension blocks used to implement security services in a DTN. This also outlines the need for security contexts customized to the networks in which BP and BPsec are deployed.

1.3. Terminology

This section defines terminology unique to the Security Context Template. Definitions of other terms specific to BP and BPsec, such as "Security Acceptor", "Security Block", "Security Context", "Security Operation", "Security Service", "Security Source", and "Security Target" are as defined in BPsec [I-D.ietf-dtn-bpsec].

2. System Overview

This section describes how a security context is used by BPsec to normalize the diversity of environments encountered by the Bundle Protocol.

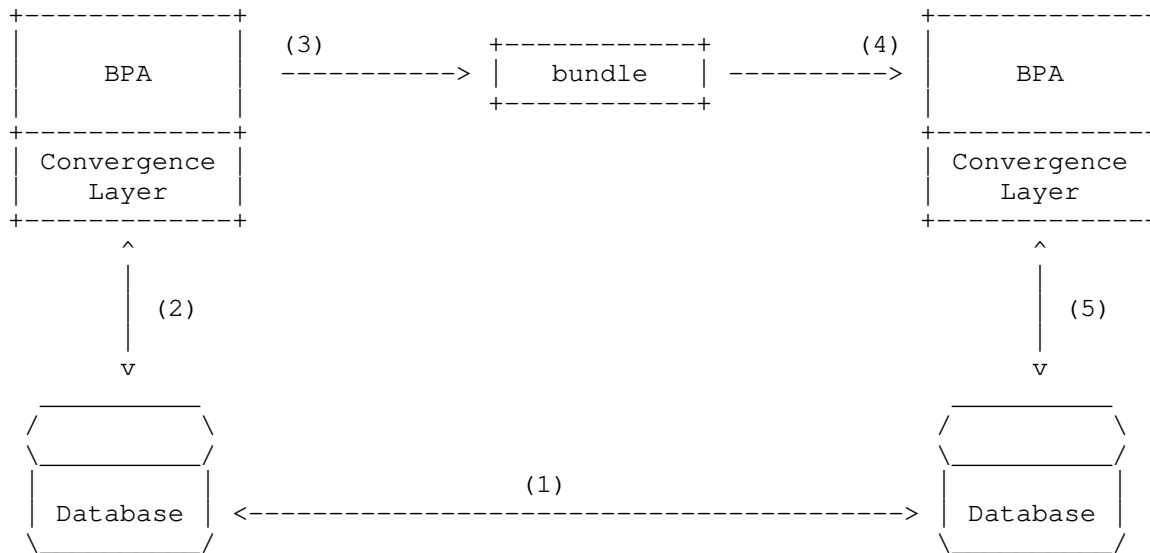
2.1. Methods of Establishing Context

The definition of a security context is based in the concept that different networks would provide annotative security information in different ways as a function of the capabilities of the network itself. This section discusses three general methods of generating context information: out-of-band, in-band, and a hybrid approach.

For each of these methods the term "in-band" refers to information carried within a bundle.

2.1.1. Out-Of-Band Mode

The Out-of-Band method of context establishment utilizes out-of-band information only, requiring session information to be pre-shared. The sending and receiving nodes must be configured using an out-of-band method such as separate key management, one-time padding, or ID-based encryption and use a pre-placed session key in order to establish the context.

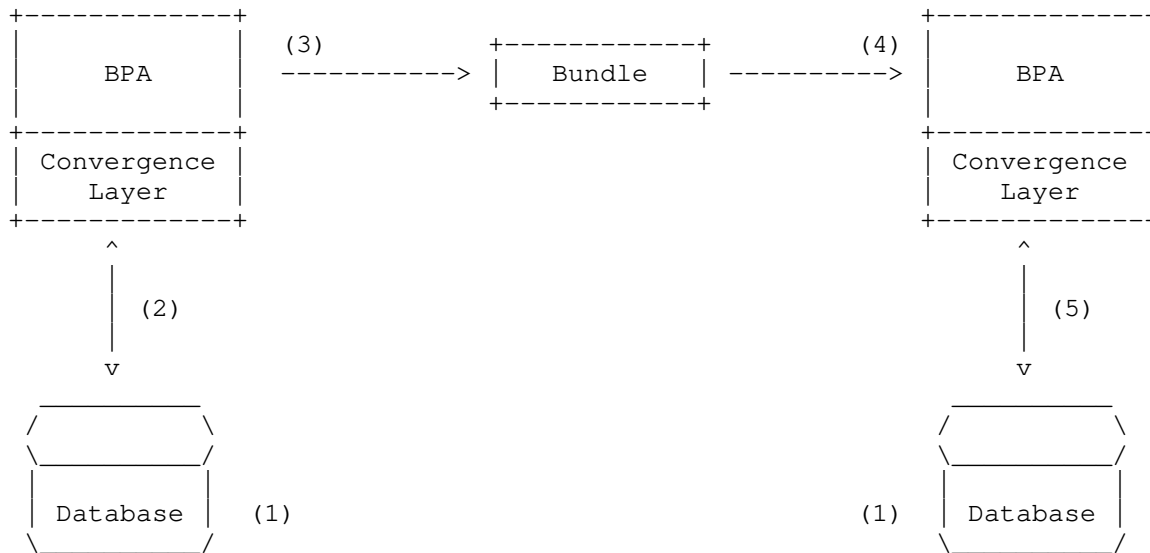


In-Band Mode

Step 1 shows the out-of-band configuration of both the sending and receiving nodes. A session key is pre-placed in the databases accessible by the nodes in step 1. Step 2 shows the sending node retrieving this session key which is used in step 3 to add a security block to the bundle which may transport a shared key between the sending and receiving nodes. The bundle arrives at the receiving node in step 4, and the pre-placed session key is retrieved in step 5. The session key can be used along with the shared key to establish the context between the two nodes.

2.1.2. In-Band Mode

The In-Band method of context establishment utilizes in-band information only. The key encryption key for both the sending and receiving node must be pre-placed so that the node can fetch the information from its in-band database. The session key and any additional information necessary to establish the context is transported by the bundle. The receiving node must use its pre-placed key-encryption-key in order to recover the session key when the bundle is received.

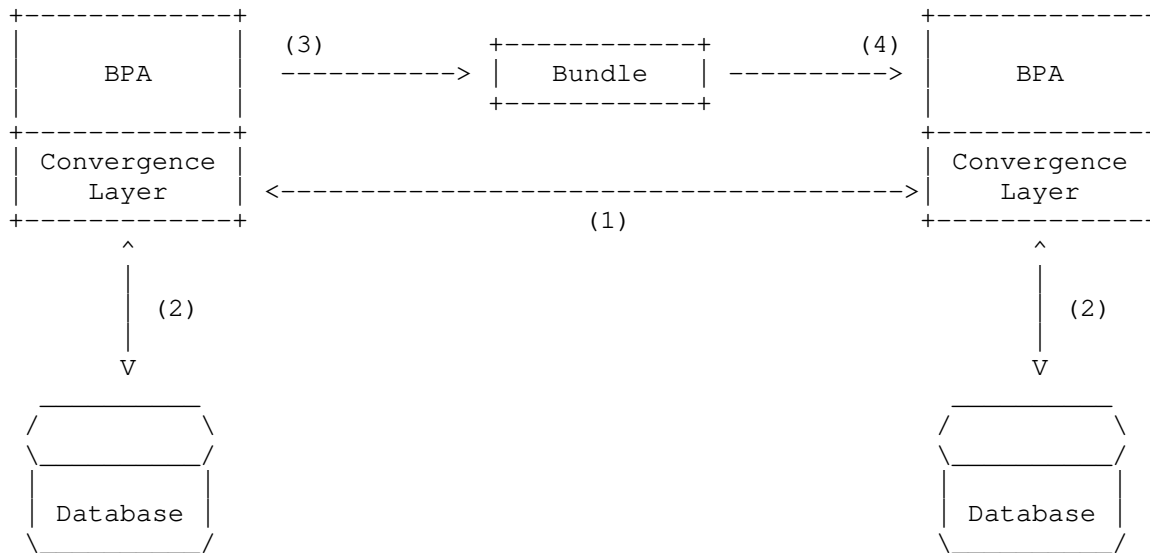


In-Band Mode

Step 1 indicates that the key encryption key has been pre-placed at both the sending and receiving node. The key encryption key is supplied to the sending node in step 2, which then uses this key in step 3 to add a security block to the bundle. This security block contains a session key and other necessary security context parameters used to establish the context. The bundle is received in step 4, and the key encryption key pre-placed at the receiving node is fetched in step 5. The key encryption key is used to recover the session key from the bundle, and the session key can be used to retrieve the rest of the security results from the block.

2.1.3. Hybrid Mode

Using the Hybrid method of context establishment, both in-band and out-of-band information is utilized. A session key is negotiated out-of-band, while any additional information necessary to establish the context is transported by the bundle.



Hybrid Mode

Step 1 shows session establishment, performed by the convergence layer of two BP nodes using in-band information including a negotiated session key. At step 2, the session data is stored at both nodes in their respective databases. The sending node adds a security block to the bundle containing the information necessary to establish the context in the form of security context parameters in step 3. Step 4 shows the augmented bundle arriving at the receiving node. The receiving node can then use the session data in its database (in-band information) and the session information transmitted in the bundle as security context parameters (out-of-band information).

2.2. Security Policy Roles

A security context may be interpreted differently based on the role of the BPsec-aware node processing the security service. This section defines the roles associated with a security operation.

Security Source

A security source node must add the security service to the bundle as specified by some policy. The bundle will first be inspected to ensure that the newly required security block has not already been added to the bundle. If it is determined that the required security block is not already represented in the bundle, a new security block is added and the specified

security context is used to apply the security service to each security target.

Security Verifier

A security verifier node must verify (not process) a security service in the bundle as specified by the receiver security policy.

A security verifier will verify the integrity signature(s) stored as security results of the BIB if the security service described by the receiver security policy rule is integrity.

If policy identifies a BCB for which the node is a security verifier, authenticity of the data belonging to the BCB's security targets is verified. Some confidentiality security contexts may not support this action, as the cipher suite(s) that they utilize do not expose an authentication function.

Security Acceptor

A security acceptor node must process a security service in the bundle as specified by policy. In order to process a security service, each security target belonging to the security block must have its integrity verified and/or its contents decrypted.

2.3. Common Events and Actions

This section describes the identification of common events and actions associated with the processing of BPsec blocks at bundle nodes. Since these items are not specific to a single security context, they should be considered standard across all defined security contexts for BPsec.

2.3.1. Bundle Protocol Reason Codes

This section describes a set of reason codes associated with the processing of a bundle based on a security analysis performed by a BPsec-aware BPA.

Bundle protocol agents (BPAs) must process blocks and bundles in accordance with both BP policy and BPsec policy. The decision to receive, forward, deliver, or delete a bundle may be communicated to the report-to address of the bundle, in the form of a status report, as a method of tracking the progress of the bundle through the network. The status report for a bundle may be augmented with a "reason code" explaining why the particular action was taken on the bundle.

Missing Security Service

This reason code indicates that a bundle was missing one or more required security services. This reason code is used when a bundle is received by a security verifier or security acceptor and is missing a security service required by that verifier or acceptor.

Unknown Security Service

This reason code indicates that a security service present in a bundle cannot be understood by the security verifier or security acceptor of that service. For example, if a security service references a security context identifier, cipher suite name, or other parameter that is not known by the verifier/acceptor then the service cannot be processed and this reason code may be sent. This reason should not be sent by a node that is not configured as a verifier or acceptor of the service. There is no requirement that all BPSec-aware nodes in a network be able to understand all security services in all bundles in the network.

Unexpected Security Service

This reason code indicates that a BPSec-aware node received a bundle which contained more security services than expected. This is typically used to indicate that a bundle contained a security service which was not required by the BPSec-aware node receiving the bundle. This reason code should not be seen as an error condition as it is expected that BPSec-aware nodes will see security services in a bundle for which they are neither a verifier nor an acceptor. In certain networks, this reason code may be useful in identifying misconfiguration in the network.

Failed Security Service

This reason code indicates that a BPSec-aware node was unable to process an existing, known security service in a bundle. This may occur when a security-source is unable to add a required service to a bundle. This may occur if a security service fails to verify at a security verifier. This may occur if a security service fails to be processed at a security acceptor.

Conflicting Security Services

This reason code indicates that a bundle received by a BPSec-aware node included a set of security services disallowed by the BPSec protocol and that security processing was unable to proceed because of a BPSec protocol violation.

2.3.2. Event Codes

A life-cycle of a security operation within BPSec is independent of the security context used to populate the contents of that security operation. However, security contexts must provide guidance on how a BPSec-aware node should react to these events for security operations using that context.

This section identifies the unique events in the life-cycle of a security operation that may identify processing points within a security context.

2.3.2.1. Security Source Events

At a security source, three events may be associated with the security operation as its life-cycle is initiated.

`source_for_sop`

When a node is designated as a security source for a security operation, there is a security policy rule that requires the security operation to be present in the bundle. When the sender security policy rule that applies to the bundle is identified, the security operation is acknowledged as needed.

`sop_added_at_source`

A security operation is added when it is represented in the bundle as a fully populated security block. In order for the security operation to be considered as added, the security block must be allocated for the bundle, represented in the bundle, and populated. Population of a security block includes information provided by the sender security policy rule and any security results associated with the security operation. These security results must be calculated and stored in either the security block or the security target block's block-type-specific data.

`sop_misconfigured_at_source`

When a security operation is transitioning from being needed to added, it may end up misconfigured. A security operation may be misconfigured if resource exhaustion occurs and there is not appropriate space to store a required security block or other components of the security block. A security operation will also be considered to be misconfigured if any of the required security results cannot be successfully calculated.

2.3.2.2. Security Verifier Events

At a security verifier, five events may be associated with the security operation.

`verifier_for_sop`

A node is designated as a security verifier when a receiver security policy rule is identified which is applicable to the current bundle and is associated with the security verifier role. The security operation described by the rule is then considered to be needed by the security verifier node.

`sop_misconfigured_at_verifier`

A security operation may be identified as misconfigured by the security verifier node. A misconfigured security operation may encounter a resource allocation issue and be unable to add an additional, required security result. A misconfigured security operation may also identify an incorrect security context or parameter, causing a conflict the security policy rule.

`sop_missing_at_verifier`

A security operation may transition from needed to missing if the required security block cannot be located in the bundle by the security verifier node.

`sop_corrupted_at_verifier`

A corrupted security operation is identified as a security block which is unsuccessfully processed by the security verifier node. A corrupted security operation indicates that the security target cannot be verified.

`sop_verified`

When a security operation is designated as processed, the security target has been successfully verified.

2.3.2.3. Security Acceptor Events

At a security acceptor, five events may be associated with the security operation.

`acceptor_for_sop`

A node is designated as a security acceptor when a receiver security policy rule is identified that is both applicable to the current bundle and associated with the security acceptor role. The security operation described by the rule is then considered to be needed by the security acceptor node.

`sop_misconfigured_at_acceptor`

A security operation may be identified as misconfigured by the security acceptor node. A misconfigured security operation signals an incorrect security context or parameter, causing a conflict the security policy rule.

sop_missing_at_acceptor

A security operation is missing if the required security block cannot be located in the bundle by the security acceptor node.

sop_corrupted_at_acceptor

A corrupted security operation is identified as a security block which is unsuccessfully processed by the security acceptor node. A corrupted security operation indicates that the security target cannot be verified and/or decrypted.

sop_processed

When a security operation is designated as processed, the security target has been successfully verified and/or decrypted and is removed from the bundle.

2.3.3. Processing Actions

This section defines a standard set of processing actions that can be specified when defining the policy associated with a security context. The benefit of enumerating these actions is to provide a common set of terminology and design across multiple security contexts with the purpose of making the development of multi-security-context BPSec implementations as similar as possible.

In particular, a security context may wish to override how a BPSec implementation treats the block processing control flags associated with a security block and/or its target block using that context. While the creator of a target block may set block processing flags it may be insecure to process the block in accordance with those flags. Similarly, if a target block has been modified since its was created, it is possible that the target block's processing flags have also been modified and should not necessarily be honored by a receiving BPA.

A security context should specify the situations in which the following actions should be taken by a BPSec implementation at a BPSec-aware node in the network.

remove_sop

When the security operation is removed, it is no longer required for that bundle. If the security operation is the only operation represented by the security block, the block must be removed from the bundle. Otherwise, the security

target's block number is removed from the security block to indicate that the particular operation no longer applies.

`remove_sop_target`

Remove security operation's security target and the security operations associated with that target - The security operation's security target is removed from the bundle, as are any additional security operations associated with that target. For example, if the target block of a confidentiality service is removed, the integrity security operation for that target would be removed as well.

`remove_all_target_sops`

Remove all security operations for the security target - This option is used to remove all of the security operations associated with the security target while retaining the security target in the bundle.

`do_not_forward`

Selecting this option will end bundle transmission at the current node, regardless of bundle destination.

`request_storage`

This option is paired with 'Do Not Forward Bundle' in order to retain a copy of the bundle at the current node. Bundle storage cannot be guaranteed, as node resources are unknown at the time of bundle transmission, but selecting this option will result in an effort to retain a copy of the bundle at the node.

`report_reason_code(code)`

Generate a status report describing the treatment of the bundle and use the provided reason code as the reason.

`override_target_bpcf(mask, new_values)`

Override one or more block processing control flags of the target block and process the block in accordance with the overridden flags. The overridden flags MUST NOT be written back to the target block, and only used for processing the block locally.

Manipulating individual flags to be forced to 0, forced to 1, or kept as specified in the target block requires two inputs. This can be accomplished by the operation:

```
local_flags = (block_flags & mask) | (~mask & values)
```

`override_sop_bpcf(mask, new_values)`

Override one or more block processing control flags of the security block associated with the sop and process the block in accordance with the overridden flags. The overridden flags MUST NOT be written back to the security block, and only used for processing the block locally.

Manipulating individual flags to be forced to 0, forced to 1, or kept as specified in the target block requires two inputs. This can be accomplished by the operation:

```
local_flags = (block_flags & mask) | (~mask & values)
```

2.4. Security Policy Considerations

A security context details the environment in which cryptographic materials are provided to, and retrieved from, the cipher suites used for security services in the network. For this reason, the policy associated with determining proper configuration information must be addressed in the specifications defining new security contexts because this information may differ amongst security contexts.

This section identifies several policy questions that should be addressed in the specification of a security context. In the absence of guidelines for a specific security context, this section defines what should be considered default behavior for any security context.

Target Block Identification

Security contexts should define how a target block of a security service should be identified. This identification is used when determining whether to add a security block at a security source, whether to check a security block at a verifier, and whether a node should consider itself the acceptor of the block.

DEFAULT BEHAVIOR: Unless otherwise specified, a target block MUST be identified at a security source by the three-tuple of {bundle source EID, bundle destination EID, and block type of the target block}. A target block at a security verifier and a security acceptor is identified by these three information elements plus the security context identifier.

Security Parameter Local Overrides

Security contexts should define the circumstances in which a local BPsec-aware node may override parameters in a security block with locally-configured parameters.

DEFAULT BEHAVIOR: Unless otherwise specified, a locally-provided security parameter MUST NOT override a security

parameter present in a security block. A local parameter can only be used in cases where the corresponding parameter is not present in the security block itself.

Security Processing Actions

Security contexts should define the circumstances in which the processing actions defined in Section 2.3.3 should be used and with what inputs.

DEFAULT BEHAVIOR: Unless otherwise specified, the local, BPSec-aware node MUST enforce policy actions as a function of some local policy definition at the node itself. This may be through the definition of generic actions for all security contexts or it may be identified based on a specific security context.

2.5. Security Context Template

This section defines a recommended outline for the definition of a security context for BPSec. The purpose of such an outline is to both ensure that no critical information is omitted when authoring new security contexts and to reduce the cognitive load associated with implementing new security contexts by having them conform to a standard representation.

A security context specification should include the following elements.

1. Overview
2. Interfaces
3. Definitions
4. Canonicalization Algorithms
5. Processing
6. Policy
7. IANA Considerations

A single specification may define one or more security contexts, in which case this information would be repeated for each security context.

The remainder of this section provides information on the topics that should be covered in each of these sections.

2.5.1. Overview

This section should identify the rationale for creating a security context, whether the context uses in-band, out-of-band, or hybrid mechanisms for information exchange, and the unique situations in which the context should be used.

2.5.2. Interfaces

Security context interfaces detail any special considerations or assumptions made when designing the algorithms for creating or otherwise processing the contents of security blocks.

2.5.2.1. Information Provided to the Cipher Suite

This section should identify how the security context interfaces with the cipher suite (or cipher suites) used to process the security service. This may be as simple as providing parameters from the security block to a cipher suite implementation. This may also be a more complex interface involving fusing parameter information carried by the security block with local information to provide an interface to the cipher suite.

A cipher suite may include parameters such as a key length, mode, or number of rounds, which is part of the cipher suite definition. These parameters are considered "locked" parameters and are not to be confused with the "free" parameters that the user may define for each security context. Where not already defined by the cipher suite itself, the security context should clearly identify which parameters should be considered "free" and which should be considered "locked".

In addition to listing cipher suites, this section should identify information relating to all "free" cipher suite parameters. This may include selected bit lengths, which parameters are provided by the local node, which parameters must be present in the security block itself, and how parameters should be protected when represented in a security block.

2.5.2.2. Information Provided by the BPA

A security context specification should describe the information it expects to be provided to it by the local BPA, separate from the contents of the security block and target block from the bundle.

For example, a BPA may additionally provide local the security context parameters.

2.5.2.3. Information Provided to the BPA

A security context specification should describe the information that it will provide back to the local BPA. Typically this is either the output of a cipher suite to be added to a security block in a bundle, or some signal for a process event associated with a verification or processing action.

2.5.3. Definitions

This section defines custom parameters and results associated with the security context and carried either within a security block or as part of local node configuration.

Each security context parameters and result is defined as a key-value pair, where the key is a CBOR-encoded integer and the value is defined as the CBOR encoding of its data type. This section must define any unique parameters and results used by the security context, to include the following information.

- o The integer identifier of the parameter or result item.
- o Whether multiple instances of this item can be present in a security block at one time, and whether at least one instance of this item is required to be defined.
- o Whether this item must be present only in the security block, or whether it can also be included as part of the configuration of a local node, and how to determine which version of the item to use in the event that it is defined in both the security block and the local node.
- o The data type of the item and how that data type must be CBOR encoded for representation in the security block.

2.5.4. Canonicalization Algorithms

Security context canonicalization algorithms take precedence over any others defined. The exact data and its form when provided to the canonicalization algorithm must be determined by the security context.

Consistency and determinism of the block-type-specific data provided as input to the security context is critical for security services to function as expected.

2.5.5. Processing

A security context specification should describe how the context should be used to perform every processing action described in Section 2.3.2.

2.5.6. Policy

A security context specification should describe how the context should be configured from a policy perspective. This should include, at a minimum, under which circumstances each policy action identified in Section 2.3.3 should be taken.

2.5.7. IANA Considerations

Each security context MUST provide an entry in the IANA security context identifier registry to uniquely identify the identifiers of each security context.

3. Normative References

[I-D.ietf-dtn-bpbis]

Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-25 (work in progress), May 2020.

[I-D.ietf-dtn-bpsec]

Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-22 (work in progress), March 2020.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Sarah Heiner
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 240 592 3704
Email: Sarah.Heiner@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: 1 May 2021

B. Sipos
RKF Engineering
28 October 2020

DTN Bundle Protocol Security COSE Security Contexts
draft-bsipos-dtn-bpsec-cose-02

Abstract

This document defines a security context suitable for using CBOR Object Signing and Encryption (COSE) algorithms within Bundle Protocol Security (BPsec) integrity and confidentiality blocks. A profile of COSE is also defined for BPsec interoperation.

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 1 May 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. BPsec Security Context	3
3.1. COSE Integrity	3
3.2. COSE Confidentiality	4
4. COSE Profile for BPsec	6
4.1. COSE Security Parameters	6
4.2. COSE Security Results	6
4.3. Interoperability Algorithms	7
5. Implementation Status	8
6. Security Considerations	9
6.1. Threat: BPsec Block Replay	9
6.2. Threat: Unidentifiable Key	10
6.3. Threat: Algorithm Vulnerabilities	10
7. IANA Considerations	10
7.1. BPsec Security Contexts	10
8. Acknowledgments	10
9. References	10
9.1. Normative References	10
9.2. Informative References	12
Appendix A. Examples	12
A.1. Symmetric Key COSE_Mac0	12
A.2. RSA Keypair COSE_Sign1	14
A.3. Symmetric Key COSE_Encrypt0	16
A.4. Symmetric KEK COSE_Encrypt	18
Author's Address	21

1. Introduction

The Bundle Protocol Security (BPsec) Specification [I-D.ietf-dtn-bpsec] defines structure and encoding for Block Integrity Block (BIB) and Block Confidentiality Block (BCB) types but does not specify any security contexts to be used by either of the security block types. The CBOR Object Signing and Encryption (COSE) specification [RFC8152] defines a structure, encoding, and algorithms to use for cryptographic signing and encryption.

This document describes how to use the algorithms and encodings of COSE within BPsec blocks to apply those algorithms to Bundle security in Section 3. A bare minimum of interoperability algorithms and algorithm parameters is specified by this document in Section 4.

This document does not address how those COSE algorithms are intended to be used within a larger security context. Examples of specific uses are provided in Appendix A to aid in implementation support of the interoperability algorithms.

2. 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.

3. BPSec Security Context

This document specifies a single security context for use in both BPsec integrity and confidentiality blocks. This is done to save code points allocated to this specification and to simplify the encoding of COSE-in-BPsec; the BPsec block type uniquely defines the acceptable COSE messages which can be present and each COSE message is type-tagged to indicate its purpose and contents.

The COSE security context shall have the Security Context ID specified in Section 7.1.

The existing CoAP Content-Format ID values in the CoRE registry [IANA-CORE] SHALL be used as BPsec Parameter ID and Result ID values within COSE security context (see tables within Section 3.1 and Section 3.2). For Result ID values used to identify COSE messages, these code points are also identical to the existing COSE message-marking tags in Section 2 of [RFC8152]. This avoids the need for value-mapping between code points of the two registries.

When embedding COSE messages, the CBOR-tagged form SHALL NOT be used. The Result ID values already provide the same information as the COSE tags (using the same code points).

3.1. COSE Integrity

When used within a Block Integrity Block, COSE context SHALL allow only the Parameter IDs from Table 1. Each integrity parameter value SHALL consist of the COSE structure indicated by Table 1 in its decoded form.

Parameter ID	Parameter Structure	Reference
101	COSE_Key	[RFC8152]
102	COSE_KeySet	[RFC8152]

Table 1: COSE Integrity Parameters

When used within a Block Integrity Block, COSE context SHALL allow only the Result IDs from Table 2. Each integrity result value SHALL consist of the COSE message indicated by Table 2 in its decoded form.

Result ID	Result Structure	Reference
97	COSE_Mac	[RFC8152]
17	COSE_Mac0	[RFC8152]
98	COSE_Sign	[RFC8152]
18	COSE_Sign1	[RFC8152]

Table 2: COSE Integrity Results

Each integrity result SHALL use the "detached" payload form with nil payload value. The integrity result for COSE_Mac and COSE_Mac0 messages are computed by the procedure in Section 6.3 of [RFC8152]. The integrity result for COSE_Sign and COSE_Sign1 messages are computed by the procedure in Section 4.4 of [RFC8152].

[NOTE: This differs from base BPsec in that the entire block and the bundle primary is signed] The COSE "payload" used to generate a signature or MAC result SHALL be the canonically serialized target block, including the canonical block array structure. The COSE "protected attributes from the application" used to generate a signature or MAC result SHALL be either:

For a primary block target: An empty byte string.

For a canonical block target: The canonically serialized primary block of the bundle.

3.2. COSE Confidentiality

When used within a Block Confidentiality Block, COSE context SHALL allow only the Parameter IDs from Table 3. Each integrity parameter value SHALL consist of the COSE structure indicated by Table 3 in its decoded form.

Parameter ID	Parameter Structure	Reference
101	COSE_Key	[RFC8152]
102	COSE_KeySet	[RFC8152]

Table 3: COSE Integrity Parameters

When used within a Block Confidentiality Block, COSE context SHALL allow only the Result IDs from Table 4. Each confidentiality result value SHALL consist of the COSE message indicated by Table 4 in its decoded form.

Result ID	Result Structure	Reference
96	COSE_Encrypt	[RFC8152]
16	COSE_Encrypt0	[RFC8152]

Table 4: COSE Confidentiality Results

Only algorithms which support Authenticated Encryption with Authenticated Data (AEAD) SHALL be usable in the first (content) layer of a confidentiality result. Because COSE encryption with AEAD appends the authentication tag with the ciphertext, the size of the block-type-specific-data will grow after an encryption operation.

Each confidentiality result SHALL use the "detached" payload form with nil payload value. The COSE plaintext and ciphertext correspond exactly with the target block-type-specific-data. The confidentiality result for COSE_Encrypt and COSE_Encrypt0 messages are computed by the procedure in Section 5.3 of [RFC8152].

[NOTE: This differs from base BPsec in that AAD from the block and the bundle primary is used] The COSE "plaintext" used to generate an encrypt result SHALL be the block-type-specific-data of the target block, the decoded byte string itself (not including the encoded CBOR item header). The COSE "protected attributes from the application" used to generate an encrypt result SHALL be the concatenation of the following:

1. The canonically serialized primary block of the bundle.

2. The canonically serialized augmented target block, which has its block-type-specific-data substituted with an empty byte string.

4. COSE Profile for BPSec

This section contains requirements which apply to the use of COSE within BPSec across any security context use.

4.1. COSE Security Parameters

When necessary to support public key infrastructure (PKI) within BPSec, a BIB or BCB with a COSE context MAY contain one or more public keys or key identifiers. Because each context contains a single set of parameters which apply to all results in the same context, security acceptors SHALL treat all COSE keys as being related to the security source itself and potentially applying to every result.

4.2. COSE Security Results

When generating a BPSec result, security sources SHALL use encode COSE labels with a uint value. When processing a BPSec result, security acceptors MAY handle COSE labels with with a tstr value.

When used in a BPSec result, each COSE message SHALL contain an explicit algorithm identifier in the lower (content) layers. When available and not implied by the bundle source, a COSE message SHOULD contain a key identifier in the highest (recipient) layer. When a key identifier is not available, BPSec acceptors SHOULD use the Security Source (if available) and the Bundle Source to imply which keys can be used for security operations. A BPSec security operation always occurs within the context of the immutable primary block with its parameters (specifically the Source Node ID) and the security block with its optional Security Source.

The algorithms required by this profile focuses on networks using shared symmetric-keys, with recommended algorithms for Elliptic Curve (EC) keypairs and RSA keypairs. The focus of this profile is to enable interoperation between security sources and acceptors on an open network, where more explicit COSE parameters make it easier for BPSec acceptors to avoid assumptions and avoid out-of-band parameters. The requirements of this profile still allow the use of potentially not-easily-interoperable algorithms and message/recipient configurations for use by private networks, where message size is more important than explicit COSE parameters.

4.3. Interoperability Algorithms

[NOTE: The required list is identical to the [I-D.ietf-dtn-bpsec-interop-sc] list.] The set of integrity algorithms needed for interoperability is listed here. The full set of COSE algorithms available is managed at [IANA-COSE].

Implementations conforming to this specification SHALL support the symmetric keyed algorithms of Table 5. Implementations capable of doing so SHOULD support the asymmetric keyed and key-encryption algorithms of Table 5.

BPsec Block	COSE Layer	Name	Code	Implementation Requirements
Integrity	1	HMAC 256/256	5	Required
Integrity	1	ES256	-7	Recommended
Integrity	1	EdDSA	-8	Recommended
Integrity	1	PS256	-37	Recommended
Confidentiality	1	A256GCM	3	Required
Integrity or Confidentiality	2	A256KW	-5	Recommended
Integrity or Confidentiality	2	ECDH-ES + A256KW	-31	Recommended
Integrity or Confidentiality	2	RSAES-OAEP w/ SHA-256	-41	Recommended

Table 5: Interoperability Algorithms

The following are recommended key and recipient uses within COSE/BPsec:

Symmetric Key Integrity: When generating a BIB result from a symmetric key, implementations SHOULD use either a COSE_Mac0 or a COSE_Mac using the private key directly. When a COSE_Mac is used with a direct key, the recipient layer SHOULD include a key identifier.

EC Keypair Integrity: When generating a BIB result from an EC keypair, implementations SHOULD use either a COSE_Sign1 or a COSE_Sign using the private key directly or a COSE_Mac from a symmetric key with a layer-2 encryption of the symmetric key. When a COSE_Sign or COSE_Mac is used with EC keypair, the recipient layer SHOULD include a public key identifier.

RSA Keypair Integrity: When generating a BIB result from an RSA keypair, implementations SHOULD use either a COSE_Sign1 or a COSE_Sign using the private key directly or a COSE_Mac from a symmetric key with a layer-2 key-wrap of the symmetric key. When a COSE_Sign or COSE_Mac is used with RSA keypair, the recipient layer SHOULD include a public key identifier. When a COSE_Sign or COSE_Sign1 is used with RSA keypair, the signature uses a maximum-length PSS salt in accordance with [RFC8230].

Symmetric Key Confidentiality: When generating a BCB result from an symmetric key, implementations SHOULD use a COSE_Encrypt message with a recipient containing a key-wrapped CEK. When generating a BCB result from a symmetric key, implementations SHOULD NOT use COSE_Encrypt0 or COSE_Encrypt with direct content encryption key (CEK). Doing so risks key overuse and the vulnerabilities associated with large amount of ciphertext from the same key.

EC Keypair Confidentiality: When generating a BCB result from an EC keypair, implementations SHOULD use a COSE_Encrypt message with a recipient containing a key-wrapped CEK.

RSA Keypair Confidentiality: When generating a BCB result from an RSA keypair, implementations SHOULD use a COSE_Encrypt message with a recipient containing a key-wrapped CEK.

5. Implementation Status

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [RFC7942] and [github-dtn-bpsec-cose].]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations can exist.

An example implementation of COSE over Blocks has been created as a GitHub project [github-dtn-bpsec-cose] and is intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation only handles CBOR encoding/decoding and cryptographic functions, it does not construct actual BIB or BCB and does not integrate with a BP Agent.

6. Security Considerations

This section separates security considerations into threat categories based on guidance of BCP 72 [RFC3552].

All of the security considerations of the underlying BPSec [I-D.ietf-dtn-bpsec] apply to these new security contexts.

6.1. Threat: BPSec Block Replay

The bundle's primary block contains fields which uniquely identify a bundle: the Source Node ID, Creation Timestamp, and fragment parameters (see Section 4.2.2 of [I-D.ietf-dtn-bpbis]). These same fields are used to correlate Administrative Records with the bundles for which the records were generated. Including the primary block in the AAD for BPSec integrity and confidentiality binds the verification of the secured block to its parent bundle and disallows replay of any block with its BIB or BCB.

This profile of COSE limits the encryption algorithms to only AEAD in order to include the context of the encrypted data as AAD. If an agent mistakenly allows the use of non-AEAD encryption when decrypting and verifying a BCB, the possibility of block replay attack is present.

6.2. Threat: Unidentifiable Key

The profile in Section 4.3 recommends key identifiers when possible and the parameters in section Section 4.1 allow encoding public keys where available. If the application using a COSE Integrity or COSE Confidentiality context leaves out key identification data (in a COSE recipient structure), the security acceptor for those BPsec blocks only has the primary block available to use when verifying or decrypting the target block. This leads to a situation, identified in BPsec Security Considerations, where a signature is verified to be valid but not from the expected Security Source.

6.3. Threat: Algorithm Vulnerabilities

Because this use of COSE leaves the specific algorithms chosen for BIB and BCB use up to the applications securing bundle data, it is important to use only COSE algorithms which are marked as recommended in the IANA registry [IANA-COSE].

7. IANA Considerations

Registration procedures referred to in this section are defined in [RFC8126].

7.1. BPsec Security Contexts

Within the "Bundle Protocol" registry [IANA-BUNDLE], the following entry has been added to the "BPsec Security Context Identifiers" sub-registry.

Value	Description	Reference
TBD-COSE	COSE	This specification.

Table 6

8. Acknowledgments

The interoperability minimum algorithms and parameters are based on the draft [I-D.ietf-dtn-bpsec-interop-sc].

9. References

9.1. Normative References

- [IANA-BUNDLE]
IANA, "Bundle Protocol",
<<https://www.iana.org/assignments/bundle/>>.
- [IANA-CORE]
IANA, "Constrained RESTful Environments (CoRE)
Parameters",
<<https://www.iana.org/assignments/core-parameters/>>.
- [IANA-COSE]
IANA, "CBOR Object Signing and Encryption (COSE)",
<<https://www.iana.org/assignments/cose/>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)",
RFC 8152, DOI 10.17487/RFC8152, July 2017,
<<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.
- [RFC8230] Jones, M., "Using RSA Algorithms with CBOR Object Signing
and Encryption (COSE) Messages", RFC 8230,
DOI 10.17487/RFC8230, September 2017,
<<https://www.rfc-editor.org/info/rfc8230>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data
Definition Language (CDDL): A Notational Convention to
Express Concise Binary Object Representation (CBOR) and
JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610,
June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [I-D.ietf-dtn-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security
Specification", Work in Progress, Internet-Draft, draft-
ietf-dtn-bpsec-22, 10 March 2020,
<<https://tools.ietf.org/html/draft-ietf-dtn-bpsec-22>>.

9.2. Informative References

- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [I-D.ietf-dtn-bpbis] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", Work in Progress, Internet-Draft, draft-ietf-dtn-bpbis-26, 28 July 2020, <<https://tools.ietf.org/html/draft-ietf-dtn-bpbis-26>>.
- [I-D.ietf-dtn-bpsec-interop-sc] Birrane, E., "BPsec Interoperability Security Contexts", Work in Progress, Internet-Draft, draft-ietf-dtn-bpsec-interop-sc-01, 4 February 2020, <<https://tools.ietf.org/html/draft-ietf-dtn-bpsec-interop-sc-01>>.
- [github-dtn-bpsec-cose] Sipos, B., "DTN Bundle Protocol Security COSE Security Contexts", <<https://github.com/BSipos-RKF/dtn-bpsec-cose/>>.

Appendix A. Examples

A.1. Symmetric Key COSE_Mac0

This is an example of a MAC with implied recipient (and its key material). The provided figures are extended diagnostic notation [RFC8610].

The 256-bit key used is shown below.

```
[
  {
    / kty / 1: 4, / symmetric /
    / kid / 2: 'ExampleMAC',
    / k / -1: h'13bf9cead057c0aca2c9e52471ca4b19ddfaf4c0784e3f3e8e3999db
      ae4ce45c'
  }
]
```

Figure 1: Symmetric Key

```
[
  7, / BP version /
  0, / flags /
  0, / CRC type /
  [1, "///dst/svc"], / destination /
  [1, "///src/bp"], / source /
  [1, "///src/bp"], / report-to /
  [0, 40], / timestamp /
  1000000 / lifetime /
]
```

Figure 2: Primary block CBOR diagnostic

```
[
  7, / type code - bundle age /
  2, / block num /
  0, / flags /
  0, / CRC type /
  <<300>> / type-specific-data: age /
]
```

Figure 3: Target block CBOR diagnostic

The external_aad is the encoded primary block. The payload is the encoded target block.

```
[
  "MAC0", / context /
  h'a10105', / protected /
  h'880700008201692f2f6473742f7376638201682f2f7372632f62708201682f2f7372
    632f6270820018281a000f4240', / external_aad /
  h'85070200004319012c' / payload /
]
```

Figure 4: MAC_structure CBOR diagnostic


```

[
  [2], / targets /
  0, / security context TBD /
  0, / flags /
  [
    [ / target block #2 /
      [ / result /
        17, / COSE_Mac0 tag /
        [
          <<{ / protected /
            / alg / 1:5 / HMAC 256//256 /
          }>>,
          { / unprotected /
            / kid / 4:'ExampleMAC'
          },
          null, / payload /
          h'1349a33b41b020e46669b714b53a1b79db458fdef0f0b7a0daebde6baf27
            7472' / tag /
        ]
      ]
    ]
  ]
]

```

Figure 5: Abstract Security Block CBOR diagnostic

A.2. RSA Keypair COSE_Sign1

This is an example of a signature with implied recipient (and its key material). The provided figures are extended diagnostic notation [RFC8610].

The 512-bit private key used is below. It is not supposed to be a secure configuration, only intended to explain the procedure. This signature uses zero-length salt for deterministic output, which differs from the parameter specified by [RFC8230] and is not recommended for normal use.

```
[
  { / signing private key /
    / kty / 1: 3, / RSA /
    / kid / 2: 'ExampleRSA',
    / n / -1: b64'3bUZ1LR9oBiBpx6lGZuvtMBPTAS5qGosF8A7QODUz13fs71PH0e9nDY4RwurZZO
9_QqNrUlamp2gmbXsuCGE-Q',
    / e / -2: b64'AQAB',
    / d / -3: b64'yCQmj2foSFAXKuB1Nmre8RLyArP5Td08lSxJ0UWllixmFRoso_2jHIjGXci8rmJ
LSgCxbSeojtoxwGg-bFmlAQ',
    / p / -4: b64'7snebs70tMJ67A1qA4Yk5ujvjyaDEIsfch_fRwVIVik',
    / q / -5: b64'7bAM_t782esDusNKAzr5EQaa3wjTQ2CUXBKEFSLgc1E',
    / dP / -6: b64'Iiay7kwhCV0rMW1luQ1NZ8z2vhV29z2-gJb4WvLxdok',
    / dQ / -7: b64'bC7WK2dJBKv9uCOHlxIIItSzxtIYfjFGNYD8i7Wo5E',
    / qInv / -8: b64'6efvn6d0ADFQJxNLqjRJyE5E1m_dYQEvCI2mAqixshA'
  }
]
```

Figure 6: Private Keys

```
[
  7, / BP version /
  0, / flags /
  0, / CRC type /
  [1, "dst/svc"], / destination /
  [1, "src/bp"], / source /
  [1, "src/bp"], / report-to /
  [0, 40], / timestamp /
  1000000 / lifetime /
]
```

Figure 7: Primary block CBOR diagnostic

```
[
  7, / type code - bundle age /
  2, / block num /
  0, / flags /
  0, / CRC type /
  <<300>> / type-specific-data: age /
]
```

Figure 8: Target block CBOR diagnostic

The external_aad is the encoded primary block. The payload is the encoded target block.

```
[
  "Signature1", / context /
  h'a1013824', / protected /
  h'880700008201692f2f6473742f7376638201682f2f7372632f62708201682f2f7372
    632f6270820018281a000f4240', / external_aad /
  h'85070200004319012c' / payload /
]
```

Figure 9: Sig_structure CBOR diagnostic

```
[
  [2], / targets /
  0, / security context TBD /
  0, / flags /
  [
    [ / target block #2 /
      [ / result /
        18, / COSE_Sign1 tag /
        [
          <<{ / protected /
            / alg / 1:-37 / PS256 /
          }>>,
          { / unprotected /
            / kid / 4:'ExampleRSA'
          },
          null, / payload /
          h'53d983df0590f529456b661d36f217d722aa88497f04779385a9a786693d
            518778a23b912e02e272ea120adf0c1ddf2e08fb5efc54c1f6d36a95054b
            745fa47e' / signature /
        ]
      ]
    ]
  ]
]
```

Figure 10: Abstract Security Block CBOR diagnostic

A.3. Symmetric Key COSE_Encrypt0

This is an example of an encryption with implied recipient (and its direct content encryption key). The provided figures are extended diagnostic notation [RFC8610].

This example uses a single shared content encryption key, which is not recommended for normal use. The 256-bit key used is shown below. A random IV is generated for this operation and is indicated in a standard way in the unprotected header.

```
[
  {
    / kty / 1: 4, / symmetric /
    / kid / 2: 'ExampleCEK',
    / k / -1: h'13bf9cead057c0aca2c9e52471ca4b19ddfaf4c0784e3f3e8e3999db
              ae4ce45c'
  }
]
```

Figure 11: Symmetric Keys

```
[
  7, / BP version /
  0, / flags /
  0, / CRC type /
  [1, "dst/svc"], / destination /
  [1, "src/bp"], / source /
  [1, "src/bp"], / report-to /
  [0, 40], / timestamp /
  1000000 / lifetime /
]
```

Figure 12: Primary block CBOR diagnostic

```
[
  7, / type code - bundle age /
  2, / block num /
  0, / flags /
  0, / CRC type /
  <<300>> / type-specific-data: age /
]
```

Figure 13: Initial Target block CBOR diagnostic

The external_aad is a concatenation of the encoded primary block and the encoded augmented target block (its block data removed).

```
[
  "Encrypt0", / context /
  h'a10103', / protected /
  h'880700008201692f2f6473742f7376638201682f2f7372632f62708201682f2f7372
    632f6270820018281a000f4240850702000040' / external_aad /
]
```

Figure 14: Enc_structure CBOR diagnostic

```

[
  [2], / targets /
  0, / security context TBD /
  0, / flags /
  [
    [ / target block #2 /
      [ / result /
        16, / COSE_Encrypt0 tag /
        [
          <<{ / protected /
            / alg / 1:3 / A256GCM /
          }>>,
          { / unprotected /
            / kid / 4:'ExampleCEK',
            / iv / 5: h'6f3093eba5d85143c3dc484a'
          },
          null / payload /
        ]
      ]
    ]
  ]
]

```

Figure 15: Abstract Security Block CBOR diagnostic

```

[
  7, / type code - bundle age /
  2, / block num /
  0, / flags /
  0, / CRC type /
  h'63bb1617fc5076cec266907a7143d28587f04e' / ciphertext /
]

```

Figure 16: Encrypted Target block CBOR diagnostic

A.4. Symmetric KEK COSE_Encrypt

This is an example of an encryption with a random CEK and an explicit key-encryption key (KEK) identified by a Key ID. The provided figures are extended diagnostic notation [RFC8610].

The keys used are shown in Figure 17. A random IV is generated for this operation and is indicated in a standard way in the unprotected header of Figure 21.

```
[
  {
    / kty / 1: 4, / symmetric /
    / kid / 2: 'ExampleKEK',
    / k / -1: h'0e8a982b921d1086241798032fedc1f883eab72e4e43bb2d11cfae38
      ad7a972e'
  },
  {
    / kty / 1: 4, / symmetric /
    / kid / 2: 'ExampleCEK',
    / k / -1: h'13bf9cead057c0aca2c9e52471ca4b19ddfaf4c0784e3f3e8e3999db
      ae4ce45c'
  }
]
```

Figure 17: Symmetric Keys

```
[
  7, / BP version /
  0, / flags /
  0, / CRC type /
  [1, "///dst/svc"], / destination /
  [1, "///src/bp"], / source /
  [1, "///src/bp"], / report-to /
  [0, 40], / timestamp /
  1000000 / lifetime /
]
```

Figure 18: Primary block CBOR diagnostic

```
[
  7, / type code - bundle age /
  2, / block num /
  0, / flags /
  0, / CRC type /
  <<300>> / type-specific-data: age /
]
```

Figure 19: Initial Target block CBOR diagnostic

The `external_aad` is a concatenation of the encoded primary block and the encoded augmented target block (its block data removed).

The CEK and content plaintext are the same here as in Figure 14 but the context text is different.

```
[
  "Encrypt", / context /
  h'a10103', / protected /
  h'880700008201692f2f6473742f7376638201682f2f7372632f62708201682f2f7372
    632f6270820018281a000f4240850702000040' / external_aad /
]
```

Figure 20: Enc_structure CBOR diagnostic

```
[
  [2], / targets /
  0, / security context TBD /
  0, / flags /
  [
    [ / target block #2 /
      [ / result /
        96, / COSE_Encrypt tag /
        [
          <<{ / protected /
            / alg / 1:3 / A256GCM /
          }>>,
          { / unprotected /
            / iv / 5: h'6f3093eba5d85143c3dc484a'
          },
          null, / payload /
          [
            [ / recipient /
              h'', / protected /
              { / unprotected /
                / alg / 1:-5, / A256KW /
                / kid / 4:'ExampleKEK'
              },
              h'917f2045e1169502756252bf119a94cdac6a9d8944245b5a9a26d403
                a6331159e3d691a708e9984d', / key-wrapped /
              [] / no more layers /
            ]
          ]
        ]
      ]
    ]
  ]
]
```

Figure 21: Abstract Security Block CBOR diagnostic

Although the same CEK is used in this example as the Encrypt0 example, the block ciphertext is different than Figure 16 because the Enc_structure (used as AAD) is different.

```
[  
  7, / type code - bundle age /  
  2, / block num /  
  0, / flags /  
  0, / CRC type /  
  h'63bb160aa1804f936570b982bf7c396694e574' / ciphertext /  
]
```

Figure 22: Encrypted Target block CBOR diagnostic

Author's Address

Brian Sipos
RKf Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
United States of America

Email: BSipos@rkf-eng.com

Delay-Tolerant Networking Working Group
Internet Draft
Intended status: Standards Track
Expires: May 3, 2021

S. Burleigh
JPL, Calif. Inst. Of Technology
K. Fall
Roland Computing Services
E. Birrane
APL, Johns Hopkins University
October 30, 2020

Bundle Protocol Version 7
draft-ietf-dtn-bpbis-28.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on May 3, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This Internet Draft presents a specification for the Bundle Protocol, adapted from the experimental Bundle Protocol specification developed by the Delay-Tolerant Networking Research group of the Internet Research Task Force and documented in RFC 5050.

Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	5
3. Service Description.....	5
3.1. Definitions.....	5
3.2. Discussion of BP concepts.....	9
3.3. Services Offered by Bundle Protocol Agents.....	12
4. Bundle Format.....	13
4.1. BP Fundamental Data Structures.....	14
4.1.1. CRC Type.....	14
4.1.2. CRC.....	14
4.1.3. Bundle Processing Control Flags.....	14
4.1.4. Block Processing Control Flags.....	16
4.1.5. Identifiers.....	17
4.1.5.1. Endpoint ID.....	17
4.1.5.1.1. The "dtn" URI scheme.....	18
4.1.5.1.2. The "ipn" URI scheme.....	20
4.1.5.2. Node ID.....	21
4.1.6. DTN Time.....	22
4.1.7. Creation Timestamp.....	22
4.1.8. Block-type-specific Data.....	23
4.2. Bundle Representation.....	23
4.2.1. Bundle.....	23
4.2.2. Primary Bundle Block.....	23
4.2.3. Canonical Bundle Block Format.....	26
4.3. Extension Blocks.....	27
4.3.1. Previous Node.....	27
4.3.2. Bundle Age.....	27
4.3.3. Hop Count.....	28
5. Bundle Processing.....	28
5.1. Generation of Administrative Records.....	29
5.2. Bundle Transmission.....	30
5.3. Bundle Dispatching.....	30
5.4. Bundle Forwarding.....	30

5.4.1. Forwarding Contraindicated.....	32
5.4.2. Forwarding Failed.....	33
5.5. Bundle Expiration.....	33
5.6. Bundle Reception.....	34
5.7. Local Bundle Delivery.....	35
5.8. Bundle Fragmentation.....	36
5.9. Application Data Unit Reassembly.....	37
5.10. Bundle Deletion.....	37
5.11. Discarding a Bundle.....	38
5.12. Canceling a Transmission.....	38
6. Administrative Record Processing.....	38
6.1. Administrative Records.....	38
6.1.1. Bundle Status Reports.....	39
6.2. Generation of Administrative Records.....	42
7. Services Required of the Convergence Layer.....	42
7.1. The Convergence Layer.....	42
7.2. Summary of Convergence Layer Services.....	42
8. Implementation Status.....	43
9. Security Considerations.....	45
10. IANA Considerations.....	46
10.1. Bundle Block Types.....	47
10.2. Primary Bundle Protocol Version.....	48
10.3. Bundle Processing Control Flags.....	48
10.4. Block Processing Control Flags.....	50
10.5. Bundle Status Report Reason Codes.....	52
10.6. Bundle Protocol URI scheme types.....	53
10.7. URI scheme "dtn".....	54
10.8. URI scheme "ipn".....	55
11. References.....	56
11.1. Normative References.....	56
11.2. Informative References.....	57
12. Acknowledgments.....	57
13. Significant Changes from RFC 5050.....	58
Appendix A. For More Information.....	59
Appendix B. CDDL expression.....	60

1. Introduction

Since the publication of the Bundle Protocol Specification (Experimental RFC 5050 [RFC5050]) in 2007, the Delay-Tolerant Networking (DTN) Bundle Protocol has been implemented in multiple programming languages and deployed to a wide variety of computing platforms. This implementation and deployment experience has identified opportunities for making the protocol simpler, more capable, and easier to use. The present document, standardizing the Bundle Protocol (BP), is adapted from RFC 5050 in that context,

reflecting lessons learned. Significant changes from the Bundle Protocol specification defined in RFC 5050 are listed in section 13.

This document describes version 7 of BP.

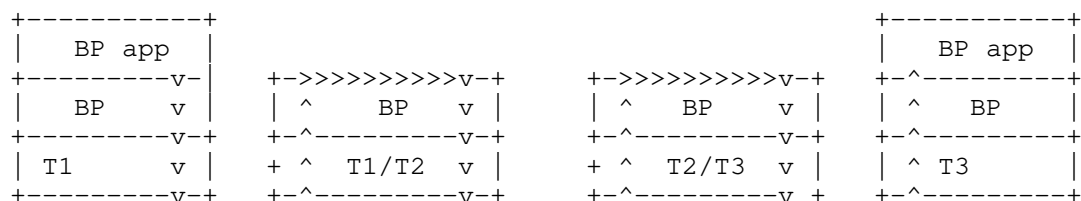
Delay Tolerant Networking is a network architecture providing communications in and/or through highly stressed environments. Stressed networking environments include those with intermittent connectivity, large and/or variable delays, and high bit error rates. To provide its services, BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network. Key capabilities of BP include:

- . Ability to use physical motility for the movement of data
- . Ability to move the responsibility for error control from one node to another
- . Ability to cope with intermittent connectivity, including cases where the sender and receiver are not concurrently present in the network
- . Ability to take advantage of scheduled, predicted, and opportunistic connectivity, whether bidirectional or unidirectional, in addition to continuous connectivity
- . Late binding of overlay network endpoint identifiers to underlying constituent network addresses

For descriptions of these capabilities and the rationale for the DTN architecture, see [ARCH] and [SIGC].

BP's location within the standard protocol stack is as shown in Figure 1. BP uses underlying "native" transport and/or network protocols for communications within a given constituent network. The layer at which those underlying protocols are located is here termed the "convergence layer" and the interface between the bundle protocol and a specific underlying protocol is termed a "convergence layer adapter".

Figure 1 shows three distinct transport and network protocols (denoted T1/N1, T2/N2, and T3/N3).



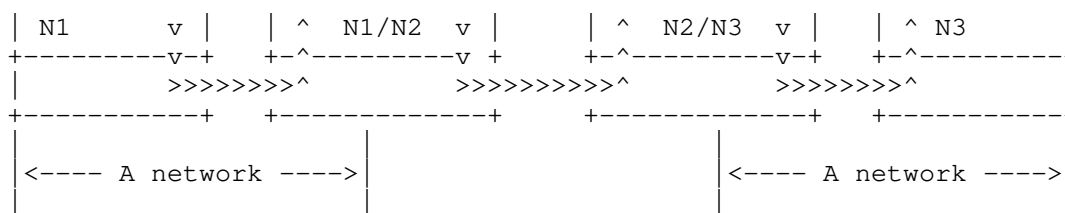


Figure 1: The Bundle Protocol in the Protocol Stack Model

This document describes the format of the protocol data units (called "bundles") passed between entities participating in BP communications.

The entities are referred to as "bundle nodes". This document does not address:

- . Operations in the convergence layer adapters that bundle nodes use to transport data through specific types of internets. (However, the document does discuss the services that must be provided by each adapter at the convergence layer.)
- . The bundle route computation algorithm.
- . Mechanisms for populating the routing or forwarding information bases of bundle nodes.
- . The mechanisms for securing bundles en route.
- . The mechanisms for managing bundle nodes.

Note that implementations of the specification presented in this document will not be interoperable with implementations of RFC 5050.

2. Conventions used in this document

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.

3. Service Description

3.1. Definitions

Bundle - A bundle is a protocol data unit of BP, so named because negotiation of the parameters of a data exchange may be impractical in a delay-tolerant network: it is often better practice to "bundle" with a unit of application data all metadata that might be needed in order to make the data immediately usable when delivered to the

application. Each bundle comprises a sequence of two or more "blocks" of protocol data, which serve various purposes.

Block - A bundle protocol block is one of the protocol data structures that together constitute a well-formed bundle.

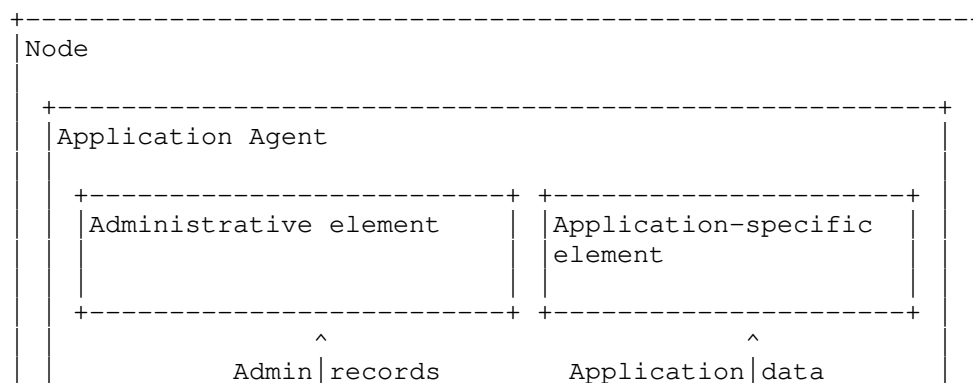
Application Data Unit (ADU) - An application data unit is the unit of data whose conveyance to the bundle's destination is the purpose for the transmission of some bundle that is not a fragment (as defined below).

Bundle payload - A bundle payload (or simply "payload") is the content of the bundle's payload block. The terms "bundle content", "bundle payload", and "payload" are used interchangeably in this document. For a bundle that is not a fragment (as defined below), the payload is an application data unit.

Partial payload - A partial payload is a payload that comprises either the first N bytes or the last N bytes of some other payload of length M, such that $0 < N < M$. Note that every partial payload is a payload and therefore can be further subdivided into partial payloads.

Fragment - A fragment, a.k.a. "fragmentary bundle", is a bundle whose payload block contains a partial payload.

Bundle node - A bundle node (or, in the context of this document, simply a "node") is any entity that can send and/or receive bundles. Each bundle node has three conceptual components, defined below, as shown in Figure 2: a "bundle protocol agent", a set of zero or more "convergence layer adapters", and an "application agent". ("CL1 PDUs" are the PDUs of the convergence-layer protocol used in network 1.)



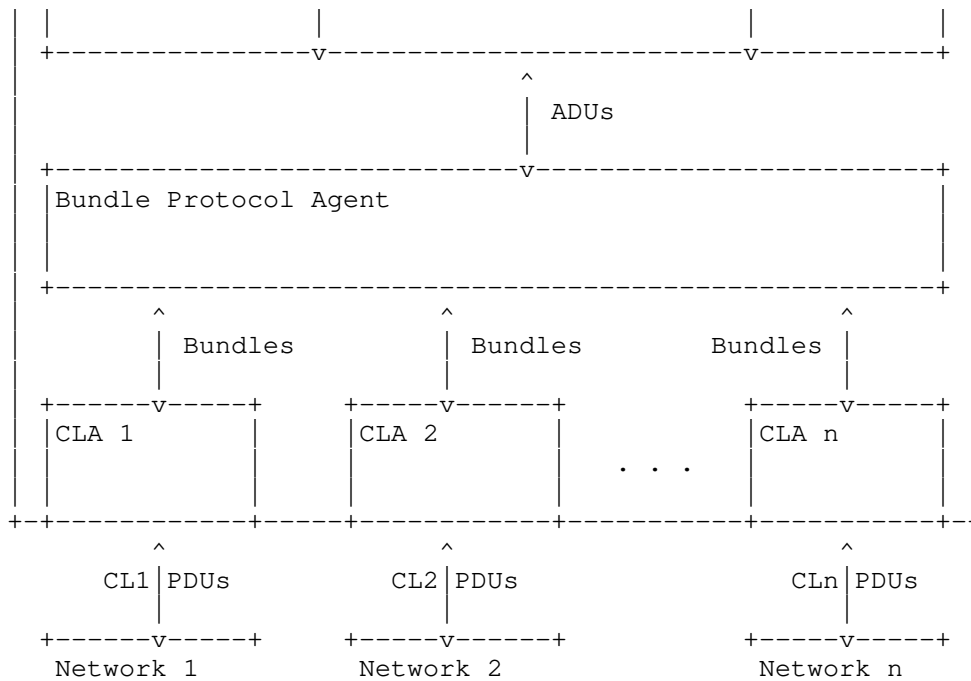


Figure 2: Components of a Bundle Node

Bundle protocol agent - The bundle protocol agent (BPA) of a node is the node component that offers the BP services and executes the procedures of the bundle protocol.

Convergence layer adapter - A convergence layer adapter (CLA) is a node component that sends and receives bundles on behalf of the BPA, utilizing the services of some 'native' protocol stack that is supported in one of the networks within which the node is functionally located.

Application agent - The application agent (AA) of a node is the node component that utilizes the BP services to effect communication for some user purpose. The application agent in turn has two elements, an administrative element and an application-specific element.

Application-specific element - The application-specific element of an AA is the node component that constructs, requests transmission of, accepts delivery of, and processes units of user application data.

Administrative element - The administrative element of an AA is the node component that constructs and requests transmission of administrative records (defined below), including status reports, and accepts delivery of and processes any administrative records that the node receives.

Administrative record - A BP administrative record is an application data unit that is exchanged between the administrative elements of nodes' application agents for some BP administrative purpose. The only administrative record defined in this specification is the status report, discussed later.

Bundle endpoint - A bundle endpoint (or simply "endpoint") is a set of zero or more bundle nodes that all identify themselves for BP purposes by some common identifier, called a "bundle endpoint ID" (or, in this document, simply "endpoint ID"; endpoint IDs are described in detail in Section 4.5.5.1 below.

Singleton endpoint - A singleton endpoint is an endpoint that always contains exactly one member.

Registration - A registration is the state machine characterizing a given node's membership in a given endpoint. Any single registration has an associated delivery failure action as defined below and must at any time be in one of two states: Active or Passive. Registrations are local; information about a node's registrations is not expected to be available at other nodes, and the Bundle Protocol does not include a mechanism for distributing information about registrations.

Delivery - A bundle is considered to have been delivered at a node subject to a registration as soon as the application data unit that is the payload of the bundle, together with any relevant metadata (an implementation matter), has been presented to the node's application agent in a manner consistent with the state of that registration.

Deliverability - A bundle is considered "deliverable" subject to a registration if and only if (a) the bundle's destination endpoint is the endpoint with which the registration is associated, (b) the bundle has not yet been delivered subject to this registration, and (c) the bundle has not yet been "abandoned" (as defined below) subject to this registration.

Abandonment - To abandon a bundle subject to some registration is to assert that the bundle is not deliverable subject to that registration.

Delivery failure action - The delivery failure action of a registration is the action that is to be taken when a bundle that is "deliverable" subject to that registration is received at a time when the registration is in the Passive state.

Destination - The destination of a bundle is the endpoint comprising the node(s) at which the bundle is to be delivered (as defined above).

Transmission - A transmission is an attempt by a node's BPA to cause copies of a bundle to be delivered to one or more of the nodes that are members of some endpoint (the bundle's destination) in response to a transmission request issued by the node's application agent.

Forwarding - To forward a bundle to a node is to invoke the services of one or more CLAs in a sustained effort to cause a copy of the bundle to be received by that node.

Discarding - To discard a bundle is to cease all operations on the bundle and functionally erase all references to it. The specific procedures by which this is accomplished are an implementation matter.

Retention constraint - A retention constraint is an element of the state of a bundle that prevents the bundle from being discarded. That is, a bundle cannot be discarded while it has any retention constraints.

Deletion - To delete a bundle is to remove unconditionally all of the bundle's retention constraints, enabling the bundle to be discarded.

3.2. Discussion of BP concepts

Multiple instances of the same bundle (the same unit of DTN protocol data) might exist concurrently in different parts of a network -- possibly differing in some blocks -- in the memory local to one or more bundle nodes and/or in transit between nodes. In the context of the operation of a bundle node, a bundle is an instance (copy), in that node's local memory, of some bundle that is in the network.

The payload for a bundle forwarded in response to a bundle transmission request is the application data unit whose location is provided as a parameter to that request. The payload for a bundle forwarded in response to reception of a bundle is the payload of the received bundle.

In the most familiar case, a bundle node is instantiated as a single process running on a general-purpose computer, but in general the definition is meant to be broader: a bundle node might alternatively be a thread, an object in an object-oriented operating system, a special-purpose hardware device, etc.

The manner in which the functions of the BPA are performed is wholly an implementation matter. For example, BPA functionality might be coded into each node individually; it might be implemented as a shared library that is used in common by any number of bundle nodes on a single computer; it might be implemented as a daemon whose services are invoked via inter-process or network communication by any number of bundle nodes on one or more computers; it might be implemented in hardware.

Every CLA implements its own thin layer of protocol, interposed between BP and the (usually "top") protocol(s) of the underlying native protocol stack; this "CL protocol" may only serve to multiplex and de-multiplex bundles to and from the underlying native protocol, or it may offer additional CL-specific functionality. The manner in which a CLA sends and receives bundles, as well as the definitions of CLAs and CL protocols, are beyond the scope of this specification.

Note that the administrative element of a node's application agent may itself, in some cases, function as a convergence-layer adapter. That is, outgoing bundles may be "tunneled" through encapsulating bundles:

- . An outgoing bundle constitutes a byte array. This byte array may, like any other, be presented to the bundle protocol agent as an application data unit that is to be transmitted to some endpoint.
- . The original bundle thus forms the payload of an encapsulating bundle that is forwarded using some other convergence-layer protocol(s).
- . When the encapsulating bundle is received, its payload is delivered to the peer application agent administrative element, which then instructs the bundle protocol agent to dispatch that original bundle in the usual way.

The purposes for which this technique may be useful (such as cross-domain security) are beyond the scope of this specification.

The only interface between the BPA and the application-specific element of the AA is the BP service interface. But between the BPA and the administrative element of the AA there is a (conceptual)

private control interface in addition to the BP service interface. This private control interface enables the BPA and the administrative element of the AA to direct each other to take action under specific circumstances.

In the case of a node that serves simply as a BP "router", the AA may have no application-specific element at all. The application-specific elements of other nodes' AAs may perform arbitrarily complex application functions, perhaps even offering multiplexed DTN communication services to a number of other applications. As with the BPA, the manner in which the AA performs its functions is wholly an implementation matter.

Singletons are the most familiar sort of endpoint, but in general the endpoint notion is meant to be broader. For example, the nodes in a sensor network might constitute a set of bundle nodes that are all registered in a single common endpoint and will all receive any data delivered at that endpoint. *Note* too that any given bundle node might be registered in multiple bundle endpoints and receive all data delivered at each of those endpoints.

Recall that every node, by definition, includes an application agent which in turn includes an administrative element, which exchanges administrative records with the administrative elements of other nodes. As such, every node is permanently, structurally registered in the singleton endpoint at which administrative records received from other nodes are delivered. Registration in no other endpoint can ever be assumed to be permanent. This endpoint, termed the node's "administrative endpoint", is therefore uniquely and permanently associated with the node, and for this reason the ID of a node's administrative endpoint additionally serves as the "node ID" (see 4.1.5.2 below) of the node.

The destination of every bundle is an endpoint, which may or may not be singleton. The source of every bundle is a node, identified by node ID. Note, though, that the source node ID asserted in a given bundle may be the null endpoint ID (as described later) rather than the ID of the source node; bundles for which the asserted source node ID is the null endpoint ID are termed "anonymous" bundles.

Any number of transmissions may be concurrently undertaken by the bundle protocol agent of a given node.

When the bundle protocol agent of a node determines that a bundle must be forwarded to a node (either to a node that is a member of the bundle's destination endpoint or to some intermediate forwarding node) in the course of completing the successful transmission of

that bundle, the bundle protocol agent invokes the services of one or more CLAs in a sustained effort to cause a copy of the bundle to be received by that node.

Upon reception, the processing of a bundle that has been received by a given node depends on whether or not the receiving node is registered in the bundle's destination endpoint. If it is, and if the payload of the bundle is non-fragmentary (possibly as a result of successful payload reassembly from fragmentary payloads, including the original payload of the newly received bundle), then the bundle is normally delivered to the node's application agent subject to the registration characterizing the node's membership in the destination endpoint.

The bundle protocol does not natively ensure delivery of a bundle to its destination. Data loss along the path to the destination node can be minimized by utilizing reliable convergence-layer protocols between neighbors on all segments of the end-to-end path, but for end-to-end bundle delivery assurance it will be necessary to develop extensions to the bundle protocol and/or application-layer mechanisms.

The bundle protocol is designed for extensibility. Bundle protocol extensions, documented elsewhere, may extend this specification by:

- . defining additional blocks;
- . defining additional administrative records;
- . defining additional bundle processing flags;
- . defining additional block processing flags;
- . defining additional types of bundle status reports;
- . defining additional bundle status report reason codes;
- . defining additional mandates and constraints on processing that conformant bundle protocol agents must perform at specified points in the inbound and outbound bundle processing cycles.

3.3. Services Offered by Bundle Protocol Agents

The BPA of each node is expected to provide the following services to the node's application agent:

- . commencing a registration (registering the node in an endpoint);
- . terminating a registration;
- . switching a registration between Active and Passive states;
- . transmitting a bundle to an identified bundle endpoint;
- . canceling a transmission;

- . polling a registration that is in the Passive state;
- . delivering a received bundle.

Note that the details of registration functionality are an implementation matter and are beyond the scope of this specification.

4. Bundle Format

The format of bundles SHALL conform to the Concise Binary Object Representation (CBOR [RFC7049]).

Cryptographic verification of a block is possible only if the sequence of octets on which the verifying node computes its hash – the canonicalized representation of the block – is identical to the sequence of octets on which the hash declared for that block was computed. To ensure that blocks are always in canonical representation when they are transmitted and received, the CBOR representation of every integer in every block SHALL be of minimum length; that is, no CBOR integer representation may contain any leading octet whose value is 0x00.

Each bundle SHALL be a concatenated sequence of at least two blocks, represented as a CBOR indefinite-length array. The first block in the sequence (the first item of the array) MUST be a primary bundle block in CBOR representation as described below; the bundle MUST have exactly one primary bundle block. The primary block MUST be followed by one or more canonical bundle blocks (additional array items) in CBOR representation as described in 4.2.3 below. The last such block MUST be a payload block; the bundle MUST have exactly one payload block. The payload block SHALL be followed by a CBOR "break" stop code, terminating the array.

(Note that, while CBOR permits considerable flexibility in the encoding of bundles, this flexibility must not be interpreted as inviting increased complexity in protocol data unit structure.)

An implementation of the Bundle Protocol MAY discard any sequence of bytes that does not conform to the Bundle Protocol specification.

An implementation of the Bundle Protocol MAY accept a sequence of bytes that does not conform to the Bundle Protocol specification (e.g., one that represents data elements in fixed-length arrays rather than indefinite-length arrays) and transform it into conformant BP structure before processing it. Procedures for accomplishing such a transformation are beyond the scope of this specification.

4.1. BP Fundamental Data Structures

4.1.1. CRC Type

CRC type is an unsigned integer type code for which the following values (and no others) are valid:

- . 0 indicates "no CRC is present."
- . 1 indicates "a standard X-25 CRC-16 is present." [CRC16]
- . 2 indicates "a standard CRC32C (Castagnoli) CRC-32 is present." [RFC4960]

CRC type SHALL be represented as a CBOR unsigned integer.

For examples of CRC32C CRCs, see Appendix A.4 of [RFC7143].

Note that more robust protection of BP data integrity, as needed, may be provided by means of Block Integrity Blocks as defined in the Bundle Security Protocol [BPSEC]).

4.1.2. CRC

CRC SHALL be omitted from a block if and only if the block's CRC type code is zero.

When not omitted, the CRC SHALL be represented as a CBOR byte string of two bytes (that is, CBOR additional information 2, if CRC type is 1) or of four bytes (that is, CBOR additional information 4, if CRC type is 2); in each case the sequence of bytes SHALL constitute an unsigned integer value (of 16 or 32 bits, respectively) in network byte order.

4.1.3. Bundle Processing Control Flags

Bundle processing control flags assert properties of the bundle as a whole rather than of any particular block of the bundle. They are conveyed in the primary block of the bundle.

The following properties are asserted by the bundle processing control flags:

- . The bundle is a fragment. (Boolean)
- . The bundle's payload is an administrative record. (Boolean)
- . The bundle must not be fragmented. (Boolean)

- . Acknowledgment by the user application is requested. (Boolean)
- . Status time is requested in all status reports. (Boolean)
- . Flags requesting types of status reports (all Boolean):
 - o Request reporting of bundle reception.
 - o Request reporting of bundle forwarding.
 - o Request reporting of bundle delivery.
 - o Request reporting of bundle deletion.

If the bundle processing control flags indicate that the bundle's application data unit is an administrative record, then all status report request flag values MUST be zero.

If the bundle's source node is omitted (i.e., the source node ID is the ID of the null endpoint, which has no members as discussed below; this option enables anonymous bundle transmission), then the bundle is not uniquely identifiable and all bundle protocol features that rely on bundle identity must therefore be disabled: the "Bundle must not be fragmented" flag value MUST be 1 and all status report request flag values MUST be zero.

Bundle processing control flags that are unrecognized MUST be ignored, as future definitions of additional flags might not be integrated simultaneously into the Bundle Protocol implementations operating at all nodes.

The bundle processing control flags SHALL be represented as a CBOR unsigned integer item, the value of which SHALL be processed as a bit field indicating the control flag values as follows (note that bit numbering in this instance is reversed from the usual practice, beginning with the low-order bit instead of the high-order bit, in recognition of the potential definition of additional control flag values in the future):

- . Bit 0 (the low-order bit, 0x0000001): bundle is a fragment.
- . Bit 1 (0x0000002): payload is an administrative record.
- . Bit 2 (0x0000004): bundle must not be fragmented.
- . Bit 3 (0x0000008): reserved.
- . Bit 4 (0x0000010): reserved.
- . Bit 5 (0x0000020): user application acknowledgement is requested.

- . Bit 6 (0x000040): status time is requested in all status reports.
- . Bit 7 (0x000080): reserved.
- . Bit 8 (0x000100): reserved.
- . Bit 9 (0x000200): reserved.
- . Bit 10 (0x000400): reserved.
- . Bit 11 (0x000800): reserved.
- . Bit 12 (0x001000): reserved.
- . Bit 13 (0x002000): reserved.
- . Bit 14 (0x004000): bundle reception status reports are requested.
- . Bit 15 (0x008000): reserved.
- . Bit 16 (0x010000): bundle forwarding status reports are requested.
- . Bit 17 (0x020000): bundle delivery status reports are requested.
- . Bit 18 (0x040000): bundle deletion status reports are requested.
- . Bits 19-20 are reserved.
- . Bits 21-63 are unassigned.

4.1.4. Block Processing Control Flags

The block processing control flags assert properties of canonical bundle blocks. They are conveyed in the header of the block to which they pertain.

Block processing control flags that are unrecognized MUST be ignored, as future definitions of additional flags might not be integrated simultaneously into the Bundle Protocol implementations operating at all nodes.

The block processing control flags SHALL be represented as a CBOR unsigned integer item, the value of which SHALL be processed as a bit field indicating the control flag values as follows (note that bit numbering in this instance is reversed from the usual practice, beginning with the low-order bit instead of the high-order bit, for agreement with the bit numbering of the bundle processing control flags):

- . Bit 0 (the low-order bit, 0x01): block must be replicated in every fragment.
- . Bit 1 (0x02): transmission of a status report is requested if block can't be processed.
- . Bit 2 (0x04): bundle must be deleted if block can't be processed.
- . Bit 3 (0x08): reserved.
- . Bit 4 (0x10): block must be removed from bundle if it can't be processed.

- . Bit 5(0x20): reserved.
- . Bit 6 (0x40): reserved.
- . Bits 7-63 are unassigned.

For each bundle whose bundle processing control flags indicate that the bundle's application data unit is an administrative record, or whose source node ID is the null endpoint ID as defined below, the value of the "Transmit status report if block can't be processed" flag in every canonical block of the bundle MUST be zero.

4.1.5. Identifiers

4.1.5.1. Endpoint ID

The destinations of bundles are bundle endpoints, identified by text strings termed "endpoint IDs" (see Section 3.1). Each endpoint ID (EID) is a Uniform Resource Identifier (URI; [URI]). As such, each endpoint ID can be characterized as having this general structure:

< scheme name > : < scheme-specific part, or "SSP" >

The scheme identified by the < scheme name > in an endpoint ID is a set of syntactic and semantic rules that fully explain how to parse and interpret the SSP. Each scheme that may be used to form a BP endpoint ID must be added to the registry of URI scheme code numbers for Bundle Protocol maintained by IANA as described in Section 10; association of a unique URI scheme code number with each scheme name in this registry helps to enable compact representation of endpoint IDs in bundle blocks. Note that the set of allowable schemes is effectively unlimited. Any scheme conforming to [URIREG] may be added to the URI scheme code number registry and thereupon used in a bundle protocol endpoint ID.

Each entry in the URI scheme code number registry MUST contain a reference to a scheme code number definition document, which defines the manner in which the scheme-specific part of any URI formed in that scheme is parsed and interpreted and MUST be encoded, in CBOR representation, for transmission as a BP endpoint ID. The scheme code number definition document may also contain information as to (a) which convergence-layer protocol(s) may be used to forward a bundle to a BP destination endpoint identified by such an ID, and (b) how the ID of the convergence-layer protocol endpoint to use for that purpose can be inferred from that destination endpoint ID.

Note that, although endpoint IDs are URIs, implementations of the BP service interface may support expression of endpoint IDs in some

internationalized manner (e.g., Internationalized Resource Identifiers (IRIs); see [RFC3987]).

Each BP endpoint ID (EID) SHALL be represented as a CBOR array comprising two items.

The first item of the array SHALL be the code number identifying the endpoint ID's URI scheme, as defined in the registry of URI scheme code numbers for Bundle Protocol. Each URI scheme code number SHALL be represented as a CBOR unsigned integer.

The second item of the array SHALL be the applicable CBOR representation of the scheme-specific part (SSP) of the EID, defined as noted in the references(s) for the URI scheme code number registry entry for the EID's URI scheme.

4.1.5.1.1. The "dtn" URI scheme

The "dtn" scheme supports the identification of BP endpoints by arbitrarily expressive character strings. It is specified as follows:

Scheme syntax: This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

dtn-uri = "dtn:" ("none" / dtn-hier-part)

dtn-hier-part = "//" node-name name-delim demux ; a path-rootless

node-name = 1*(ALPHA/DIGIT/"-"/"."/"_") reg-name

name-delim = "/"

demux = *VCHAR

Scheme semantics: URIs of the dtn scheme are used as endpoint identifiers in the Delay-Tolerant Networking (DTN) Bundle Protocol (BP) as described in the present document.

The endpoint ID "dtn:none" identifies the "null endpoint", the endpoint that by definition never has any members.

All BP endpoints identified by all other dtn-scheme endpoint IDs for which the first character of demux is a character other than '~' (tilde) are singleton endpoints. All BP endpoints identified by dtn-scheme endpoint IDs for which the first character *is* '~' (tilde) are *not* singleton endpoints.

A dtn-scheme endpoint ID for which the demux is of length zero MAY identify the administrative endpoint for the node identified by node-name, and as such may serve as a node ID. No dtn-scheme endpoint ID for which the demux is of non-zero length may do so.

Encoding considerations: For transmission as a BP endpoint ID, the scheme-specific part of a URI of the dtn scheme SHALL be represented as a CBOR text string unless the EID's SSP is "none", in which case the SSP SHALL be represented as a CBOR unsigned integer with the value zero. For all other purposes, URIs of the dtn scheme are encoded exclusively in US-ASCII characters.

Interoperability considerations: none.

Security considerations:

- . Reliability and consistency: none of the BP endpoints identified by the URIs of the DTN scheme are guaranteed to be reachable at any time, and the identity of the processing entities operating on those endpoints is never guaranteed by the Bundle Protocol itself. Bundle authentication as defined by the Bundle Security Protocol is required for this purpose.
- . Malicious construction: malicious construction of a conformant DTN-scheme URI is limited to the malicious selection of node names and the malicious selection of demux strings. That is, a maliciously constructed DTN-scheme URI could be used to direct a bundle to an endpoint that might be damaged by the arrival of that bundle or, alternatively, to declare a false source for a bundle and thereby cause incorrect processing at a node that receives the bundle. In both cases (and indeed in all bundle processing), the node that receives a bundle should verify its authenticity and validity before operating on it in any way.
- . Back-end transcoding: the limited expressiveness of URIs of the DTN scheme effectively eliminates the possibility of threat due to errors in back-end transcoding.
- . Rare IP address formats: not relevant, as IP addresses do not appear anywhere in conformant DTN-scheme URIs.
- . Sensitive information: because DTN-scheme URIs are used only to represent the identities of Bundle Protocol endpoints, the risk of disclosure of sensitive information due to interception of these URIs is minimal. Examination of DTN-scheme URIs could be used to support traffic analysis; where traffic analysis is a plausible danger, bundles should be conveyed by secure convergence-layer protocols that do not expose endpoint IDs.
- . Semantic attacks: the simplicity of DTN-scheme URI syntax minimizes the possibility of misinterpretation of a URI by a human user.

4.1.5.1.2. The "ipn" URI scheme

The "ipn" scheme supports the identification of BP endpoints by pairs of unsigned integers, for compact representation in bundle blocks. It is specified as follows:

Scheme syntax: This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the core ABNF syntax rule for DIGIT defined by that specification.

ipn-uri = "ipn:" ipn-hier-part

ipn-hier-part = node-nbr nbr-delim service-nbr ; a path-rootless

node-nbr = 1*DIGIT

nbr-delim = "."

service-nbr = 1*DIGIT

Scheme semantics: URIs of the ipn scheme are used as endpoint identifiers in the Delay-Tolerant Networking (DTN) Bundle Protocol (BP) as described in the present document.

All BP endpoints identified by ipn-scheme endpoint IDs are singleton endpoints.

An ipn-scheme endpoint ID for which service-nbr is zero MAY identify the administrative endpoint for the node identified by node-nbr, and as such may serve as a node ID. No ipn-scheme endpoint ID for which service-nbr is non-zero may do so.

Encoding considerations: For transmission as a BP endpoint ID, the scheme-specific part of a URI of the dtn scheme the SSP SHALL be represented as a CBOR array comprising two items. The first item of this array SHALL be the EID's node number (a number that identifies the node) represented as a CBOR unsigned integer. The second item of this array SHALL be the EID's service number (a number that identifies some application service) represented as a CBOR unsigned integer. For all other purposes, URIs of the IPN scheme are encoded exclusively in US-ASCII characters.

Interoperability considerations: none.

Security considerations:

- . Reliability and consistency: none of the BP endpoints identified by the URIs of the IPN scheme are guaranteed to be reachable at any time, and the identity of the processing entities operating on those endpoints is never guaranteed by the Bundle Protocol itself. Bundle authentication as defined by the Bundle Security Protocol [BPSEC] is required for this purpose.
- . Malicious construction: malicious construction of a conformant IPN-scheme URI is limited to the malicious selection of node numbers and the malicious selection of service numbers. That is, a maliciously constructed IPN-scheme URI could be used to direct a bundle to an endpoint that might be damaged by the arrival of that bundle or, alternatively, to declare a false source for a bundle and thereby cause incorrect processing at a node that receives the bundle. In both cases (and indeed in all bundle processing), the node that receives a bundle should verify its authenticity and validity before operating on it in any way.
- . Back-end transcoding: the limited expressiveness of URIs of the IPN scheme effectively eliminates the possibility of threat due to errors in back-end transcoding.
- . Rare IP address formats: not relevant, as IP addresses do not appear anywhere in conformant IPN-scheme URIs.
- . Sensitive information: because IPN-scheme URIs are used only to represent the identities of Bundle Protocol endpoints, the risk of disclosure of sensitive information due to interception of these URIs is minimal. Examination of IPN-scheme URIs could be used to support traffic analysis; where traffic analysis is a plausible danger, bundles should be conveyed by secure convergence-layer protocols that do not expose endpoint IDs.
- . Semantic attacks: the simplicity of IPN-scheme URI syntax minimizes the possibility of misinterpretation of a URI by a human user.

4.1.5.2. Node ID

For many purposes of the Bundle Protocol it is important to identify the node that is operative in some context.

As discussed in 3.1 above, nodes are distinct from endpoints; specifically, an endpoint is a set of zero or more nodes. But rather than define a separate namespace for node identifiers, we instead use endpoint identifiers to identify nodes as discussed in 3.2 above. Formally:

- . Every node is, by definition, permanently registered in the singleton endpoint at which administrative records are

- delivered to its application agent's administrative element, termed the node's "administrative endpoint".
- . As such, the EID of a node's administrative endpoint SHALL uniquely identify that node.
- . A "node ID" is an EID that identifies the administrative endpoint of a node.

4.1.6. DTN Time

A DTN time is an unsigned integer indicating the number of milliseconds that have elapsed since the start of the year 2000 on the Coordinated Universal Time (UTC) scale [UTC]. Each DTN time SHALL be represented as a CBOR unsigned integer item.

Implementers need to be aware that DTN time values conveyed in CBOR representation in bundles will nearly always exceed $(2^{32} - 1)$.

4.1.7. Creation Timestamp

Each bundle's creation timestamp SHALL be represented as a CBOR array comprising two items.

The first item of the array, termed "bundle creation time", SHALL be the DTN time at which the transmission request was received that resulted in the creation of the bundle, represented as a CBOR unsigned integer.

The second item of the array, termed the creation timestamp's "sequence number", SHALL be the latest value (as of the time at which the transmission request was received) of a monotonically increasing positive integer counter managed by the source node's bundle protocol agent, represented as a CBOR unsigned integer. The sequence counter MAY be reset to zero whenever the current time advances by one millisecond.

For nodes that lack accurate clocks, it is recommended that bundle creation time be set to zero and that the counter used as the source of the bundle sequence count never be reset to zero.

Note that, in general, the creation of two distinct bundles with the same source node ID and bundle creation timestamp may result in unexpected network behavior and/or suboptimal performance. The combination of source node ID and bundle creation timestamp serves to identify a single transmission request, enabling it to be acknowledged by the receiving application (provided the source node ID is not the null endpoint ID).

4.1.8. Block-type-specific Data

Block-type-specific data in each block (other than the primary block) SHALL be the applicable CBOR representation of the content of the block. Details of this representation are included in the specification defining the block type.

4.2. Bundle Representation

This section describes the primary block in detail and non-primary blocks in general. Rules for processing these blocks appear in Section 5 of this document.

Note that supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may require that BP implementations conforming to those protocols construct and process additional blocks.

4.2.1. Bundle

Each bundle SHALL be represented as a CBOR indefinite-length array. The first item of this array SHALL be the CBOR representation of a Primary Block. Every other item of the array SHALL be the CBOR representation of a Canonical Block. The last block of the bundle SHALL be followed by a CBOR "break" stop code, terminating the array.

Associated with each block of a bundle is a block number. The block number uniquely identifies the block within the bundle, enabling blocks (notably bundle security protocol blocks) to reference other blocks in the same bundle without ambiguity. The block number of the primary block is implicitly zero; the block numbers of all other blocks are explicitly stated in block headers as noted below. Block numbering is unrelated to the order in which blocks are sequenced in the bundle. The block number of the payload block is always 1.

4.2.2. Primary Bundle Block

The primary bundle block contains the basic information needed to forward bundles to their destinations.

Each primary block SHALL be represented as a CBOR array; the number of elements in the array SHALL be 8 (if the bundle is not a fragment and the block has no CRC), 9 (if the block has a CRC and the bundle is not a fragment), 10 (if the bundle is a fragment and the block has no CRC), or 11 (if the bundle is a fragment and the block has a CRC).

The primary block of each bundle SHALL be immutable. The CBOR-encoded values of all fields in the primary block MUST remain unchanged from the time the block is created to the time it is delivered.

The fields of the primary bundle block SHALL be as follows, listed in the order in which they MUST appear:

Version: An unsigned integer value indicating the version of the bundle protocol that constructed this block. The present document describes version 7 of the bundle protocol. Version number SHALL be represented as a CBOR unsigned integer item.

Bundle Processing Control Flags: The Bundle Processing Control Flags are discussed in Section 4.1.3. above.

CRC Type: CRC Type codes are discussed in Section 4.1.1. above. The CRC Type code for the primary block MAY be zero if the bundle contains a BPsec [BPSEC] Block Integrity Block whose target is the primary block; otherwise the CRC Type code for the primary block MUST be non-zero.

Destination EID: The Destination EID field identifies the bundle endpoint that is the bundle's destination, i.e., the endpoint that contains the node(s) at which the bundle is to be delivered.

Source node ID: The Source node ID field identifies the bundle node at which the bundle was initially transmitted, except that Source node ID may be the null endpoint ID in the event that the bundle's source chooses to remain anonymous.

Report-to EID: The Report-to EID field identifies the bundle endpoint to which status reports pertaining to the forwarding and delivery of this bundle are to be transmitted.

Creation Timestamp: The creation timestamp (discussed in 4.1.7 above) comprises two unsigned integers that, together with the source node ID and (if the bundle is a fragment) the fragment offset and payload length, serve to identify the bundle. The first of these integers is the bundle's creation time, while the second is the bundle's creation timestamp sequence number. Bundle creation time SHALL be the DTN time at which the transmission request was received that resulted in the creation of the bundle. Sequence count SHALL be the latest value (as of the time at which that transmission request was received) of a monotonically increasing positive integer counter managed by the source node's bundle protocol agent that MAY be reset to zero whenever the current time advances by one millisecond. For

nodes that lack accurate clocks, it is recommended that bundle creation time be set to zero and that the counter used as the source of the bundle sequence count never be reset to zero. Note that, in general, the creation of two distinct bundles with the same source node ID and bundle creation timestamp may result in unexpected network behavior and/or suboptimal performance. The combination of source node ID and bundle creation timestamp serves to identify a single transmission request, enabling it to be acknowledged by the receiving application (provided the source node ID is not the null endpoint ID).

Lifetime: The lifetime field is an unsigned integer that indicates the time at which the bundle's payload will no longer be useful, encoded as a number of milliseconds past the creation time. (For high-rate deployments with very brief disruptions, fine-grained expression of bundle lifetime may be useful.) When a bundle's age exceeds its lifetime, bundle nodes need no longer retain or forward the bundle; the bundle SHOULD be deleted from the network.

If the asserted lifetime for a received bundle is so lengthy that retention of the bundle until its expiration time might degrade operation of the node at which the bundle is received, or if the bundle protocol agent of that node determines that the bundle must be deleted in order to prevent network performance degradation (e.g., the bundle appears to be part of a denial-of-service attack), then that bundle protocol agent MAY impose a temporary overriding lifetime of shorter duration; such overriding lifetime SHALL NOT replace the lifetime asserted in the bundle but SHALL serve as the bundle's effective lifetime while the bundle resides at that node. Procedures for imposing lifetime overrides are beyond the scope of this specification.

For bundles originating at nodes that lack accurate clocks, it is recommended that bundle age be obtained from the Bundle Age extension block (see 4.3.2 below) rather than from the difference between current time and bundle creation time. Bundle lifetime SHALL be represented as a CBOR unsigned integer item.

Fragment offset: If and only if the Bundle Processing Control Flags of this Primary block indicate that the bundle is a fragment, fragment offset SHALL be present in the primary block. Fragment offset SHALL be represented as a CBOR unsigned integer indicating the offset from the start of the original application data unit at which the bytes comprising the payload of this bundle were located.

Total Application Data Unit Length: If and only if the Bundle Processing Control Flags of this Primary block indicate that the

bundle is a fragment, total application data unit length SHALL be present in the primary block. Total application data unit length SHALL be represented as a CBOR unsigned integer indicating the total length of the original application data unit of which this bundle's payload is a part.

CRC: A CRC SHALL be present in the primary block unless the bundle includes a BPsec [BPSEC] Block Integrity Block whose target is the primary block, in which case a CRC MAY be present in the primary block. The length and nature of the CRC SHALL be as indicated by the CRC type. The CRC SHALL be computed over the concatenation of all bytes (including CBOR "break" characters) of the primary block including the CRC field itself, which for this purpose SHALL be temporarily populated with all bytes set to zero.

4.2.3. Canonical Bundle Block Format

Every block other than the primary block (all such blocks are termed "canonical" blocks) SHALL be represented as a CBOR array; the number of elements in the array SHALL be 5 (if CRC type is zero) or 6 (otherwise).

The fields of every canonical block SHALL be as follows, listed in the order in which they MUST appear:

- . Block type code, an unsigned integer. Bundle block type code 1 indicates that the block is a bundle payload block. Block type codes 2 through 9 are explicitly reserved as noted later in this specification. Block type codes 192 through 255 are not reserved and are available for private and/or experimental use. All other block type code values are reserved for future use.
- . Block number, an unsigned integer as discussed in 4.2.1 above. Block number SHALL be represented as a CBOR unsigned integer.
- . Block processing control flags as discussed in Section 4.1.4 above.
- . CRC type as discussed in Section 4.1.1 above.
- . Block-type-specific data represented as a single definite-length CBOR byte string, i.e., a CBOR byte string that is not of indefinite length. For each type of block, the block-type-specific data byte string is the serialization, in a block-type-specific manner, of the data conveyed by that type of block; definitions of blocks are required to define the manner in which block-type-specific data are serialized within the block-type-specific data field. For the Payload Block in particular (block type 1), the block-type-specific data field, termed the "payload", SHALL be an application data unit, or

some contiguous extent thereof, represented as a definite-length CBOR byte string.

- . If and only if the value of the CRC type field of this block is non-zero, a CRC. If present, the length and nature of the CRC SHALL be as indicated by the CRC type and the CRC SHALL be computed over the concatenation of all bytes of the block (including CBOR "break" characters) including the CRC field itself, which for this purpose SHALL be temporarily populated with all bytes set to zero.

4.3. Extension Blocks

"Extension blocks" are all blocks other than the primary and payload blocks. Because not all extension blocks are defined in the Bundle Protocol specification (the present document), not all nodes conforming to this specification will necessarily instantiate Bundle Protocol implementations that include procedures for processing (that is, recognizing, parsing, acting on, and/or producing) all extension blocks. It is therefore possible for a node to receive a bundle that includes extension blocks that the node cannot process. The values of the block processing control flags indicate the action to be taken by the bundle protocol agent when this is the case.

The following extension blocks are defined in the current document.

4.3.1. Previous Node

The Previous Node block, block type 6, identifies the node that forwarded this bundle to the local node (i.e., to the node at which the bundle currently resides); its block-type-specific data is the node ID of that forwarder node which SHALL take the form of a node ID represented as described in Section 4.1.5.2. above. If the local node is the source of the bundle, then the bundle MUST NOT contain any Previous Node block. Otherwise the bundle SHOULD contain one (1) occurrence of this type of block and MUST NOT contain more than one.

4.3.2. Bundle Age

The Bundle Age block, block type 7, contains the number of milliseconds that have elapsed between the time the bundle was created and time at which it was most recently forwarded. It is intended for use by nodes lacking access to an accurate clock, to aid in determining the time at which a bundle's lifetime expires. The block-type-specific data of this block is an unsigned integer containing the age of the bundle in milliseconds, which SHALL be represented as a CBOR unsigned integer item. (The age of the bundle

is the sum of all known intervals of the bundle's residence at forwarding nodes, up to the time at which the bundle was most recently forwarded, plus the summation of signal propagation time over all episodes of transmission between forwarding nodes. Determination of these values is an implementation matter.) If the bundle's creation time is zero, then the bundle MUST contain exactly one (1) occurrence of this type of block; otherwise, the bundle MAY contain at most one (1) occurrence of this type of block. A bundle MUST NOT contain multiple occurrences of the bundle age block, as this could result in processing anomalies.

4.3.3. Hop Count

The Hop Count block, block type 10, contains two unsigned integers, hop limit and hop count. A "hop" is here defined as an occasion on which a bundle was forwarded from one node to another node. Hop limit MUST be in the range 1 through 255. The hop limit value SHOULD NOT be changed at any time after creation of the Hop Count block; the hop count value SHOULD initially be zero and SHOULD be increased by 1 on each hop.

The hop count block is mainly intended as a safety mechanism, a means of identifying bundles for removal from the network that can never be delivered due to a persistent forwarding error. Hop count is particularly valuable as a defense against routing anomalies that might cause a bundle to be forwarded in a cyclical "ping-pong" fashion between two nodes. When a bundle's hop count exceeds its hop limit, the bundle SHOULD be deleted for the reason "hop limit exceeded", following the bundle deletion procedure defined in Section 5.10.

Procedures for determining the appropriate hop limit for a bundle are beyond the scope of this specification.

The block-type-specific data in a hop count block SHALL be represented as a CBOR array comprising two items. The first item of this array SHALL be the bundle's hop limit, represented as a CBOR unsigned integer. The second item of this array SHALL be the bundle's hop count, represented as a CBOR unsigned integer. A bundle MAY contain one occurrence of this type of block but MUST NOT contain more than one.

5. Bundle Processing

The bundle processing procedures mandated in this section and in Section 6 govern the operation of the Bundle Protocol Agent and the Application Agent administrative element of each bundle node. They

are neither exhaustive nor exclusive. Supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may augment, override, or supersede the mandates of this document.

5.1. Generation of Administrative Records

All transmission of bundles is in response to bundle transmission requests presented by nodes' application agents. When required to "generate" an administrative record (such as a bundle status report), the bundle protocol agent itself is responsible for causing a new bundle to be transmitted, conveying that record. In concept, the bundle protocol agent discharges this responsibility by directing the administrative element of the node's application agent to construct the record and request its transmission as detailed in Section 6 below. In practice, the manner in which administrative record generation is accomplished is an implementation matter, provided the constraints noted in Section 6 are observed.

Status reports are relatively small bundles. Moreover, even when the generation of status reports is enabled the decision on whether or not to generate a requested status report is left to the discretion of the bundle protocol agent. Nonetheless, note that requesting status reports for any single bundle might easily result in the generation of $(1 + (2 * (N-1)))$ status report bundles, where N is the number of nodes on the path from the bundle's source to its destination, inclusive. That is, the requesting of status reports for large numbers of bundles could result in an unacceptable increase in the bundle traffic in the network. For this reason, the generation of status reports MUST be disabled by default and enabled only when the risk of excessive network traffic is deemed acceptable. Mechanisms that could assist in assessing and mitigating this risk, such as pre-placed agreements authorizing the generation of status reports under specified circumstances, are beyond the scope of this specification.

Notes on administrative record terminology:

- . A "bundle reception status report" is a bundle status report with the "reporting node received bundle" flag set to 1.
- . A "bundle forwarding status report" is a bundle status report with the "reporting node forwarded the bundle" flag set to 1.
- . A "bundle delivery status report" is a bundle status report with the "reporting node delivered the bundle" flag set to 1.
- . A "bundle deletion status report" is a bundle status report with the "reporting node deleted the bundle" flag set to 1.

5.2. Bundle Transmission

The steps in processing a bundle transmission request are:

Step 1: Transmission of the bundle is initiated. An outbound bundle MUST be created per the parameters of the bundle transmission request, with the retention constraint "Dispatch pending". The source node ID of the bundle MUST be either the null endpoint ID, indicating that the source of the bundle is anonymous, or else the EID of a singleton endpoint whose only member is the node of which the BPA is a component.

Step 2: Processing proceeds from Step 1 of Section 5.4.

5.3. Bundle Dispatching

The steps in dispatching a bundle are:

Step 1: If the bundle's destination endpoint is an endpoint of which the node is a member, the bundle delivery procedure defined in Section 5.7 MUST be followed and for the purposes of all subsequent processing of this bundle at this node the node's membership in the bundle's destination endpoint SHALL be disavowed; specifically, even though the node is a member of the bundle's destination endpoint, the node SHALL NOT undertake to forward the bundle to itself in the course of performing the procedure described in Section 5.4.

Step 2: Processing proceeds from Step 1 of Section 5.4.

5.4. Bundle Forwarding

The steps in forwarding a bundle are:

Step 1: The retention constraint "Forward pending" MUST be added to the bundle, and the bundle's "Dispatch pending" retention constraint MUST be removed.

Step 2: The bundle protocol agent MUST determine whether or not forwarding is contraindicated (that is, rendered inadvisable) for any of the reasons listed in the IANA registry of Bundle Status Report Reason Codes (see section 10.5 below), whose initial contents are listed in Figure 4. In particular:

- . The bundle protocol agent MAY choose either to forward the bundle directly to its destination node(s) (if possible) or to forward the bundle to some other node(s) for further forwarding. The manner in which this decision is made may

depend on the scheme name in the destination endpoint ID and/or on other state but in any case is beyond the scope of this document; one possible mechanism is described in [SABR]. If the BPA elects to forward the bundle to some other node(s) for further forwarding but finds it impossible to select any node(s) to forward the bundle to, then forwarding is contraindicated.

- . Provided the bundle protocol agent succeeded in selecting the node(s) to forward the bundle to, the bundle protocol agent MUST subsequently select the convergence layer adapter(s) whose services will enable the node to send the bundle to those nodes. The manner in which specific appropriate convergence layer adapters are selected is beyond the scope of this document; the TCP convergence-layer adapter [TCPCL] MUST be implemented when some or all of the bundles forwarded by the bundle protocol agent must be forwarded via the Internet but may not be appropriate for the forwarding of any particular bundle. If the agent finds it impossible to select any appropriate convergence layer adapter(s) to use in forwarding this bundle, then forwarding is contraindicated.

Step 3: If forwarding of the bundle is determined to be contraindicated for any of the reasons listed in the IANA registry of Bundle Status Report Reason Codes (see section 10.5 below), then the Forwarding Contraindicated procedure defined in Section 5.4.1 MUST be followed; the remaining steps of Section 5.4 are skipped at this time.

Step 4: For each node selected for forwarding, the bundle protocol agent MUST invoke the services of the selected convergence layer adapter(s) in order to effect the sending of the bundle to that node. Determining the time at which the bundle protocol agent invokes convergence layer adapter services is a BPA implementation matter. Determining the time at which each convergence layer adapter subsequently responds to this service invocation by sending the bundle is a convergence-layer adapter implementation matter.

Note that:

- . If the bundle has a Previous Node block, as defined in 4.3.1 above, then that block MUST be removed from the bundle before the bundle is forwarded.
- . If the bundle protocol agent is configured to attach Previous Node blocks to forwarded bundles, then a Previous Node block containing the node ID of the forwarding node MUST be inserted into the bundle before the bundle is forwarded.
- . If the bundle has a bundle age block, as defined in 4.3.2. above, then at the last possible moment before the CLA

initiates conveyance of the bundle via the CL protocol the bundle age value MUST be increased by the difference between the current time and the time at which the bundle was received (or, if the local node is the source of the bundle, created).

Step 5: When all selected convergence layer adapters have informed the bundle protocol agent that they have concluded their data sending procedures with regard to this bundle, processing may depend on the results of those procedures.

If completion of the data sending procedures by all selected convergence layer adapters has not resulted in successful forwarding of the bundle (an implementation-specific determination that is beyond the scope of this specification), then the bundle protocol agent MAY choose (in an implementation-specific manner, again beyond the scope of this specification) to initiate another attempt to forward the bundle. In that event, processing proceeds from Step 4. The minimum number of times a given node will initiate another forwarding attempt for any single bundle in this event (a number which may be zero) is a node configuration parameter that must be exposed to other nodes in the network to the extent that this is required by the operating environment.

If completion of the data sending procedures by all selected convergence layer adapters HAS resulted in successful forwarding of the bundle, or if it has not but the bundle protocol agent does not choose to initiate another attempt to forward the bundle, then:

- . If the "request reporting of bundle forwarding" flag in the bundle's status report request field is set to 1, and status reporting is enabled, then a bundle forwarding status report SHOULD be generated, destined for the bundle's report-to endpoint ID. The reason code on this bundle forwarding status report MUST be "no additional information".
- . If any applicable bundle protocol extensions mandate generation of status reports upon conclusion of convergence-layer data sending procedures, all such status reports SHOULD be generated with extension-mandated reason codes.
- . The bundle's "Forward pending" retention constraint MUST be removed.

5.4.1. Forwarding Contraindicated

The steps in responding to contraindication of forwarding are:

Step 1: The bundle protocol agent MUST determine whether or not to declare failure in forwarding the bundle. Note: this decision is

likely to be influenced by the reason for which forwarding is contraindicated.

Step 2: If forwarding failure is declared, then the Forwarding Failed procedure defined in Section 5.4.2 MUST be followed.

Otherwise, when - at some future time - the forwarding of this bundle ceases to be contraindicated, processing proceeds from Step 4 of Section 5.4.

5.4.2. Forwarding Failed

The steps in responding to a declaration of forwarding failure are:

Step 1: The bundle protocol agent MAY forward the bundle back to the node that sent it, as identified by the Previous Node block, if present. This forwarding, if performed, SHALL be accomplished by performing Step 4 and Step 5 of section 5.4 where the sole node selected for forwarding SHALL be the node that sent the bundle.

Step 2: If the bundle's destination endpoint is an endpoint of which the node is a member, then the bundle's "Forward pending" retention constraint MUST be removed. Otherwise, the bundle MUST be deleted: the bundle deletion procedure defined in Section 5.10 MUST be followed, citing the reason for which forwarding was determined to be contraindicated.

5.5. Bundle Expiration

A bundle expires when the bundle's age exceeds its lifetime as specified in the primary bundle block or as overridden by the bundle protocol agent. Bundle age MAY be determined by subtracting the bundle's creation timestamp time from the current time if (a) that timestamp time is not zero and (b) the local node's clock is known to be accurate; otherwise bundle age MUST be obtained from the Bundle Age extension block. Bundle expiration MAY occur at any point in the processing of a bundle. When a bundle expires, the bundle protocol agent MUST delete the bundle for the reason "lifetime expired" (when the expired lifetime is the lifetime as specified in the primary block) or "traffic pared" (when the expired lifetime is a lifetime override as imposed by the bundle protocol agent): the bundle deletion procedure defined in Section 5.10 MUST be followed.

5.6. Bundle Reception

The steps in processing a bundle that has been received from another node are:

Step 1: The retention constraint "Dispatch pending" MUST be added to the bundle.

Step 2: If the "request reporting of bundle reception" flag in the bundle's status report request field is set to 1, and status reporting is enabled, then a bundle reception status report with reason code "No additional information" SHOULD be generated, destined for the bundle's report-to endpoint ID.

Step 3: CRCs SHOULD be computed for every block of the bundle that has an attached CRC. If any block of the bundle is malformed according to this specification (including syntactically invalid CBOR), or if any block has an attached CRC and the CRC computed for this block upon reception differs from that attached CRC, then the bundle protocol agent MUST delete the bundle for the reason "Block unintelligible". The bundle deletion procedure defined in Section 5.10 MUST be followed and all remaining steps of the bundle reception procedure MUST be skipped.

Step 4: For each block in the bundle that is an extension block that the bundle protocol agent cannot process:

- . If the block processing flags in that block indicate that a status report is requested in this event, and status reporting is enabled, then a bundle reception status report with reason code "Block unintelligible" SHOULD be generated, destined for the bundle's report-to endpoint ID.
- . If the block processing flags in that block indicate that the bundle must be deleted in this event, then the bundle protocol agent MUST delete the bundle for the reason "Block unintelligible"; the bundle deletion procedure defined in Section 5.10 MUST be followed and all remaining steps of the bundle reception procedure MUST be skipped.
- . If the block processing flags in that block do NOT indicate that the bundle must be deleted in this event but do indicate that the block must be discarded, then the bundle protocol agent MUST remove this block from the bundle.
- . If the block processing flags in that block indicate neither that the bundle must be deleted nor that the block must be discarded, then processing continues with the next extension block that the bundle protocol agent cannot process, if any; otherwise, processing proceeds from step 5.

Step 5: Processing proceeds from Step 1 of Section 5.3.

5.7. Local Bundle Delivery

The steps in processing a bundle that is destined for an endpoint of which this node is a member are:

Step 1: If the received bundle is a fragment, the application data unit reassembly procedure described in Section 5.9 MUST be followed. If this procedure results in reassembly of the entire original application data unit, processing of the fragmentary bundle whose payload has been replaced by the reassembled application data unit (whether this bundle or a previously received fragment) proceeds from Step 2; otherwise, the retention constraint "Reassembly pending" MUST be added to the bundle and all remaining steps of this procedure MUST be skipped.

Step 2: Delivery depends on the state of the registration whose endpoint ID matches that of the destination of the bundle:

- . An additional implementation-specific delivery deferral procedure MAY optionally be associated with the registration.
- . If the registration is in the Active state, then the bundle MUST be delivered automatically as soon as it is the next bundle that is due for delivery according to the BPA's bundle delivery scheduling policy, an implementation matter.
- . If the registration is in the Passive state, or if delivery of the bundle fails for some implementation-specific reason, then the registration's delivery failure action MUST be taken. Delivery failure action MUST be one of the following:
 - o defer delivery of the bundle subject to this registration until (a) this bundle is the least recently received of all bundles currently deliverable subject to this registration and (b) either the registration is polled or else the registration is in the Active state, and also perform any additional delivery deferral procedure associated with the registration; or
 - o abandon delivery of the bundle subject to this registration (as defined in 3.1.).

Step 3: As soon as the bundle has been delivered, if the "request reporting of bundle delivery" flag in the bundle's status report request field is set to 1 and bundle status reporting is enabled, then a bundle delivery status report SHOULD be generated, destined for the bundle's report-to endpoint ID. Note that this status report

only states that the payload has been delivered to the application agent, not that the application agent has processed that payload.

5.8. Bundle Fragmentation

It may at times be advantageous for bundle protocol agents to reduce the sizes of bundles in order to forward them. This might be the case, for example, if a node to which a bundle is to be forwarded is accessible only via intermittent contacts and no upcoming contact is long enough to enable the forwarding of the entire bundle.

The size of a bundle can be reduced by "fragmenting" the bundle. To fragment a bundle whose payload is of size M is to replace it with two "fragments" - new bundles with the same source node ID and creation timestamp as the original bundle - whose payloads are the first N and the last $(M - N)$ bytes of the original bundle's payload, where $0 < N < M$.

Note that fragments are bundles and therefore may themselves be fragmented, so multiple episodes of fragmentation may in effect replace the original bundle with more than two fragments. (However, there is only one 'level' of fragmentation, as in IP fragmentation.)

Any bundle whose primary block's bundle processing flags do NOT indicate that it must not be fragmented MAY be fragmented at any time, for any purpose, at the discretion of the bundle protocol agent. NOTE, however, that some combinations of bundle fragmentation, replication, and routing might result in unexpected traffic patterns.

Fragmentation SHALL be constrained as follows:

- . The concatenation of the payloads of all fragments produced by fragmentation MUST always be identical to the payload of the fragmented bundle (that is, the bundle that is being fragmented). Note that the payloads of fragments resulting from different fragmentation episodes, in different parts of the network, may be overlapping subsets of the fragmented bundle's payload.
- . The primary block of each fragment MUST differ from that of the fragmented bundle, in that the bundle processing flags of the fragment MUST indicate that the bundle is a fragment and both fragment offset and total application data unit length must be provided. Additionally, the CRC of the primary block of the fragmented bundle, if any, MUST be replaced in each fragment by a new CRC computed for the primary block of that fragment.

- . The payload blocks of fragments will differ from that of the fragmented bundle as noted above.
- . If the fragmented bundle is not a fragment or is the fragment with offset zero, then all extension blocks of the fragmented bundle MUST be replicated in the fragment whose offset is zero.
- . Each of the fragmented bundle's extension blocks whose "Block must be replicated in every fragment" flag is set to 1 MUST be replicated in every fragment.
- . Beyond these rules, rules for the replication of extension blocks in the fragments must be defined in the specifications for those extension block types.

5.9. Application Data Unit Reassembly

Note that the bundle fragmentation procedure described in 5.8 above may result in the replacement of a single original bundle with an arbitrarily large number of fragmentary bundles. In order to be delivered at a destination node, the original bundle's payload must be reassembled from the payloads of those fragments.

If the concatenation - as informed by fragment offsets and payload lengths - of the non-overlapping portions of the payloads of all previously received fragments with the same source node ID and creation timestamp as this fragment, together with the non-overlapping portion of the payload of this fragment, forms a continuous byte array whose length is equal to the total application data unit length in the fragment's primary block, then:

- . This byte array -- the reassembled application data unit -- MUST replace the payload of that fragment whose payload is a subset, starting at offset zero, of the reassembled application data unit. Note that this will enable delivery of the reconstituted original bundle as described in Step 1 of 5.7.
- . The "Reassembly pending" retention constraint MUST be removed from every previously received fragment whose payload is a subset of the reassembled application data unit.

Note: reassembly of application data units from fragments occurs at the nodes that are members of destination endpoints as necessary; an application data unit MAY also be reassembled at some other node on the path to the destination.

5.10. Bundle Deletion

The steps in deleting a bundle are:

Step 1: If the "request reporting of bundle deletion" flag in the bundle's status report request field is set to 1, and if status reporting is enabled, then a bundle deletion status report citing the reason for deletion SHOULD be generated, destined for the bundle's report-to endpoint ID.

Step 2: All of the bundle's retention constraints MUST be removed.

5.11. Discarding a Bundle

As soon as a bundle has no remaining retention constraints it MAY be discarded, thereby releasing any persistent storage that may have been allocated to it.

5.12. Canceling a Transmission

When requested to cancel a specified transmission, where the bundle created upon initiation of the indicated transmission has not yet been discarded, the bundle protocol agent MUST delete that bundle for the reason "transmission cancelled". For this purpose, the procedure defined in Section 5.10 MUST be followed.

6. Administrative Record Processing

6.1. Administrative Records

Administrative records are standard application data units that are used in providing some of the features of the Bundle Protocol. One type of administrative record has been defined to date: bundle status reports. Note that additional types of administrative records may be defined by supplementary DTN protocol specification documents.

Every administrative record consists of:

- . Record type code (an unsigned integer for which valid values are as defined below).
- . Record content in type-specific format.

Valid administrative record type codes are defined as follows:

+-----+-----+-----+-----+-----+-----+					
	Value		Meaning		
+=====+=====+=====+=====+=====+=====+					

	1	Bundle status report.	
+-----+-----+-----+-----+-----+-----+			
	(other)	Reserved for future use.	
+-----+-----+-----+-----+-----+-----+			

Figure 3: Administrative Record Type Codes

Each BP administrative record SHALL be represented as a CBOR array comprising two items.

The first item of the array SHALL be a record type code, which SHALL be represented as a CBOR unsigned integer.

The second element of this array SHALL be the applicable CBOR representation of the content of the record. Details of the CBOR representation of administrative record type 1 are provided below. Details of the CBOR representation of other types of administrative record type are included in the specifications defining those records.

6.1.1. Bundle Status Reports

The transmission of "bundle status reports" under specified conditions is an option that can be invoked when transmission of a bundle is requested. These reports are intended to provide information about how bundles are progressing through the system, including notices of receipt, forwarding, final delivery, and deletion. They are transmitted to the Report-to endpoints of bundles.

Each bundle status report SHALL be represented as a CBOR array. The number of elements in the array SHALL be either 6 (if the subject bundle is a fragment) or 4 (otherwise).

The first item of the bundle status report array SHALL be bundle status information represented as a CBOR array of at least 4 elements. The first four items of the bundle status information array shall provide information on the following four status assertions, in this order:

- . Reporting node received bundle.
- . Reporting node forwarded the bundle.
- . Reporting node delivered the bundle.
- . Reporting node deleted the bundle.

Each item of the bundle status information array SHALL be a bundle status item represented as a CBOR array; the number of elements in each such array SHALL be either 2 (if the value of the first item of this bundle status item is 1 AND the "Report status time" flag was set to 1 in the bundle processing flags of the bundle whose status is being reported) or 1 (otherwise). The first item of the bundle status item array SHALL be a status indicator, a Boolean value indicating whether or not the corresponding bundle status is asserted, represented as a CBOR Boolean value. The second item of the bundle status item array, if present, SHALL indicate the time (as reported by the local system clock, an implementation matter) at which the indicated status was asserted for this bundle, represented as a DTN time as described in Section 4.1.6. above.

The second item of the bundle status report array SHALL be the bundle status report reason code explaining the value of the status indicator, represented as a CBOR unsigned integer. Valid status report reason codes are registered in the IANA Bundle Status Report Reason Codes registry in the Bundle Protocol Namespace (see 10.5 below). The initial contents of that registry are listed in Figure 4 below but the list of status report reason codes provided here is neither exhaustive nor exclusive; supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may define additional reason codes.

+-----+-----+-----+		
+-----+-----+-----+		
Value	Meaning	
+=====+=====+=====+		
0	No additional information.	
+-----+-----+-----+		
1	Lifetime expired.	
+-----+-----+-----+		
2	Forwarded over unidirectional link.	
+-----+-----+-----+		
3	Transmission canceled.	
+-----+-----+-----+		

	4	Depleted storage.	
+-----+-----+-----+			
	5	Destination endpoint ID unavailable.	
+-----+-----+-----+			
	6	No known route to destination from here.	
+-----+-----+-----+			
	7	No timely contact with next node on route.	
+-----+-----+-----+			
	8	Block unintelligible.	
+-----+-----+-----+			
	9	Hop limit exceeded.	
+-----+-----+-----+			
	10	Traffic pared (e.g., status reports).	
+-----+-----+-----+			
	(other)	Reserved for future use.	
+-----+-----+-----+			

Figure 4: Status Report Reason Codes

The third item of the bundle status report array SHALL be the source node ID identifying the source of the bundle whose status is being reported, represented as described in Section 4.1.5.1.1. above.

The fourth item of the bundle status report array SHALL be the creation timestamp of the bundle whose status is being reported, represented as described in Section 4.1.7. above.

The fifth item of the bundle status report array SHALL be present if and only if the bundle whose status is being reported contained a fragment offset. If present, it SHALL be the subject bundle's fragment offset represented as a CBOR unsigned integer item.

The sixth item of the bundle status report array SHALL be present if and only if the bundle whose status is being reported contained a fragment offset. If present, it SHALL be the length of the subject bundle's payload represented as a CBOR unsigned integer item.

Note that the forwarding parameters (such as lifetime, applicable security measures, etc.) of the bundle whose status is being reported MAY be reflected in the parameters governing the forwarding of the bundle that conveys a status report, but this is an implementation matter. Bundle protocol deployment experience to date has not been sufficient to suggest any clear guidance on this topic.

6.2. Generation of Administrative Records

Whenever the application agent's administrative element is directed by the bundle protocol agent to generate an administrative record, the following procedure must be followed:

Step 1: The administrative record must be constructed. If the administrative record references a bundle and the referenced bundle is a fragment, the administrative record MUST contain the fragment offset and fragment length.

Step 2: A request for transmission of a bundle whose payload is this administrative record MUST be presented to the bundle protocol agent.

7. Services Required of the Convergence Layer

7.1. The Convergence Layer

The successful operation of the end-to-end bundle protocol depends on the operation of underlying protocols at what is termed the "convergence layer"; these protocols accomplish communication between nodes. A wide variety of protocols may serve this purpose, so long as each convergence layer protocol adapter provides a defined minimal set of services to the bundle protocol agent. This convergence layer service specification enumerates those services.

7.2. Summary of Convergence Layer Services

Each convergence layer protocol adapter is expected to provide the following services to the bundle protocol agent:

- . sending a bundle to a bundle node that is reachable via the convergence layer protocol;

- . notifying the bundle protocol agent of the disposition of its data sending procedures with regard to a bundle, upon concluding those procedures;
- . delivering to the bundle protocol agent a bundle that was sent by a bundle node via the convergence layer protocol.

The convergence layer service interface specified here is neither exhaustive nor exclusive. That is, supplementary DTN protocol specifications (including, but not restricted to, the Bundle Security Protocol [BPSEC]) may expect convergence layer adapters that serve BP implementations conforming to those protocols to provide additional services such as reporting on the transmission and/or reception progress of individual bundles (at completion and/or incrementally), retransmitting data that were lost in transit, discarding bundle-conveying data units that the convergence layer protocol determines are corrupt or inauthentic, or reporting on the integrity and/or authenticity of delivered bundles.

In addition, bundle protocol relies on the capabilities of protocols at the convergence layer to minimize congestion in the store-carry-forward overlay network. The potentially long round-trip times characterizing delay-tolerant networks are incompatible with end-to-end reactive congestion control mechanisms, so convergence-layer protocols MUST provide rate limiting or congestion control.

8. Implementation Status

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to RFC 7942.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented

protocols more mature. It is up to the individual working groups to use this information as they see fit".

At the time of this writing, there are six known implementations of the current document.

The first known implementation is microPCN (<https://upcn.eu/>). According to the developers:

The Micro Planetary Communication Network (uPCN) is a free software project intended to offer an implementation of Delay-tolerant Networking protocols for POSIX operating systems (well, and for Linux) plus for the ARM Cortex STM32F4 microcontroller series. More precisely it currently provides an implementation of

- . the Bundle Protocol (BP, RFC 5050),
- . version 6 of the Bundle Protocol version 7 specification draft,
- . the DTN IP Neighbor Discovery (IPND) protocol, and
- . a routing approach optimized for message-ferry micro LEO satellites.

uPCN is written in C and is built upon the real-time operating system FreeRTOS. The source code of uPCN is released under the "BSD 3-Clause License".

The project depends on an execution environment offering link layer protocols such as AX.25. The source code uses the USB subsystem to interact with the environment.

The second known implementation is PyDTN, developed by X-works, s.r.o (<https://x-works.sk/>). The final third of the implementation was developed during the IETF 101 Hackathon. According to the developers, PyDTN implements bundle coding/decoding and neighbor discovery. PyDTN is written in Python and has been shown to be interoperable with uPCN.

The third known implementation is "Terra" (<https://github.com/RightMesh/Terra/>), a Java implementation developed in the context of terrestrial DTN. It includes an implementation of a "minimal TCP" convergence layer adapter.

The fourth and fifth known implementations are products of cooperating groups at two German universities:

- . An implementation written in Go, licensed under GPLv3, is focused on being easily extensible suitable for research. It

is maintained at the University of Marburg and can be accessed from <https://github.com/dtn7/dtn7-go>.

- . An implementation written in Rust, licensed under the MIT/Apache license, is intended for environments with limited resources or demanding safety and/or performance requirements. It is maintained at the Technical University of Darmstadt and can be accessed at <https://github.com/dtn7/dtn7-rs/>.

The sixth known implementation is the "bpv7" module in version 4.0.0 of the Interplanetary Overlay Network (ION) software maintained at the Jet Propulsion Laboratory, California Institute of Technology, for the U.S. National Aeronautics and Space Administration (NASA).

9. Security Considerations

The bundle protocol security architecture and the available security services are specified in an accompanying document, the Bundle Security Protocol (BPsec) specification [BPSEC]. Whenever Bundle Protocol security services (as opposed to the security services provided by overlying application protocols or underlying convergence-layer protocols) are required, those services SHALL be provided by BPsec rather than by some other mechanism with the same or similar scope.

A Bundle Protocol Agent (BPA) which sources, cryptographically verifies, and/or accepts a bundle MUST implement support for BPsec. Use of BPsec for a particular Bundle Protocol session is optional.

The BPsec extensions to Bundle Protocol enable each block of a bundle (other than a BPsec extension block) to be individually authenticated by a signature block (Block Integrity Block, or BIB) and also enable each block of a bundle other than the primary block (and the BPsec extension blocks themselves) to be individually encrypted by a Block Confidentiality Block (BCB).

Because the security mechanisms are extension blocks that are themselves inserted into the bundle, the protections they afford apply while the bundle is at rest, awaiting transmission at the next forwarding opportunity, as well as in transit.

Additionally, convergence-layer protocols that ensure authenticity of communication between adjacent nodes in BP network topology SHOULD be used where available, to minimize the ability of unauthenticated nodes to introduce inauthentic traffic into the network. Convergence-layer protocols that ensure confidentiality of communication between adjacent nodes in BP network topology SHOULD also be used where available, to minimize exposure of the bundle's

primary block and other clear-text blocks, thereby offering some defense against traffic analysis.

In order to provide authenticity and/or confidentiality of communication between BP nodes, the convergence-layer protocol requires as input the name(s) of the expected communication peer(s). These must be supplied by the convergence-layer adapter. Details of the means by which the CLA determines which CL endpoint name(s) must be provided to the CL protocol are out of scope for this specification. Note, though, that when the CL endpoint names are a function of BP endpoint IDs, the correctness and authenticity of that mapping will be vital to the overall security properties that the CL provides to the system.

Note that, while the primary block must remain in the clear for routing purposes, the Bundle Protocol could be protected against traffic analysis to some extent by using bundle-in-bundle encapsulation [BIBE] to tunnel bundles to a safe forward distribution point: the encapsulated bundle could form the payload of an encapsulating bundle, and that payload block could be encrypted by a BCB.

Note that the generation of bundle status reports is disabled by default because malicious initiation of bundle status reporting could result in the transmission of extremely large numbers of bundles, effecting a denial of service attack. Imposing bundle lifetime overrides would constitute one defense against such an attack.

Note also that the reception of large numbers of fragmentary bundles with very long lifetimes could constitute a denial of service attack, occupying storage while pending reassembly that will never occur. Imposing bundle lifetime overrides would, again, constitute one defense against such an attack.

This protocol makes use of absolute timestamps for several purposes. Provisions are included for nodes without accurate clocks to retain most of the protocol functionality, but nodes that are unaware that their clock is inaccurate may exhibit unexpected behavior.

10. IANA Considerations

The Bundle Protocol includes fields requiring registries managed by IANA.

10.1. Bundle Block Types

The current Bundle Block Types registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle protocol (Bundle Protocol Version) that applies to the new values. IANA is requested to add the following values, as described in section 4.3.1, to the Bundle Block Types registry. The current values in the Bundle Block Types registry should have the Bundle Protocol Version set to the value "6", as shown below.

Bundle	Value	Description	Reference
Protocol			
Version			
none	0	Reserved	[RFC6255]
6,7	1	Bundle Payload Block	[RFC5050]
			RFC-to-be
6	2	Bundle Authentication Block	[RFC6257]
6	3	Payload Integrity Block	[RFC6257]
6	4	Payload Confidentiality	[RFC6257]
		Block	RFC-to-be
6	5	Previous-Hop Insertion Block	[RFC6259]
7	6	Previous node (proximate	RFC-to-be
		sender)	
7	7	Bundle age (in milliseconds)	RFC-to-be
6	8	Metadata Extension Block	[RFC6258]
6	9	Extension Security Block	[RFC6257]
7	10	Hop count (#prior xmit	RFC-to-be

		attempts)		
	7	11-191 Unassigned		
	6	192-255 Reserved for Private and/or	[RFC5050],	
		Experimental Use	RFC-to-be	
+-----+-----+-----+-----+				

10.2. Primary Bundle Protocol Version

IANA is requested to add the following value to the Primary Bundle Protocol Version registry in the Bundle Protocol Namespace.

+-----+-----+-----+			
Value	Description	Reference	
+-----+-----+-----+			
7	Assigned	RFC-to-be	
+-----+-----+-----+			

Values 8-255 (rather than 7-255) are now Unassigned.

10.3. Bundle Processing Control Flags

The current Bundle Processing Control Flags registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle protocol (Bundle Protocol Version) that applies to the new values. IANA is requested to add the following values, as described in section 4.1.3, to the Bundle Processing Control Flags registry. The current values in the Bundle Processing Control Flags registry should have the Bundle Protocol Version set to the value 6 or "6, 7", as shown below.

Bundle Processing Control Flags Registry

+-----+-----+-----+			
Bundle	Bit	Description	Reference
Protocol	Position		
Version	(right		

		to left)			
+-----+-----+-----+					
	6,7		0	Bundle is a fragment	[RFC5050],
					RFC-to-be
	6,7		1	Application data unit is an	[RFC5050],
				administrative record	RFC-to-be
	6,7		2	Bundle must not be fragmented	[RFC5050],
					RFC-to-be
	6		3	Custody transfer is requested	[RFC5050]
	6		4	Destination endpoint is singleton	[RFC5050]
	6,7		5	Acknowledgement by application	[RFC5050],
				is requested	RFC-to-be
	7		6	Status time requested in reports	RFC-to-be
	6		7	Class of service, priority	[RFC5050],
					RFC-to-be
	6		8	Class of service, priority	[RFC5050],
					RFC-to-be
	6		9	Class of service, reserved	[RFC5050],
					RFC-to-be
	6		10	Class of service, reserved	[RFC5050],
					RFC-to-be
	6		11	Class of service, reserved	[RFC5050],
					RFC-to-be
	6		12	Class of service, reserved	[RFC5050],

			RFC-to-be
6	13	Class of service, reserved	[RFC5050],
			RFC-to-be
6,7	14	Request reporting of bundle	[RFC5050],
		reception	RFC-to-be
6,7	16	Request reporting of bundle	[RFC5050],
		forwarding	RFC-to-be
6,7	17	Request reporting of bundle	[RFC5050],
		delivery	RFC-to-be
6,7	18	Request reporting of bundle	[RFC5050],
		deletion	RFC-to-be
6	19	Reserved	[RFC5050],
			RFC-to-be
6	20	Reserved	[RFC5050],
			RFC-to-be
	21-63	Unassigned	

+-----+-----+-----+

The registration policy for this registry is changed to "Specification Required" with expert verification of reasonable use of the code points. Given the limited number of bits available, the allocation should only be granted for a standards-track RFC approved by the IESG.

10.4. Block Processing Control Flags

The current Block Processing Control Flags registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle protocol (Bundle Protocol Version) that applies to the related BP version. The current values in the Block

Processing Control Flags registry should have the Bundle Protocol Version set to the value 6 or "6, 7", as shown below.

Block Processing Control Flags Registry

Bundle Protocol Version	Bit Position (right to left)	Description	Reference
6,7	0	Block must be replicated in every fragment	[RFC5050], RFC-to-be
6,7	1	Transmit status report if block can't be processed	[RFC5050], RFC-to-be
6,7	2	Delete bundle if block can't be processed	[RFC5050], RFC-to-be
6	3	Last block	[RFC5050]
6,7	4	Discard block if it can't be processed	[RFC5050], RFC-to-be
6	5	Block was forwarded without being processed	[RFC5050]
6	6	Block contains an EID reference field	[RFC5050]
	7-63	Unassigned	

The registration policy for this registry is changed to "Specification Required" with expert verification of reasonable use of the code points. Given the limited number of bits available, the allocation should only be granted for a standards-track RFC approved by the IESG.

10.5. Bundle Status Report Reason Codes

The current Bundle Status Report Reason Codes registry in the Bundle Protocol Namespace is augmented by adding a column identifying the version of the Bundle protocol (Bundle Protocol Version) that applies to the new values. IANA is requested to add the following values, as described in section 6.1.1, to the Bundle Status Report Reason Codes registry. The current values in the Bundle Status Report Reason Codes registry should have the Bundle Protocol Version set to the value 6 or 7 or "6, 7", as shown below.

Bundle Status Report Reason Codes Registry

+-----+-----+-----+			
Bundle	Value	Description	Reference
Protocol			
Version			
+-----+-----+-----+			
6,7	0	No additional information	[RFC5050],
			RFC-to-be
6,7	1	Lifetime expired	[RFC5050],
			RFC-to-be
6,7	2	Forwarded over unidirectional	[RFC5050],
		link	RFC-to-be
6,7	3	Transmission canceled	[RFC5050],
			RFC-to-be

	6,7		4		Depleted storage		[RFC5050],	
							RFC-to-be	
	6,7		5		Destination endpoint ID		[RFC5050],	
					unavailable		RFC-to-be	
	6,7		6		No known route to destination		[RFC5050],	
					from here		RFC-to-be	
	6,7		7		No timely contact with next node		[RFC5050],	
					on route		RFC-to-be	
	6,7		8		Block unintelligible		[RFC5050],	
							RFC-to-be	
	7		9		Hop limit exceeded		RFC-to-be	
	7		10		Traffic pared		RFC-to-be	
			11-254		Unassigned			
	6		255		Reserved		[RFC6255],	
							RFC-to-be	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								

10.6. Bundle Protocol URI scheme types

The Bundle Protocol has a URI scheme type field - an unsigned integer of indefinite length - for which IANA is requested to create and maintain a new "Bundle Protocol URI Scheme Type" registry in the Bundle Protocol Namespace. The "Bundle Protocol URI Scheme Type" registry governs an unsigned integer namespace. Initial values for the Bundle Protocol URI Scheme Type registry are given below.

The registration policy for this registry is: Standards Action. The allocation should only be granted for a standards-track RFC approved by the IESG.

The value range is: unsigned integer.

Each assignment consists of a URI scheme type name and its associated description, a reference to the document that defines the URI scheme, and a reference to the document that defines the use of this URI scheme in BP endpoint IDs (including the CBOR representation of those endpoint IDs in transmitted bundles).

Bundle Protocol URI Scheme Type Registry

		BP Utilization	URI Definition
Value	Description	Reference	Reference
0	Reserved	n/a	
1	dtn	RFC-to-be	RFC-to-be
2	ipn	RFC-to-be	[RFC6260],
			RFC-to-be
3-254	Unassigned	n/a	
255-65535	reserved	n/a	
>65535	open for	n/a	
	private use	n/a	

10.7. URI scheme "dtn"

In the Uniform Resource Identifier (URI) Schemes (uri-schemes) registry, IANA is requested to update the registration of the URI scheme with the string "dtn" as the scheme name, as follows:

URI scheme name: "dtn"

Status: permanent

Applications and/or protocols that use this URI scheme name: the Delay-Tolerant Networking (DTN) Bundle Protocol (BP).

Contact:

Scott Burleigh
Jet Propulsion Laboratory,
California Institute of Technology
scott.c.burleigh@jpl.nasa.gov
+1 (800) 393-3353

Change controller:

IETF, iesg@ietf.org

10.8. URI scheme "ipn"

In the Uniform Resource Identifier (URI) Schemes (uri-schemes) registry, IANA is requested to update the registration of the URI scheme with the string "ipn" as the scheme name, originally documented in RFC 6260 [RFC6260], as follows.

URI scheme name: "ipn"

Status: permanent

Applications and/or protocols that use this URI scheme name: the Delay-Tolerant Networking (DTN) Bundle Protocol (BP).

Contact:

Scott Burleigh
Jet Propulsion Laboratory,
California Institute of Technology
scott.c.burleigh@jpl.nasa.gov
+1 (800) 393-3353

Change controller:

IETF, iesg@ietf.org

11. References

11.1. Normative References

[BPSEC] Birrane, E., "Bundle Security Protocol Specification", draft-ietf-dtn-bpsec, January 2020.

[CRC16] ITU-T Recommendation X.25, p. 9, section 2.2.7.4, International Telecommunications Union, October 1996.

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

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC7049] Borman, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

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

[SABR] "Schedule-Aware Bundle Routing", CCSDS Recommended Standard 734.3-B-1, Consultative Committee for Space Data Systems, July 2019.

[TCPCL] Sipos, B., Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4", draft-ietf-dtn-tcpclv4, January 2020.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, STD 66, January 2005.

[URIREG] Thaler, D., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", RFC 7595, BCP 35, June 2015.

[UTC] Arias, E. and B. Guinot, "Coordinated universal time UTC: historical background and perspectives" in "Journées systemes de reference spatio-temporels", 2004.

11.2. Informative References

- [ARCH] V. Cerf et al., "Delay-Tolerant Network Architecture", RFC 4838, April 2007.
- [BIBE] Burleigh, S., "Bundle-in-Bundle Encapsulation", draft-ietf-dtn-bibect, August 2019.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.
- [RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, May 2011.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.
- [RFC6258] Symington, S., "Delay-Tolerant Networking Metadata Extension Block", RFC 6258, May 2011.
- [RFC6259] Symington, S., "Delay-Tolerant Networking Previous-Hop Insertion Block", RFC 6259, May 2011.
- [RFC6260] Burleigh, S., "Compressed Bundle Header Encoding (CBHE)", RFC 6260, May 2011.
- [RFC7143] Chadalapaka, M., Satran, J., Meth, K., and D. Black, "Internet Small Computer System Interface (iSCSI) Protocol (Consolidated)", RFC 7143, April 2014.
- [SIGC] Fall, K., "A Delay-Tolerant Network Architecture for Challenged Internets", SIGCOMM 2003.

12. Acknowledgments

This work is freely adapted from RFC 5050, which was an effort of the Delay Tolerant Networking Research Group. The following DTNRG participants contributed significant technical material and/or inputs to that document: Dr. Vinton Cerf of Google, Scott Burleigh, Adrian Hooke, and Leigh Torgerson of the Jet Propulsion Laboratory, Michael Demmer of the University of California at Berkeley, Robert Durst, Keith Scott, and Susan Symington of The MITRE Corporation, Kevin Fall of Carnegie Mellon University, Stephen Farrell of Trinity

College Dublin, Howard Weiss and Peter Lovell of SPARTA, Inc., and Manikantan Ramadas of Ohio University.

This document was prepared using 2-Word-v2.0.template.dot.

13. Significant Changes from RFC 5050

Points on which this draft significantly differs from RFC 5050 include the following:

- . Clarify the difference between transmission and forwarding.
- . Migrate custody transfer to the bundle-in-bundle encapsulation specification [BIBE].
- . Introduce the concept of "node ID" as functionally distinct from endpoint ID, while having the same syntax.
- . Restructure primary block, making it immutable. Add optional CRC.
- . Add optional CRCs to non-primary blocks.
- . Add block ID number to canonical block format (to support BPsec).
- . Add definition of bundle age extension block.
- . Add definition of previous node extension block.
- . Add definition of hop count extension block.
- . Remove Quality of Service markings.
- . Change from SDNVs to CBOR representation.
- . Add lifetime overrides.

Appendix A.

For More Information

Copyright (c) 2020 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 (<http://trustee.ietf.org/license-info>).

Appendix B. CDDL expression

For informational purposes, Carsten Bormann and Brian Sipos have kindly provided an expression of the Bundle Protocol specification in the Concise Data Definition Language (CDDL). That CDDL expression is presented below. Note that wherever the CDDL expression is in disagreement with the textual representation of the BP specification presented in the earlier sections of this document, the textual representation rules.

```
start = bundle / #6.55799(bundle)

; Times before 2000 are invalid

dtn-time = uint

; CRC enumerated type

crc-type = &(
    crc-none: 0,
    crc-16bit: 1,
    crc-32bit: 2
)

; Either 16-bit or 32-bit

crc-value = (bstr .size 2) / (bstr .size 4)

creation-timestamp = [
    dtn-time, ; absolute time of creation
    sequence: uint ; sequence within the time
]

eid = $eid .within eid-structure

eid-structure = [
    uri-code: uint,
```

```
    SSP: any
  ]
  $eid /= [
    uri-code: 1,
    SSP: (tstr / 0)
  ]
  $eid /= [
    uri-code: 2,
    SSP: [
      nodenum: uint,
      servicenum: uint
    ]
  ]
]
; The root bundle array
bundle = [primary-block, *extension-block, payload-block]
primary-block = [
  version: 7,
  bundle-control-flags,
  crc-type,
  destination: eid,
  source-node: eid,
  report-to: eid,
  creation-timestamp,
  lifetime: uint,
```

```
? (  
    fragment-offset: uint,  
    total-application-data-length: uint  
) ,  
? crc-value,  
]  
bundle-control-flags = uint .bits bundleflagbits  
bundleflagbits = &(  
    reserved: 21,  
    reserved: 20,  
    reserved: 19,  
    bundle-deletion-status-reports-are-requested: 18,  
    bundle-delivery-status-reports-are-requested: 17,  
    bundle-forwarding-status-reports-are-requested: 16,  
    reserved: 15,  
    bundle-reception-status-reports-are-requested: 14,  
    reserved: 13,  
    reserved: 12,  
    reserved: 11,  
    reserved: 10,  
    reserved: 9,  
    reserved: 8,  
    reserved: 7,
```

```
    status-time-is-requested-in-all-status-reports: 6,
    user-application-acknowledgement-is-requested: 5,
    reserved: 4,
    reserved: 3,
    bundle-must-not-be-fragmented: 2,
    payload-is-an-administrative-record: 1,
    bundle-is-a-fragment: 0
)
; Abstract shared structure of all non-primary blocks
canonical-block-structure = [
    block-type-code: uint,
    block-number: uint,
    block-control-flags,
    crc-type,
    ; Each block type defines the content within the bytestring
    block-type-specific-data,
    ? crc-value
]
block-control-flags = uint .bits blockflagbits
blockflagbits = &(amp;
    reserved: 7,
    reserved: 6,
    reserved: 5,
    block-must-be-removed-from-bundle-if-it-cannot-be-processed: 4,
```

```
    reserved: 3,

    bundle-must-be-deleted-if-block-cannot-be-processed: 2,

    status-report-must-be-transmitted-if-block-cannot-be-processed: 1,

    block-must-be-replicated-in-every-fragment: 0
)

block-type-specific-data = bstr / #6.24(bstr)

; Actual CBOR data embedded in a bytestring, with optional tag to
; indicate so

embedded-cbor<Item> = (bstr .cbor Item) / #6.24(bstr .cbor Item)

; Extension block type, which does not specialize other than the
; code/number

extension-block = $extension-block-structure .within canonical-
block-structure

; Generic shared structure of all non-primary blocks

extension-block-use<CodeValue, BlockData> = [

    block-type-code: CodeValue,

    block-number: (uint .gt 1),

    block-control-flags,

    crc-type,

    BlockData,

    ? crc-value

]

; Payload block type

payload-block = payload-block-structure .within canonical-block-
structure
```



```
payload-block-structure = [  
    block-type-code: 1,  
    block-number: 1,  
    block-control-flags,  
    crc-type,  
    $payload-block-data,  
    ? crc-value  
]  
  
; Arbitrary payload data, including non-CBOR bytestring  
$payload-block-data /= block-type-specific-data  
  
; Administrative record as a payload data specialization  
$payload-block-data /= embedded-cbor<admin-record>  
  
admin-record = $admin-record .within admin-record-structure  
  
admin-record-structure = [  
    record-type-code: uint,  
    record-content: any  
]  
  
; Only one defined record type  
$admin-record /= [1, status-record-content]  
  
status-record-content = [  
    bundle-status-information,  
    status-report-reason-code: uint,  
    source-node-eid: eid,  
    subject-creation-timestamp: creation-timestamp,
```

```
    ? (
      subject-payload-offset: uint,
      subject-payload-length: uint
    )
  ]
bundle-status-information = [
  reporting-node-received-bundle: status-info-content,
  reporting-node-forwarded-bundle: status-info-content,
  reporting-node-delivered-bundle: status-info-content,
  reporting-node-deleted-bundle: status-info-content
]
status-info-content = [
  status-indicator: bool,
  ? timestamp: dtn-time
]
; Previous Node extension block
$extension-block-structure /=
  extension-block-use<6, embedded-cbor<ext-data-previous-node>>
ext-data-previous-node = eid
; Bundle Age extension block
$extension-block-structure /=
  extension-block-use<7, embedded-cbor<ext-data-bundle-age>>
ext-data-bundle-age = uint
; Hop Count extension block
```

```
$extension-block-structure /=  
    extension-block-use<10, embedded-cbor<ext-data-hop-count>>  
ext-data-hop-count = [  
    hop-limit: uint,  
    hop-count: uint  
]
```

Authors' Addresses

Scott Burleigh
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099
US
Phone: +1 818 393 3353
Email: Scott.C.Burleigh@jpl.nasa.gov

Kevin Fall
Roland Computing Services
3871 Piedmont Ave. Suite 8
Oakland, CA 94611
US
Email: kfall+racs@kfall.com

Edward J. Birrane
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd
Laurel, MD 20723
US
Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: May 5, 2021

E. Birrane
K. McKeever
JHU/APL
November 1, 2020

Bundle Protocol Security Specification
draft-ietf-dtn-bpsec-24

Abstract

This document defines a security protocol providing data integrity and confidentiality services for the Bundle Protocol.

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 5, 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1.	Introduction	3
1.1.	Supported Security Services	3
1.2.	Specification Scope	4
1.3.	Related Documents	5
1.4.	Terminology	6
2.	Design Decisions	7
2.1.	Block-Level Granularity	7
2.2.	Multiple Security Sources	8
2.3.	Mixed Security Policy	9
2.4.	User-Defined Security Contexts	9
2.5.	Deterministic Processing	9
3.	Security Blocks	10
3.1.	Block Definitions	10
3.2.	Uniqueness	10
3.3.	Target Multiplicity	12
3.4.	Target Identification	13
3.5.	Block Representation	13
3.6.	Abstract Security Block	13
3.7.	Block Integrity Block	16
3.8.	Block Confidentiality Block	17
3.9.	Block Interactions	18
3.10.	Parameter and Result Identification	20
3.11.	BSP Block Examples	20
3.11.1.	Example 1: Constructing a Bundle with Security	20
3.11.2.	Example 2: Adding More Security At A New Node	21
4.	Canonical Forms	23
5.	Security Processing	24
5.1.	Bundles Received from Other Nodes	24
5.1.1.	Receiving BCBs	24
5.1.2.	Receiving BIBs	25
5.2.	Bundle Fragmentation and Reassembly	26
6.	Key Management	27
7.	Security Policy Considerations	27
7.1.	Security Reason Codes	28
8.	Security Considerations	29
8.1.	Attacker Capabilities and Objectives	30
8.2.	Attacker Behaviors and BPSec Mitigations	31
8.2.1.	Eavesdropping Attacks	31
8.2.2.	Modification Attacks	32
8.2.3.	Topology Attacks	33
8.2.4.	Message Injection	33
9.	Security Context Considerations	34
9.1.	Mandating Security Contexts	34
9.2.	Identification and Configuration	35
9.3.	Authorship	36
10.	Defining Other Security Blocks	37

11. IANA Considerations	39
11.1. Bundle Block Types	39
11.2. Bundle Status Report Reason Codes	39
11.3. Security Context Identifiers	40
12. References	40
12.1. Normative References	40
12.2. Informative References	41
Appendix A. Acknowledgements	41
Authors' Addresses	42

1. Introduction

This document defines security features for the Bundle Protocol (BP) [I-D.ietf-dtn-bpbis] and is intended for use in Delay Tolerant Networks (DTNs) to provide security services between a security source and a security acceptor. When the security source is the bundle source and when the security acceptor is the bundle destination, the security service provides end-to-end protection.

The Bundle Protocol specification [I-D.ietf-dtn-bpbis] defines DTN as referring to "a networking architecture providing communications in and/or through highly stressed environments" where "BP may be viewed as sitting at the application layer of some number of constituent networks, forming a store-carry-forward overlay network". The term "stressed" environment refers to multiple challenging conditions including intermittent connectivity, large and/or variable delays, asymmetric data rates, and high bit error rates.

It should be presumed that the BP will be deployed such that the network cannot be trusted, posing the usual security challenges related to confidentiality and integrity. However, the stressed nature of the BP operating environment imposes unique conditions where usual transport security mechanisms may not be sufficient. For example, the store-carry-forward nature of the network may require protecting data at rest, preventing unauthorized consumption of critical resources such as storage space, and operating without regular contact with a centralized security oracle (such as a certificate authority).

An end-to-end security service is needed that operates in all of the environments where the BP operates.

1.1. Supported Security Services

BPSec provides integrity and confidentiality services for BP bundles, as defined in this section.

Integrity services ensure that changes to target data within a bundle can be discovered. Data changes may be caused by processing errors, environmental conditions, or intentional manipulation. In the context of BPsec, integrity services apply to plain text in the bundle.

Confidentiality services ensure that target data is unintelligible to nodes in the DTN, except for authorized nodes possessing special information. This generally means producing cipher text from plain text and generating authentication information for that cipher text. Confidentiality, in this context, applies to the contents of target data and does not extend to hiding the fact that confidentiality exists in the bundle.

NOTE: Hop-by-hop authentication is NOT a supported security service in this specification, for two reasons.

1. The term "hop-by-hop" is ambiguous in a BP overlay, as nodes that are adjacent in the overlay may not be adjacent in physical connectivity. This condition is difficult or impossible to detect and therefore hop-by-hop authentication is difficult or impossible to enforce.
2. Hop-by-hop authentication cannot be deployed in a network if adjacent nodes in the network have incompatible security capabilities.

1.2. Specification Scope

This document defines the security services provided by the BPsec. This includes the data specification for representing these services as BP extension blocks, and the rules for adding, removing, and processing these blocks at various points during the bundle's traversal of the DTN.

BPsec addresses only the security of data traveling over the DTN, not the underlying DTN itself. Furthermore, while the BPsec protocol can provide security-at-rest in a store-carry-forward network, it does not address threats which share computing resources with the DTN and/or BPsec software implementations. These threats may be malicious software or compromised libraries which intend to intercept data or recover cryptographic material. Here, it is the responsibility of the BPsec implementer to ensure that any cryptographic material, including shared secret or private keys, is protected against access within both memory and storage devices.

Completely trusted networks are extremely uncommon. Amongst untrusted networks, different networking conditions and operational

considerations require varying strengths of security mechanism. Mandating a single security context may result in too much security for some networks and too little security in others. It is expected that separate documents define different security contexts for use in different networks. A set of default security contexts are defined in ([I-D.ietf-dtn-bpsec-interop-sc]) and provide basic security services for interoperability testing and for operational use on the terrestrial Internet.

This specification addresses neither the fitness of externally-defined cryptographic methods nor the security of their implementation.

This specification does not address the implementation of security policy and does not provide a security policy for the BPsec. Similar to cipher suites, security policies are based on the nature and capabilities of individual networks and network operational concepts. This specification does provide policy considerations when building a security policy.

With the exception of the Bundle Protocol, this specification does not address how to combine the BPsec security blocks with other protocols, other BP extension blocks, or other best practices to achieve security in any particular network implementation.

1.3. Related Documents

This document is best read and understood within the context of the following other DTN documents:

"Delay-Tolerant Networking Architecture" [RFC4838] defines the architecture for DTNs and identifies certain security assumptions made by existing Internet protocols that are not valid in a DTN.

The Bundle Protocol [I-D.ietf-dtn-bpbis] defines the format and processing of bundles, defines the extension block format used to represent BPsec security blocks, and defines the canonical block structure used by this specification.

The Concise Binary Object Representation (CBOR) format [RFC7049] defines a data format that allows for small code size, fairly small message size, and extensibility without version negotiation. The block-specific-data associated with BPsec security blocks are encoded in this data format.

The Bundle Security Protocol [RFC6257] and Streamlined Bundle Security Protocol [I-D.birrane-dtn-sbsp] documents introduced the

concepts of using BP extension blocks for security services in a DTN. The BPsec is a continuation and refinement of these documents.

1.4. Terminology

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.

This section defines terminology either unique to the BPsec or otherwise necessary for understanding the concepts defined in this specification.

- o Bundle Destination - the node which receives a bundle and delivers the payload of the bundle to an application. Also, the Node ID of the Bundle Protocol Agent (BPA) receiving the bundle. The bundle destination acts as the security acceptor for every security target in every security block in every bundle it receives.
- o Bundle Source - the node which originates a bundle. Also, the Node ID of the BPA originating the bundle.
- o Cipher Suite - a set of one or more algorithms providing integrity and/or confidentiality services. Cipher suites may define user parameters (e.g. secret keys to use) but do not provide values for those parameters.
- o Forwarder - any node that transmits a bundle in the DTN. Also, the Node ID of the BPA that sent the bundle on its most recent hop.
- o Intermediate Receiver, Waypoint, or Next Hop - any node that receives a bundle from a Forwarder that is not the Bundle Destination. Also, the Node ID of the BPA at any such node.
- o Path - the ordered sequence of nodes through which a bundle passes on its way from Source to Destination. The path is not necessarily known in advance by the bundle or any BPAs in the DTN.
- o Security Acceptor - a bundle node that processes and dispositions one or more security blocks in a bundle. Also, the Node ID of that node.
- o Security Block - a BPsec extension block in a bundle.

- o Security Context - the set of assumptions, algorithms, configurations and policies used to implement security services.
- o Security Operation - the application of a given security service to a security target, notated as OP(security service, security target). For example, OP(bcb-confidentiality, payload). Every security operation in a bundle MUST be unique, meaning that a given security service can only be applied to a security target once in a bundle. A security operation is implemented by a security block.
- o Security Service - a process that gives some protection to a security target. For example, this specification defines security services for plain text integrity (bib-integrity), and authenticated plain text confidentiality with additional authenticated data (bcb-confidentiality).
- o Security Source - a bundle node that adds a security block to a bundle. Also, the Node ID of that node.
- o Security Target - the block within a bundle that receives a security service as part of a security operation.

2. Design Decisions

The application of security services in a DTN is a complex endeavor that must consider physical properties of the network (such as connectivity and propagation times), policies at each node, application security requirements, and current and future threat environments. This section identifies those desirable properties that guide design decisions for this specification and are necessary for understanding the format and behavior of the BPSec protocol.

2.1. Block-Level Granularity

Security services within this specification must allow different blocks within a bundle to have different security services applied to them.

Blocks within a bundle represent different types of information. The primary block contains identification and routing information. The payload block carries application data. Extension blocks carry a variety of data that may augment or annotate the payload, or otherwise provide information necessary for the proper processing of a bundle along a path. Therefore, applying a single level and type of security across an entire bundle fails to recognize that blocks in a bundle represent different types of information with different security needs.

For example, a payload block might be encrypted to protect its contents and an extension block containing summary information related to the payload might be integrity signed but unencrypted to provide waypoints access to payload-related data without providing access to the payload.

2.2. Multiple Security Sources

A bundle can have multiple security blocks and these blocks can have different security sources. BPSEC implementations **MUST NOT** assume that all blocks in a bundle have the same security operations applied to them.

The Bundle Protocol allows extension blocks to be added to a bundle at any time during its existence in the DTN. When a waypoint adds a new extension block to a bundle, that extension block **MAY** have security services applied to it by that waypoint. Similarly, a waypoint **MAY** add a security service to an existing block, consistent with its security policy.

When a waypoint adds a security service to the bundle, the waypoint is the security source for that service. The security block(s) which represent that service in the bundle may need to record this security source as the bundle destination might need this information for processing.

For example, a bundle source may choose to apply an integrity service to its plain text payload. Later a waypoint node, representing a gateway to another portion of the DTN, may receive the bundle and choose to apply a confidentiality service. In this case, the integrity security source is the bundle source and the confidentiality security source is the waypoint node.

In cases where the security source and security acceptor are not the bundle source and bundle destination, it is possible that the bundle will reach the bundle destination prior to reaching a security acceptor. In cases where this may be a practical problem, it is recommended that solutions such as bundle encapsulation can be used to ensure that a bundle be delivered to a security acceptor prior to being delivered to the bundle destination. Generally, if a bundle reaches a waypoint that has the appropriate configuration and policy to act as a security acceptor for a security service in the bundle, then the waypoint should act as that security acceptor.

2.3. Mixed Security Policy

The security policy enforced by nodes in the DTN may differ.

Some waypoints will have security policies that require evaluating security services even if they are not the bundle destination or the final intended acceptor of the service. For example, a waypoint could choose to verify an integrity service even though the waypoint is not the bundle destination and the integrity service will be needed by other nodes along the bundle's path.

Some waypoints will determine, through policy, that they are the intended recipient of the security service and terminate the security service in the bundle. For example, a gateway node could determine that, even though it is not the destination of the bundle, it should verify and remove a particular integrity service or attempt to decrypt a confidentiality service, before forwarding the bundle along its path.

Some waypoints could understand security blocks but refuse to process them unless they are the bundle destination.

2.4. User-Defined Security Contexts

A security context is the union of security algorithms (cipher suites), policies associated with the use of those algorithms, and configuration values. Different contexts may specify different algorithms, different policies, or different configuration values used in the implementation of their security services. BPsec provides a mechanism to define security contexts. Users may select from registered security contexts and customize those contexts through security context parameters.

For example, some users might prefer a SHA2 hash function for integrity whereas other users might prefer a SHA3 hash function. Providing either separate security contexts or a single, parameterized security context allows users flexibility in applying the desired cipher suite, policy, and configuration when populating a security block.

2.5. Deterministic Processing

Whenever a node determines that it must process more than one security block in a received bundle (either because the policy at a waypoint states that it should process security blocks or because the node is the bundle destination) the order in which security blocks are processed must be deterministic. All nodes must impose this same deterministic processing order for all security blocks. This

specification provides determinism in the application and evaluation of security services, even when doing so results in a loss of flexibility.

3. Security Blocks

3.1. Block Definitions

This specification defines two types of security block: the Block Integrity Block (BIB) and the Block Confidentiality Block (BCB).

The BIB is used to ensure the integrity of its plain text security target(s). The integrity information in the BIB MAY be verified by any node along the bundle path from the BIB security source to the bundle destination. Waypoints add or remove BIBs from bundles in accordance with their security policy. BIBs are never used for integrity protection of the cipher text provided by a BCB. Because security policy at BPsec nodes may differ regarding integrity verification, BIBs do not guarantee hop-by-hop authentication, as discussed in Section 1.1.

The BCB indicates that the security target(s) have been encrypted at the BCB security source in order to protect their content while in transit. The BCB is decrypted by security acceptor nodes in the network, up to and including the bundle destination, as a matter of security policy. BCBs additionally provide integrity protection mechanisms for the cipher text they generate.

3.2. Uniqueness

Security operations in a bundle MUST be unique; the same security service MUST NOT be applied to a security target more than once in a bundle. Since a security operation is represented by a security block, this means that multiple security blocks of the same type cannot share the same security targets. A new security block MUST NOT be added to a bundle if a pre-existing security block of the same type is already defined for the security target of the new security block.

This uniqueness requirement ensures that there is no ambiguity related to the order in which security blocks are processed or how security policy can be specified to require certain security services be present in a bundle.

Using the notation OP(service, target), several examples illustrate this uniqueness requirement.

- o Signing the payload twice: The two operations OP(bib-integrity, payload) and OP(bib-integrity, payload) are redundant and MUST NOT both be present in the same bundle at the same time.
- o Signing different blocks: The two operations OP(bib-integrity, payload) and OP(bib-integrity, extension_block_1) are not redundant and both may be present in the same bundle at the same time. Similarly, the two operations OP(bib-integrity, extension_block_1) and OP(bib-integrity, extension_block_2) are also not redundant and may both be present in the bundle at the same time.
- o Different Services on same block: The two operations OP(bib-integrity, payload) and OP(bcb-confidentiality, payload) are not inherently redundant and may both be present in the bundle at the same time, pursuant to other processing rules in this specification.
- o Different services from different block types: The notation OP(service, target) refers specifically to a security block, as the security block is the embodiment of a security service applied to a security target in a bundle. Were some Other Security Block (OSB) to be defined providing an integrity service, then the operations OP(bib-integrity, target) and OP(osb-integrity, target) MAY both be present in the same bundle if so allowed by the definition of the OSB, as discussed in Section 10.

NOTES:

A security block may be removed from a bundle as part of security processing at a waypoint node with a new security block being added to the bundle by that node. In this case, conflicting security blocks never co-exist in the bundle at the same time and the uniqueness requirement is not violated.

A cipher text integrity mechanism (such as associated authenticated data) calculated by a cipher suite and transported in a BCB is considered part of the confidentiality service and, therefore, unique from the plain text integrity service provided by a BIB.

The security blocks defined in this specification (BIB and BCB) are designed with the intention that the BPA adding these blocks is the authoritative source of the security service. If a BPA adds a BIB on a security target, then the BIB is expected to be the authoritative source of integrity for that security target. If a BPA adds a BCB to a security target, then the BCB is expected to be the authoritative source of confidentiality for that

security target. More complex scenarios, such as having multiple nodes in a network sign the same security target, can be accommodated using the definition of custom security contexts (Section 9) and/or the definition of other security blocks (Section 10).

3.3. Target Multiplicity

A single security block MAY represent multiple security operations as a way of reducing the overall number of security blocks present in a bundle. In these circumstances, reducing the number of security blocks in the bundle reduces the amount of redundant information in the bundle.

A set of security operations can be represented by a single security block when all of the following conditions are true.

- o The security operations apply the same security service. For example, they are all integrity operations or all confidentiality operations.
- o The security context parameters for the security operations are identical.
- o The security source for the security operations is the same, meaning the set of operations are being added by the same node.
- o No security operations have the same security target, as that would violate the need for security operations to be unique.
- o None of the security operations conflict with security operations already present in the bundle.

When representing multiple security operations in a single security block, the information that is common across all operations is represented once in the security block, and the information which is different (e.g., the security targets) are represented individually.

It is RECOMMENDED that if a node processes any security operation in a security block that it process all security operations in the security block. This allows security sources to assert that the set of security operations in a security block are expected to be processed by the same security acceptor. However, the determination of whether a node actually is a security acceptor or not is a matter of the policy of the node itself. In cases where a receiving node determines that it is the security acceptor of only a subset of the security operations in a security block, the node may choose to only process that subset of security operations.

3.4. Target Identification

A security target is a block in the bundle to which a security service applies. This target must be uniquely and unambiguously identifiable when processing a security block. The definition of the extension block header from [I-D.ietf-dtn-bpbis] provides a "Block Number" field suitable for this purpose. Therefore, a security target in a security block MUST be represented as the Block Number of the target block.

3.5. Block Representation

Each security block uses the Canonical Bundle Block Format as defined in [I-D.ietf-dtn-bpbis]. That is, each security block is comprised of the following elements:

- o block type code
- o block number
- o block processing control flags
- o CRC type
- o block-type-specific-data
- o CRC field (if present)

Security-specific information for a security block is captured in the block-type-specific-data field.

3.6. Abstract Security Block

The structure of the security-specific portions of a security block is identical for both the BIB and BCB Block Types. Therefore, this section defines an Abstract Security Block (ASB) data structure and discusses the definition, processing, and other constraints for using this structure. An ASB is never directly instantiated within a bundle, it is only a mechanism for discussing the common aspects of BIB and BCB security blocks.

The fields of the ASB SHALL be as follows, listed in the order in which they must appear.

Security Targets:

This field identifies the block(s) targeted by the security operation(s) represented by this security block. Each target block is represented by its unique Block Number. This field

SHALL be represented by a CBOR array of data items. Each target within this CBOR array SHALL be represented by a CBOR unsigned integer. This array MUST have at least 1 entry and each entry MUST represent the Block Number of a block that exists in the bundle. There MUST NOT be duplicate entries in this array. The order of elements in this list has no semantic meaning outside of the context of this block. Within the block, the ordering of targets must match the ordering of results associated with these targets.

Security Context Id:

This field identifies the security context used to implement the security service represented by this block and applied to each security target. This field SHALL be represented by a CBOR unsigned integer. The values for this Id should come from the registry defined in Section 11.3

Security Context Flags:

This field identifies which optional fields are present in the security block. This field SHALL be represented as a CBOR unsigned integer whose contents shall be interpreted as a bit field. Each bit in this bit field indicates the presence (bit set to 1) or absence (bit set to 0) of optional data in the security block. The association of bits to security block data is defined as follows.

Bit 0 (the least-significant bit, 0x01): Security Context Parameters Present Flag.

Bit 1 (0x02): Security Source Present Flag.

Bit >1 Reserved

Implementations MUST set reserved bits to 0 when writing this field and MUST ignore the values of reserved bits when reading this field. For unreserved bits, a value of 1 indicates that the associated security block field MUST be included in the security block. A value of 0 indicates that the associated security block field MUST NOT be in the security block.

Security Source (Optional):

This field identifies the Endpoint that inserted the security block in the bundle. If the security source field is not present then the source MUST be inferred from other information, such as the bundle source, previous hop, or other values defined by security policy. This field SHALL be represented by a CBOR array in accordance with

[I-D.ietf-dtn-bpbis] rules for representing Endpoint Identifiers (EIDs).

Security Context Parameters (Optional):

This field captures one or more security context parameters that should be used when processing the security service described by this security block. This field SHALL be represented by a CBOR array. Each entry in this array is a single security context parameter. A single parameter SHALL also be represented as a CBOR array comprising a 2-tuple of the id and value of the parameter, as follows.

- * Parameter Id. This field identifies which parameter is being specified. This field SHALL be represented as a CBOR unsigned integer. Parameter Ids are selected as described in Section 3.10.
- * Parameter Value. This field captures the value associated with this parameter. This field SHALL be represented by the applicable CBOR representation of the parameter, in accordance with Section 3.10.

The logical layout of the parameters array is illustrated in Figure 1.

Parameter 1		Parameter 2		...	Parameter N	
Id	Value	Id	Value		Id	Value

Figure 1: Security Context Parameters

Security Results:

This field captures the results of applying a security service to the security targets of the security block. This field SHALL be represented as a CBOR array of target results. Each entry in this array represents the set of security results for a specific security target. The target results MUST be ordered identically to the Security Targets field of the security block. This means that the first set of target results in this array corresponds to the first entry in the Security Targets field of the security block, and so on. There MUST be one entry in this array for each entry in the Security Targets field of the security block.

The set of security results for a target is also represented as a CBOR array of individual results. An individual result is

represented as a 2-tuple of a result id and a result value, defined as follows.

- * Result Id. This field identifies which security result is being specified. Some security results capture the primary output of a cipher suite. Other security results contain additional annotative information from cipher suite processing. This field SHALL be represented as a CBOR unsigned integer. Security result Ids will be as specified in Section 3.10.
- * Result Value. This field captures the value associated with the result. This field SHALL be represented by the applicable CBOR representation of the result value, in accordance with Section 3.10.

The logical layout of the security results array is illustrated in Figure 2. In this figure there are N security targets for this security block. The first security target contains M results and the Nth security target contains K results.

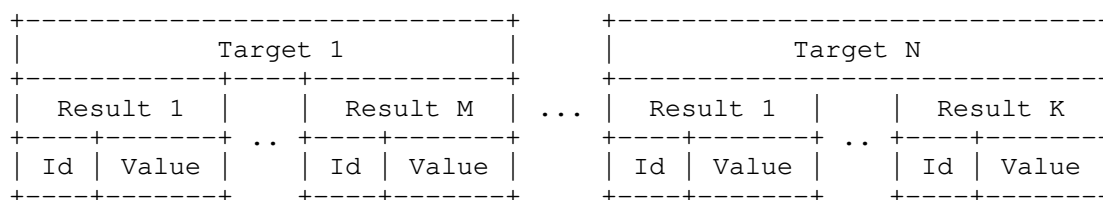


Figure 2: Security Results

3.7. Block Integrity Block

A BIB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The block-type-specific-data field follows the structure of the ASB.

A security target listed in the Security Targets field MUST NOT reference a security block defined in this specification (e.g., a BIB or a BCB).

The Security Context MUST utilize an authentication mechanism or an error detection mechanism.

The EID of the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of security context parameters described in Section 3.10.

Notes:

- o Designers SHOULD carefully consider the effect of setting flags that either discard the block or delete the bundle in the event that this block cannot be processed.
- o Since OP(bib-integrity, target) is allowed only once in a bundle per target, it is RECOMMENDED that users wishing to support multiple integrity mechanisms for the same target define a multi-result security context. Such a context could generate multiple security results for the same security target using different integrity-protection mechanisms or different configurations for the same integrity-protection mechanism.
- o A BIB is used to verify the plain text integrity of its security target. However, a single BIB MAY include security results for blocks other than its security target when doing so establishes a needed relationship between the BIB security target and other blocks in the bundle (such as the primary block).
- o Security information MAY be checked at any hop on the way to the bundle destination that has access to the required keying information, in accordance with Section 3.9.

3.8. Block Confidentiality Block

A BCB is a bundle extension block with the following characteristics.

The Block Type Code value is as specified in Section 11.1.

The Block Processing Control flags value can be set to whatever values are required by local policy with the following exceptions. BCB blocks MUST have the "block must be replicated in every fragment" flag set if one of the targets is the payload block. Having that BCB in each fragment indicates to a receiving node that the payload portion of each fragment represents cipher text. BCB blocks MUST NOT have the "block must be removed from bundle if it can't be processed" flag set. Removing a BCB from a bundle without decrypting its security targets removes information from the bundle necessary for their later decryption.

The block-type-specific-data fields follow the structure of the ASB.

A security target listed in the Security Targets field can reference the payload block, a non-security extension block, or a BIB. A BCB MUST NOT include another BCB as a security target. A BCB MUST NOT target the primary block. A BCB MUST NOT target a BIB block unless it shares a security target with that BIB block.

Any Security Context used by a BCB MUST utilize a confidentiality cipher that provides authenticated encryption with associated data (AEAD).

Additional information created by a cipher suite (such as an authentication tag) can be placed either in a security result field or in the generated cipher text. The determination of where to place this information is a function of the cipher suite and security context used.

The EID of the security source MAY be present. If this field is not present, then the security source of the block SHOULD be inferred according to security policy and MAY default to the bundle source. The security source MAY be specified as part of security context parameters described in Section 3.10.

The BCB modifies the contents of its security target(s). When a BCB is applied, the security target body data are encrypted "in-place". Following encryption, the security target block-type-specific-data field contains cipher text, not plain text.

Notes:

- o It is RECOMMENDED that designers carefully consider the effect of setting flags that delete the bundle in the event that this block cannot be processed.
- o The BCB block processing control flags can be set independently from the processing control flags of the security target(s). The setting of such flags should be an implementation/policy decision for the encrypting node.

3.9. Block Interactions

The security block types defined in this specification are designed to be as independent as possible. However, there are some cases where security blocks may share a security target creating processing dependencies.

If a security target of a BCB is also a security target of a BIB, an undesirable condition occurs where a waypoint would be unable to validate the BIB because one of its security target's contents have been encrypted by a BCB. To address this situation the following processing rules MUST be followed.

- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB also match all of the security targets of an existing BIB, then the existing BIB MUST also be encrypted. This can be accomplished by either adding a new BCB that targets the existing BIB, or by adding the BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.
- o When adding a BCB to a bundle, if some (or all) of the security targets of the BCB match some (but not all) of the security targets of a BIB then that BIB MUST be altered in the following way. Any security results in the BIB associated with the BCB security targets MUST be removed from the BIB and placed in a new BIB. This newly created BIB MUST then be encrypted. The encryption of the new BIB can be accomplished by either adding a new BCB that targets the new BIB, or by adding the new BIB to the list of security targets for the BCB. Deciding which way to represent this situation is a matter of security policy.
- o A BIB MUST NOT be added for a security target that is already the security target of a BCB as this would cause ambiguity in block processing order.
- o A BIB integrity value MUST NOT be checked if the BIB is the security target of an existing BCB. In this case, the BIB data is encrypted.
- o A BIB integrity value MUST NOT be checked if the security target associated with that value is also the security target of a BCB. In such a case, the security target data contains cipher text as it has been encrypted.
- o As mentioned in Section 3.7, a BIB MUST NOT have a BCB as its security target.

These restrictions on block interactions impose a necessary ordering when applying security operations within a bundle. Specifically, for a given security target, BIBs MUST be added before BCBs. This ordering MUST be preserved in cases where the current BPA is adding all of the security blocks for the bundle or whether the BPA is a waypoint adding new security blocks to a bundle that already contains security blocks.

In cases where a security source wishes to calculate both a plain text integrity mechanism and encrypt a security target, a BCB with a security context that generates an integrity-protection mechanism as one or more additional security results MUST be used instead of adding both a BIB and then a BCB for the security target at the security source.

3.10. Parameter and Result Identification

Each security context MUST define its own context parameters and results. Each defined parameter and result is represented as the tuple of an identifier and a value. Identifiers are always represented as a CBOR unsigned integer. The CBOR encoding of values is as defined by the security context specification.

Identifiers MUST be unique for a given security context but do not need to be unique amongst all security contexts.

An example of a security context can be found at [I-D.ietf-dtn-bpsec-interop-sc].

3.11. BSP Block Examples

This section provides two examples of BPsec blocks applied to a bundle. In the first example, a single node adds several security operations to a bundle. In the second example, a waypoint node received the bundle created in the first example and adds additional security operations. In both examples, the first column represents blocks within a bundle and the second column represents the Block Number for the block, using the terminology B1...Bn for the purpose of illustration.

3.11.1. Example 1: Constructing a Bundle with Security

In this example a bundle has four non-security-related blocks: the primary block (B1), two extension blocks (B4,B5), and a payload block (B6). The bundle source wishes to provide an integrity signature of the plain text associated with the primary block, the second extension block, and the payload. The bundle source also wishes to provide confidentiality for the first extension block. The resultant bundle is illustrated in Figure 3 and the security actions are described below.

Block in Bundle	ID
Primary Block	B1
BIB OP(bib-integrity, targets=B1, B5, B6)	B2
BCB OP(hcb-confidentiality, target=B4)	B3
Extension Block (encrypted)	B4
Extension Block	B5
Payload Block	B6

Figure 3: Security at Bundle Creation

The following security actions were applied to this bundle at its time of creation.

- o An integrity signature applied to the canonical form of the primary block (B1), the canonical form of the block-type-specific-data field of the second extension block (B5) and the canonical form of the payload block (B6). This is accomplished by a single BIB (B2) with multiple targets. A single BIB is used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BIBs could have been added instead.
- o Confidentiality for the first extension block (B4). This is accomplished by a BCB (B3). Once applied, the block-type-specific-data field of extension block B4 is encrypted. The BCB MUST hold an authentication tag for the cipher text either in the cipher text that now populates the first extension block or as a security result in the BCB itself, depending on which security context is used to form the BCB. A plain text integrity signature may also exist as a security result in the BCB if one is provided by the selected confidentiality security context.

3.11.2. Example 2: Adding More Security At A New Node

Consider that the bundle as it is illustrated in Figure 3 is now received by a waypoint node that wishes to encrypt the second extension block and the bundle payload. The waypoint security policy is to allow existing BIBs for these blocks to persist, as they may be required as part of the security policy at the bundle destination.

The resultant bundle is illustrated in Figure 4 and the security actions are described below. Note that block IDs provided here are ordered solely for the purpose of this example and not meant to impose an ordering for block creation. The ordering of blocks added to a bundle MUST always be in compliance with [I-D.ietf-dtn-bpbis].

Block in Bundle	ID
Primary Block	B1
BIB OP(bib-integrity, targets=B1)	B2
BIB (encrypted) OP(bib-integrity, targets=B5, B6)	B7
BCB OP(bcb-confidentiality, targets=B5, B6, B7)	B8
BCB OP(bcb-confidentiality, target=B4)	B3
Extension Block (encrypted)	B4
Extension Block (encrypted)	B5
Payload Block (encrypted)	B6

Figure 4: Security At Bundle Forwarding

The following security actions were applied to this bundle prior to its forwarding from the waypoint node.

- o Since the waypoint node wishes to encrypt the block-type-specific-data field of blocks B5 and B6, it MUST also encrypt the block-type-specific-data field of the BIBs providing plain text integrity over those blocks. However, BIB B2 could not be encrypted in its entirety because it also held a signature for the primary block (B1). Therefore, a new BIB (B7) is created and security results associated with B5 and B6 are moved out of BIB B2 and into BIB B7.
- o Now that there is no longer confusion of which plain text integrity signatures must be encrypted, a BCB is added to the bundle with the security targets being the second extension block (B5) and the payload (B6) as well as the newly created BIB holding their plain text integrity signatures (B7). A single new BCB is

used in this case because all three targets share a security source, security context, and security context parameters. Had this not been the case, multiple BCBs could have been added instead.

4. Canonical Forms

Security services require consistency and determinism in how information is presented to cipher suites at security sources, verifiers, and acceptors. For example, integrity services require that the same target information (e.g., the same bits in the same order) is provided to the cipher suite when generating an original signature and when validating a signature. Canonicalization algorithms transcode the contents of a security target into a canonical form.

Canonical forms are used to generate input to a security context for security processing at a BP node. If the values of a security target are unchanged, then the canonical form of that target will be the same even if the encoding of those values for wire transmission is different.

BPsec operates on data fields within bundle blocks (e.g., the block-type-specific-data field). In their canonical form, these fields MUST include their own CBOR encoding and MUST NOT include any other encapsulating CBOR encoding. For example, the canonical form of the block-type-specific-data field is a CBOR byte string existing within the CBOR array containing the fields of the extension block. The entire CBOR byte string is considered the canonical block-type-specific-data field. The CBOR array framing is not considered part of the field.

The canonical form of the primary block is as specified in [I-D.ietf-dtn-bpbis] with the following constraint.

- o CBOR values from the primary block MUST be canonicalized using the rules for Canonical CBOR, as specified in [RFC7049]. Canonical CBOR generally requires that integers and type lengths are encoded to be as small as possible, that map values be sorted, and that indefinite-length items be made into definite-length items.

All non-primary blocks share the same block structure and are canonicalized as specified in [I-D.ietf-dtn-bpbis] with the following constraints.

- o CBOR values from the non-primary block MUST be canonicalized using the rules for Canonical CBOR, as specified in [RFC7049].

- o Only the block-type-specific-data field may be provided to a cipher suite for encryption as part of a confidentiality security service. Other fields within a non-primary-block MUST NOT be encrypted or decrypted and MUST NOT be included in the canonical form used by the cipher suite for encryption and decryption. These other fields MAY have an integrity protection mechanism applied to them by treating them as associated authenticated data.
- o Reserved and unassigned flags in the block processing control flags field MUST be set to 0 in a canonical form as it is not known if those flags will change in transit.

Security contexts MAY define their own canonicalization algorithms and require the use of those algorithms over the ones provided in this specification. In the event of conflicting canonicalization algorithms, algorithms defined in a security context take precedence over this specification when constructing canonical forms for that security context.

5. Security Processing

This section describes the security aspects of bundle processing.

5.1. Bundles Received from Other Nodes

Security blocks must be processed in a specific order when received by a BP node. The processing order is as follows.

- o When BIBs and BCBs share a security target, BCBs MUST be evaluated first and BIBs second.

5.1.1. Receiving BCBs

If a received bundle contains a BCB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BCB. If so, the node MUST process those operations and remove any operation-specific information from the BCB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BCB have been removed from the BCB, the BCB MUST be removed from the bundle.

If the receiving node is the destination of the bundle, the node MUST decrypt any BCBs remaining in the bundle. If the receiving node is not the destination of the bundle, the node MUST process the BCB if directed to do so as a matter of security policy.

If the security policy of a node specifies that a node should have applied confidentiality to a specific security target and no such BCB is present in the bundle, then the node MUST process this security target in accordance with the security policy. It is RECOMMENDED that the node remove the security target from the bundle because the confidentiality (and possibly the integrity) of the security target cannot be guaranteed. If the removed security target is the payload block, the bundle MUST be discarded.

If an encrypted payload block cannot be decrypted (i.e., the cipher text cannot be authenticated), then the bundle MUST be discarded and processed no further. If an encrypted security target other than the payload block cannot be decrypted then the associated security target and all security blocks associated with that target MUST be discarded and processed no further. In both cases, requested status reports (see [I-D.ietf-dtn-bpbis]) MAY be generated to reflect bundle or block deletion.

When a BCB is decrypted, the recovered plain text for each security target MUST replace the cipher text in each of the security targets' block-type-specific-data fields. If the plain text is of different size than the cipher text, the CBOR byte string framing of this field must be updated to ensure this field remains a valid CBOR byte string. The length of the recovered plain text is known by the decrypting security context.

If a BCB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BCB with a single security operation for the purposes of report generation and policy processing.

5.1.2. Receiving BIBs

If a received bundle contains a BIB, the receiving node MUST determine whether it is the security acceptor for any of the security operations in the BIB. If so, the node MUST process those operations and remove any operation-specific information from the BIB prior to delivering data to an application at the node or forwarding the bundle. If processing a security operation fails, the target SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. When all security operations for a BIB have been removed from the BIB, the BIB MUST be removed from the bundle.

A BIB MUST NOT be processed if the security target of the BIB is also the security target of a BCB in the bundle. Given the order of operations mandated by this specification, when both a BIB and a BCB share a security target, it means that the security target must have

been encrypted after it was integrity signed and, therefore, the BIB cannot be verified until the security target has been decrypted by processing the BCB.

If the security policy of a node specifies that a node should have applied integrity to a specific security target and no such BIB is present in the bundle, then the node MUST process this security target in accordance with the security policy. It is RECOMMENDED that the node remove the security target from the bundle if the security target is not the payload or primary block. If the security target is the payload or primary block, the bundle MAY be discarded. This action can occur at any node that has the ability to verify an integrity signature, not just the bundle destination.

If a receiving node is not the security acceptor of a security operation in a BIB it MAY attempt to verify the security operation anyway to prevent forwarding corrupt data. If the verification fails, the node SHALL process the security target in accordance to local security policy. It is RECOMMENDED that if a payload integrity check fails at a waypoint that it is processed in the same way as if the check fails at the bundle destination. If the check passes, the node MUST NOT remove the security operation from the BIB prior to forwarding.

If a BIB contains multiple security operations, each operation processed by the node MUST be treated as if the security operation has been represented by a single BIB with a single security operation for the purposes of report generation and policy processing.

5.2. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle payload, and security services have been applied to that bundle, the fragmentation rules described in [I-D.ietf-dtn-bpbis] MUST be followed. As defined there and summarized here for completeness, only the payload block can be fragmented; security blocks, like all extension blocks, can never be fragmented.

Due to the complexity of payload block fragmentation, including the possibility of fragmenting payload block fragments, integrity and confidentiality operations are not to be applied to a bundle representing a fragment. Specifically, a BCB or BIB MUST NOT be added to a bundle if the "Bundle is a Fragment" flag is set in the Bundle Processing Control Flags field.

Security processing in the presence of payload block fragmentation may be handled by other mechanisms outside of the BPsec protocol or by applying BPsec blocks in coordination with an encapsulation

mechanism. A node should apply any confidentiality protection prior to performing any fragmentation.

6. Key Management

There exist a myriad of ways to establish, communicate, and otherwise manage key information in a DTN. Certain DTN deployments might follow established protocols for key management whereas other DTN deployments might require new and novel approaches. BPSec assumes that key management is handled as a separate part of network management and this specification neither defines nor requires a specific key management strategy.

7. Security Policy Considerations

When implementing BPSec, several policy decisions must be considered. This section describes key policies that affect the generation, forwarding, and receipt of bundles that are secured using this specification. No single set of policy decisions is envisioned to work for all secure DTN deployments.

- o If a bundle is received that contains combinations of security operations that are disallowed by this specification the BPA must determine how to handle the bundle. The bundle may be discarded, the block affected by the security operation may be discarded, or one security operation may be favored over another.
- o BPAs in the network must understand what security operations they should apply to bundles. This decision may be based on the source of the bundle, the destination of the bundle, or some other information related to the bundle.
- o If a waypoint has been configured to add a security operation to a bundle, and the received bundle already has the security operation applied, then the receiver must understand what to do. The receiver may discard the bundle, discard the security target and associated BPsec blocks, replace the security operation, or some other action.
- o It is RECOMMENDED that security operations be applied to every block in a bundle and that the default behavior of a bundle agent is to use the security services defined in this specification. Designers should only deviate from the use of security operations when the deviation can be justified – such as when doing so causes downstream errors when processing blocks whose contents must be inspected or changed at one or more hops along the path.

- o BCB security contexts can alter the size of extension blocks and the payload block. Security policy SHOULD consider how changes to the size of a block could negatively effect bundle processing (e.g., calculating storage needs and scheduling transmission times).
- o Adding a BIB to a security target that has already been encrypted by a BCB is not allowed. If this condition is likely to be encountered, there are (at least) three possible policies that could handle this situation.
 1. At the time of encryption, a security context can be selected which computes a plain text integrity-protection mechanism that is included as a security context result field.
 2. The encrypted block may be replicated as a new block with a new block number and given integrity protection.
 3. An encapsulation scheme may be applied to encapsulate the security target (or the entire bundle) such that the encapsulating structure is, itself, no longer the security target of a BCB and may therefore be the security target of a BIB.
- o Security policy SHOULD address whether cipher suites whose cipher text is larger than the initial plain text are permitted and, if so, for what types of blocks. Changing the size of a block may cause processing difficulties for networks that calculate block offsets into bundles or predict transmission times or storage availability as a function of bundle size. In other cases, changing the size of a payload as part of encryption has no significant impact.

7.1. Security Reason Codes

Bundle protocol agents (BPAs) must process blocks and bundles in accordance with both BP policy and BPsec policy. The decision to receive, forward, deliver, or delete a bundle may be communicated to the report-to address of the bundle, in the form of a status report, as a method of tracking the progress of the bundle through the network. The status report for a bundle may be augmented with a "reason code" explaining why the particular action was taken on the bundle.

This section describes a set of reason codes associated with the security processing of a bundle. Security reason codes are assigned in accordance with Section 11.2.

Missing Security Operation:

This reason code indicates that a bundle was missing one or more required security operations. This reason code is typically used by a security verifier or security acceptor.

Unknown Security Operation:

This reason code indicates that one or more security operations present in a bundle cannot be understood by the security verifier or security acceptor for the operation. For example, this reason code may be used if a security block references an unknown security context identifier or security context parameter. This reason code should not be used for security operations for which the node is not a security verifier or security acceptor; there is no requirement that all nodes in a network understand all security contexts, security context parameters, and security services for every bundle in a network.

Unexpected Security Operation:

This reason code indicates that a receiving node is neither a security verifier nor a security acceptor for at least one security operation in a bundle. This reason code should not be seen as an error condition; not every node is a security verifier or security acceptor for every security operation in every bundle. In certain networks, this reason code may be useful in identifying misconfigurations of security policy.

Failed Security Operation:

This reason code indicates that one or more security operations in a bundle failed to process as expected for reasons other than misconfiguration. This may occur when a security-source is unable to add a security block to a bundle. This may occur if the target of a security operation fails to verify using the defined security context at a security verifier. This may also occur if a security operation fails to be processed without error at a security acceptor.

Conflicting Security Operations:

This reason code indicates that two or more security operations in a bundle are not conformant with the BPSec specification and that security processing was unable to proceed because of a BPSec protocol violation.

8. Security Considerations

Given the nature of DTN applications, it is expected that bundles may traverse a variety of environments and devices which each pose unique security risks and requirements on the implementation of security

within BPSec. For these reasons, it is important to introduce key threat models and describe the roles and responsibilities of the BPSec protocol in protecting the confidentiality and integrity of the data against those threats. This section provides additional discussion on security threats that BPSec will face and describes how BPSec security mechanisms operate to mitigate these threats.

The threat model described here is assumed to have a set of capabilities identical to those described by the Internet Threat Model in [RFC3552], but the BPSec threat model is scoped to illustrate threats specific to BPSec operating within DTN environments and therefore focuses on man-in-the-middle (MITM) attackers. In doing so, it is assumed that the DTN (or significant portions of the DTN) are completely under the control of an attacker.

8.1. Attacker Capabilities and Objectives

BPSec was designed to protect against MITM threats which may have access to a bundle during transit from its source, Alice, to its destination, Bob. A MITM node, Mallory, is a non-cooperative node operating on the DTN between Alice and Bob that has the ability to receive bundles, examine bundles, modify bundles, forward bundles, and generate bundles at will in order to compromise the confidentiality or integrity of data within the DTN. There are three classes of MITM nodes which are differentiated based on their access to cryptographic material:

- o Unprivileged Node: Mallory has not been provisioned within the secure environment and only has access to cryptographic material which has been publicly-shared.
- o Legitimate Node: Mallory is within the secure environment and therefore has access to cryptographic material which has been provisioned to Mallory (i.e., K_M) as well as material which has been publicly-shared.
- o Privileged Node: Mallory is a privileged node within the secure environment and therefore has access to cryptographic material which has been provisioned to Mallory, Alice and/or Bob (i.e. K_M , K_A , and/or K_B) as well as material which has been publicly-shared.

If Mallory is operating as a privileged node, this is tantamount to compromise; BPSec does not provide mechanisms to detect or remove Mallory from the DTN or BPSec secure environment. It is up to the BPSec implementer or the underlying cryptographic mechanisms to provide appropriate capabilities if they are needed. It should also be noted that if the implementation of BPSec uses a single set of

shared cryptographic material for all nodes, a legitimate node is equivalent to a privileged node because $K_M == K_A == K_B$. For this reason, sharing cryptographic material in this way is not recommended.

A special case of the legitimate node is when Mallory is either Alice or Bob (i.e., $K_M == K_A$ or $K_M == K_B$). In this case, Mallory is able to impersonate traffic as either Alice or Bob, respectively, which means that traffic to and from that node can be decrypted and encrypted, respectively. Additionally, messages may be signed as originating from one of the endpoints.

8.2. Attacker Behaviors and BPsec Mitigations

8.2.1. Eavesdropping Attacks

Once Mallory has received a bundle, she is able to examine the contents of that bundle and attempt to recover any protected data or cryptographic keying material from the blocks contained within. The protection mechanism that BPsec provides against this action is the BCB, which encrypts the contents of its security target, providing confidentiality of the data. Of course, it should be assumed that Mallory is able to attempt offline recovery of encrypted data, so the cryptographic mechanisms selected to protect the data should provide a suitable level of protection.

When evaluating the risk of eavesdropping attacks, it is important to consider the lifetime of bundles on a DTN. Depending on the network, bundles may persist for days or even years. Long-lived bundles imply that the data exists in the network for a longer period of time and, thus, there may be more opportunities to capture those bundles. Additionally, bundles that are long-lived imply that the information stored within them may remain relevant and sensitive for long enough that, once captured, there is sufficient time to crack encryption associated with the bundle. If a bundle does persist on the network for years and the cipher suite used for a BCB provides inadequate protection, Mallory may be able to recover the protected data either before that bundle reaches its intended destination or before the information in the bundle is no longer considered sensitive.

NOTE: Mallory is not limited by the bundle lifetime and may retain a given bundle indefinitely.

NOTE: Irrespective of whether BPsec is used, traffic analysis will be possible.

8.2.2. Modification Attacks

As a node participating in the DTN between Alice and Bob, Mallory will also be able to modify the received bundle, including non-BPSEC data such as the primary block, payload blocks, or block processing control flags as defined in [I-D.ietf-dtn-bpbis]. Mallory will be able to undertake activities which include modification of data within the blocks, replacement of blocks, addition of blocks, or removal of blocks. Within BPSEC, both the BIB and BCB provide integrity protection mechanisms to detect or prevent data manipulation attempts by Mallory.

The BIB provides that protection to another block which is its security target. The cryptographic mechanisms used to generate the BIB should be strong against collision attacks and Mallory should not have access to the cryptographic material used by the originating node to generate the BIB (e.g., K_A). If both of these conditions are true, Mallory will be unable to modify the security target or the BIB and lead Bob to validate the security target as originating from Alice.

Since BPSEC security operations are implemented by placing blocks in a bundle, there is no in-band mechanism for detecting or correcting certain cases where Mallory removes blocks from a bundle. If Mallory removes a BCB, but keeps the security target, the security target remains encrypted and there is a possibility that there may no longer be sufficient information to decrypt the block at its destination. If Mallory removes both a BCB (or BIB) and its security target there is no evidence left in the bundle of the security operation. Similarly, if Mallory removes the BIB but not the security target there is no evidence left in the bundle of the security operation. In each of these cases, the implementation of BPSEC must be combined with policy configuration at endpoints in the network which describe the expected and required security operations that must be applied on transmission and are expected to be present on receipt. This or other similar out-of-band information is required to correct for removal of security information in the bundle.

A limitation of the BIB may exist within the implementation of BIB validation at the destination node. If Mallory is a legitimate node within the DTN, the BIB generated by Alice with K_A can be replaced with a new BIB generated with K_M and forwarded to Bob. If Bob is only validating that the BIB was generated by a legitimate user, Bob will acknowledge the message as originating from Mallory instead of Alice. Validating a BIB indicates only that the BIB was generated by a holder of the relevant key; it does not provide any guarantee that the bundle or block was created by the same entity. In order to provide verifiable integrity checks BCB should require an encryption

scheme that is Indistinguishable under adaptive Chosen Ciphertext Attack (IND-CCA2) secure. Such an encryption scheme will guard against signature substitution attempts by Mallory. In this case, Alice creates a BIB with the protected data block as the security target and then creates a BCB with both the BIB and protected data block as its security targets.

8.2.3. Topology Attacks

If Mallory is in a MITM position within the DTN, she is able to influence how any bundles that come to her may pass through the network. Upon receiving and processing a bundle that must be routed elsewhere in the network, Mallory has three options as to how to proceed: not forward the bundle, forward the bundle as intended, or forward the bundle to one or more specific nodes within the network.

Attacks that involve re-routing the packets throughout the network are essentially a special case of the modification attacks described in this section where the attacker is modifying fields within the primary block of the bundle. Given that BPSec cannot encrypt the contents of the primary block, alternate methods must be used to prevent this situation. These methods may include requiring BIBs for primary blocks, using encapsulation, or otherwise strategically manipulating primary block data. The specifics of any such mitigation technique are specific to the implementation of the deploying network and outside of the scope of this document.

Furthermore, routing rules and policies may be useful in enforcing particular traffic flows to prevent topology attacks. While these rules and policies may utilize some features provided by BPSec, their definition is beyond the scope of this specification.

8.2.4. Message Injection

Mallory is also able to generate new bundles and transmit them into the DTN at will. These bundles may either be copies or slight modifications of previously-observed bundles (i.e., a replay attack) or entirely new bundles generated based on the Bundle Protocol, BPSec, or other bundle-related protocols. With these attacks Mallory's objectives may vary, but may be targeting either the bundle protocol or application-layer protocols conveyed by the bundle protocol. The target could also be the storage and compute of the nodes running the bundle or application layer protocols (e.g., a denial of service to flood on the storage of the store-and-forward mechanism; or compute which would process the packets and perhaps prevent other activities).

BPsec relies on cipher suite capabilities to prevent replay or forged message attacks. A BCB used with appropriate cryptographic mechanisms may provide replay protection under certain circumstances. Alternatively, application data itself may be augmented to include mechanisms to assert data uniqueness and then protected with a BIB, a BCB, or both along with other block data. In such a case, the receiving node would be able to validate the uniqueness of the data.

For example, a BIB may be used to validate the integrity of a bundle's primary block, which includes a timestamp and lifetime for the bundle. If a bundle is replayed outside of its lifetime, then the replay attack will fail as the bundle will be discarded. Similarly, additional blocks such as the Bundle Age may be signed and validated to identify replay attacks. Finally, security context parameters within BIB and BCB blocks may include anti-replay mechanisms such as session identifiers, nonces, and dynamic passwords as supported by network characteristics.

9. Security Context Considerations

9.1. Mandating Security Contexts

Because of the diversity of networking scenarios and node capabilities that may utilize BPsec there is a risk that a single security context mandated for every possible BPsec implementation is not feasible. For example, a security context appropriate for a resource-constrained node with limited connectivity may be inappropriate for use in a well-resourced, well connected node.

This does not mean that the use of BPsec in a particular network is meant to be used without security contexts for interoperability and default behavior. Network designers must identify the minimal set of security contexts necessary for functions in their network. For example, a default set of security contexts could be created for use over the terrestrial Internet and required by any BPsec implementation communicating over the terrestrial Internet.

To ensure interoperability among various implementations, all BPsec implementations MUST support at least the current IETF standards-track mandatory security context(s). As of this writing, that BCP mandatory security context is specified in [I-D.ietf-dtn-bpsec-interop-sc], but the mandatory security context(s) might change over time in accordance with usual IETF processes. Such changes are likely to occur in the future if/when flaws are discovered in the applicable cryptographic algorithms, for example.

Additionally, BPsec implementations need to support the security contexts which are specified and/or used by the BP networks in which they are deployed.

If a node serves as a gateway amongst two or more networks, the BPsec implementation at that node needs to support the union of security contexts mandated in those networks.

BPsec has been designed to allow for a diversity of security contexts and for new contexts to be defined over time. The use of different security contexts does not change the BPsec protocol itself and the definition of new security contexts **MUST** adhere to the requirements of such contexts as presented in this section and generally in this specification.

Implementors should monitor the state of security context specifications to check for future updates and replacement.

9.2. Identification and Configuration

Security blocks uniquely identify the security context to be used in the processing of their security services. The security context for a security block **MUST** be uniquely identifiable and **MAY** use parameters for customization.

To reduce the number of security contexts used in a network, security context designers should make security contexts customizable through the definition of security context parameters. For example, a single security context could be associated with a single cipher suite and security context parameters could be used to configure the use of this security context with different key lengths and different key management options without needing to define separate security contexts for each possible option.

A single security context may be used in the application of more than one security service. This means that a security context identifier **MAY** be used with a BIB, with a BCB, or with any other BPsec-compliant security block. The definition of a security context **MUST** identify which security services may be used with the security context, how security context parameters are interpreted as a function of the security operation being supported, and which security results are produced for each security service.

Network operators must determine the number, type, and configuration of security contexts in a system. Networks with rapidly changing configurations may define relatively few security contexts with each context customized with multiple parameters. For networks with more stability, or an increased need for confidentiality, a larger number

of contexts can be defined with each context supporting few, if any, parameters.

Security Context Examples

Context Type	Parameters	Definition
Key Exchange AES	Encrypted Key, IV	AES-GCM-256 cipher suite with provided ephemeral key encrypted with a predetermined key encryption key and clear text initialization vector.
Pre-shared Key AES	IV	AES-GCM-256 cipher suite with predetermined key and predetermined key rotation policy.
Out of Band AES	None	AES-GCM-256 cipher suite with all info predetermined.

Table 1

9.3. Authorship

Developers or implementers should consider the diverse performance and conditions of networks on which the Bundle Protocol (and therefore BPsec) will operate. Specifically, the delay and capacity of delay-tolerant networks can vary substantially. Developers should consider these conditions to better describe the conditions when those contexts will operate or exhibit vulnerability, and selection of these contexts for implementation should be made with consideration for this reality. There are key differences that may limit the opportunity for a security context to leverage existing cipher suites and technologies that have been developed for use in traditional, more reliable networks:

- o Data Lifetime: Depending on the application environment, bundles may persist on the network for extended periods of time, perhaps even years. Cryptographic algorithms should be selected to ensure protection of data against attacks for a length of time reasonable for the application.
- o One-Way Traffic: Depending on the application environment, it is possible that only a one-way connection may exist between two endpoints, or if a two-way connection does exist, the round-trip time may be extremely large. This may limit the utility of session key generation mechanisms, such as Diffie-Hellman, as a two-way handshake may not be feasible or reliable.

- o Opportunistic Access: Depending on the application environment, a given endpoint may not be guaranteed to be accessible within a certain amount of time. This may make asymmetric cryptographic architectures which rely on a key distribution center or other trust center impractical under certain conditions.

When developing security contexts for use with BPsec, the following information SHOULD be considered for inclusion in these specifications.

- o Security Context Parameters. Security contexts MUST define their parameter Ids, the data types of those parameters, and their CBOR encoding.
- o Security Results. Security contexts MUST define their security result Ids, the data types of those results, and their CBOR encoding.
- o New Canonicalizations. Security contexts may define new canonicalization algorithms as necessary.
- o Cipher-Text Size. Security contexts MUST state whether their associated cipher suites generate cipher text (to include any authentication information) that is of a different size than the input plain text.

If a security context does not wish to alter the size of the plain text it should place overflow bytes and authentication tags in security result fields.

- o Block Header Information. Security contexts SHOULD include block header information that is considered to be immutable for the block. This information MAY include the block type code, block number, CRC Type and CRC field (if present or if missing and unlikely to be added later), and possibly certain block processing control flags. Designers should input these fields as additional data for integrity protection when these fields are expected to remain unchanged over the path the block will take from the security source to the security acceptor. Security contexts considering block header information MUST describe expected behavior when these fields fail their integrity verification.

10. Defining Other Security Blocks

Other security blocks (OSBs) may be defined and used in addition to the security blocks identified in this specification. Both the usage of BIB, BCB, and any future OSBs can co-exist within a bundle and can

be considered in conformance with BPsec if each of the following requirements are met by any future identified security blocks.

- o Other security blocks (OSBs) MUST NOT reuse any enumerations identified in this specification, to include the block type codes for BIB and BCB.
- o An OSB definition MUST state whether it can be the target of a BIB or a BCB. The definition MUST also state whether the OSB can target a BIB or a BCB.
- o An OSB definition MUST provide a deterministic processing order in the event that a bundle is received containing BIBs, BCBs, and OSBs. This processing order MUST NOT alter the BIB and BCB processing orders identified in this specification.
- o An OSB definition MUST provide a canonicalization algorithm if the default non-primary-block canonicalization algorithm cannot be used to generate a deterministic input for a cipher suite. This requirement can be waived if the OSB is defined so as to never be the security target of a BIB or a BCB.
- o An OSB definition MUST NOT require any behavior of a BPSEC-BPA that is in conflict with the behavior identified in this specification. In particular, the security processing requirements imposed by this specification must be consistent across all BPSEC-BPAs in a network.
- o The behavior of an OSB when dealing with fragmentation must be specified and MUST NOT lead to ambiguous processing states. In particular, an OSB definition should address how to receive and process an OSB in a bundle fragment that may or may not also contain its security target. An OSB definition should also address whether an OSB may be added to a bundle marked as a fragment.

Additionally, policy considerations for the management, monitoring, and configuration associated with blocks SHOULD be included in any OSB definition.

NOTE: The burden of showing compliance with processing rules is placed upon the specifications defining new security blocks and the identification of such blocks shall not, alone, require maintenance of this specification.

11. IANA Considerations

This specification includes fields requiring registries managed by IANA.

11.1. Bundle Block Types

This specification allocates two block types from the existing "Bundle Block Types" registry defined in [RFC6255].

Additional Entries for the Bundle Block-Type Codes Registry:

Value	Description	Reference
TBA	Block Integrity Block	This document
TBA	Block Confidentiality Block	This document

Table 2

The Bundle Block Types namespace notes whether a block type is meant for use in BP version 6, BP version 7, or both. The two block types defined in this specification are meant for use with BP version 7.

11.2. Bundle Status Report Reason Codes

This specification allocates five reason codes from the existing "Bundle Status Report Reason Codes" registry defined in [RFC6255].

Additional Entries for the Bundle Status Report Reason Codes Registry:

BP Version	Value	Description	Reference
7	TBD	Missing Security Operation	This document, Section 7.1
7	TBD	Unknown Security Operation	This document, Section 7.1
7	TBD	Unexpected Security Operation	This document, Section 7.1
7	TBD	Failed Security Operation	This document, Section 7.1
7	TBD	Conflicting Security Operation	This document, Section 7.1

11.3. Security Context Identifiers

BPsec has a Security Context Identifier field for which IANA is requested to create and maintain a new registry named "BPsec Security Context Identifiers". Initial values for this registry are given below.

The registration policy for this registry is: Specification Required.

The value range is: unsigned 16-bit integer.

BPsec Security Context Identifier Registry

Value	Description	Reference
< 0	Reserved	This document
0	Reserved	This document

Table 3

Negative security context identifiers are reserved for local/site-specific uses. The use of 0 as a security context identifier is for non-operational testing purposes only.

12. References

12.1. Normative References

- [I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-28 (work in progress), October 2020.
- [I-D.ietf-dtn-bpsec-interop-sc]
Birrane, E., "BPsec Default Security Contexts", draft-ietf-dtn-bpsec-interop-sc-02 (work in progress), November 2020.
- [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>>.

- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC6255] Blanchet, M., "Delay-Tolerant Networking Bundle Protocol IANA Registries", RFC 6255, DOI 10.17487/RFC6255, May 2011, <<https://www.rfc-editor.org/info/rfc6255>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.

12.2. Informative References

- [I-D.birrane-dtn-sbsp] Birrane, E., Pierce-Mayer, J., and D. Iannicca, "Streamlined Bundle Security Protocol Specification", draft-birrane-dtn-sbsp-01 (work in progress), October 2015.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, DOI 10.17487/RFC6257, May 2011, <<https://www.rfc-editor.org/info/rfc6257>>.

Appendix A. Acknowledgements

The following participants contributed technical material, use cases, and useful thoughts on the overall approach to this security specification: Scott Burleigh of the Jet Propulsion Laboratory, Angela Hennessy of the Laboratory for Telecommunications Sciences, and Amy Alford, Angela Dalton, and Cherita Corbett of the Johns Hopkins University Applied Physics Laboratory.

Authors' Addresses

Edward J. Birrane, III
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Kenneth McKeever
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 2237
Email: Ken.McKeever@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2021

E. Birrane
JHU/APL
October 31, 2020

BPsec Default Security Contexts
draft-ietf-dtn-bpsec-interop-sc-02

Abstract

This document defines default integrity and confidentiality security contexts that may be used with the Bundle Protocol Security Protocol (BPsec) implementations. These security contexts may be used for both testing the interoperability of BPsec implementations and for providing basic security operations when no other security contexts are defined or otherwise required for a network.

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 4, 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Integrity Security Context BIB-HMAC-SHA2	3
3.1. Overview	3
3.2. Scope	4
3.3. Parameters	5
3.3.1. SHA Variant	5
3.3.2. Encapsulated Key	6
3.3.3. Integrity Scope Flags	6
3.3.4. Enumerations	7
3.4. Results	7
3.5. Key Considerations	7
3.6. Canonicalization Algorithms	8
3.7. Processing	9
3.7.1. Keyed Hash Generation	9
3.7.2. Keyed Hash Verification	10
4. Security Context BCB-AES-GCM	10
4.1. Overview	11
4.2. Scope	11
4.3. Parameters	13
4.3.1. Initialization Vector (IV)	13
4.3.2. Key Length	13
4.3.3. Encapsulated Key	14
4.3.4. AAD Scope Flags	14
4.3.5. Enumerations	15
4.4. Results	15
4.4.1. Authentication Tag	15
4.4.2. Enumerations	16
4.5. Key Considerations	16
4.6. Canonicalization Algorithms	17
4.6.1. Cipher text related calculations	17
4.6.2. Additional Authenticated Data	18
4.7. Processing	18
4.7.1. Encryption	18
4.7.2. Decryption	20
5. IANA Considerations	21
5.1. Security Context Identifiers	21
6. Normative References	21
Appendix A. Acknowledgements	22
Author's Address	22

1. Introduction

The Bundle Protocol Security Protocol (BPsec) [I-D.ietf-dtn-bpsec] specification provides inter-bundle integrity and confidentiality operations for networks deploying the Bundle Protocol (BP) [I-D.ietf-dtn-bpbis]. BPsec defines BP extension blocks to carry security information produced under the auspices of some security context.

This document defines two security contexts (one for an integrity service and one for a confidentiality service) for populating BPsec Block Integrity Blocks (BIBs) and Block Confidentiality Blocks (BCBs).

These contexts generate information that **MUST** be encoded using the CBOR specification documented in [RFC7049].

2. 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.

3. Integrity Security Context BIB-HMAC-SHA2

3.1. Overview

The BIB-HMAC-SHA2 security context provides a keyed hash over a set of plain text information. This context uses the Secure Hash Algorithm 2 (SHA-2) discussed in [RFC4634] combined with the HMAC keyed hash discussed in [RFC2104].

BIB-HMAC-SHA2 supports three variants of HMAC-SHA, based on the supported length of the SHA-2 hash value. These variants correspond to HMAC256-SHA256, HMAC384-SHA384, and HMAC512-SHA512 as defined in [RFC8152] Table 7: HMAC Algorithm Values. The selection of which variant is used by this context is provided as a security context parameter.

The output of the HMAC shall be equal to the size of the SHA2 hashing function: 256 bits for SHA-256, 384 bits for SHA-384, and 512 bits for SHA-512.

The BIB-HMAC-SHA2 security context shall have the Security Context ID specified in Section 5.1.

3.2. Scope

The scope of BIB-HMAC-SHA2 refers to the set of information used to produce the plain text over which a keyed hash is calculated. This plain text is termed the "Integrity Protected Plain Text" (IPPT). The contents of the IPPT is constructed as the concatenation of information whose integrity is being preserved from the BIB-HMAC-SHA2 security source to its security acceptor. There are four types of information that can be used in the generation of the IPPT, based on how broadly the concept of integrity is being applied. These four types of information, whether they are required, and why they are important for integrity, are discussed as follows.

Security target contents

The contents of the block-type-specific data field of the security target MUST be included in the IPPT. Including this information protects the security target data and is considered the minimal, required set of information for an integrity service on the security target.

Primary block

The primary block identifies a bundle and, once created, the contents of this block are immutable. Changes to the primary block associated with the security target indicate that the security target (and BIB) may no longer be in the correct bundle.

For example, if a security target and associated BIB are copied from one bundle to another bundle, the BIB may still contain a verifiable signature for the security target unless information associated with the bundle primary block is included in the keyed hash carried by the BIB.

Including this information in the IPPT protects the integrity of the association of the security target with a specific bundle.

Security target other fields

The other fields of the security target include block identification and processing information. Changing this information changes how the security target is treated by nodes in the network even when the "user data" of the security target are otherwise unchanged.

For example, if the block processing control flags of a security target are different at a security verifier than they were originally set at the security source then the policy for handling the security target has been modified.

Including this information in the IPPT protects the integrity of the policy and identification of the security target data.

BIB other fields

The other fields of the BIB carrying the security result for this security context security block include block identification and processing information. Changing this information changes how the BIB is treated by nodes in the network, even when other aspects of the BIB are unchanged.

For example, if the block processing control flags of the BIB are different at a security verifier than they were originally set at the security source, then the policy for handling the BIB has been modified.

Including this information in the IPPT protects the integrity of the policy and identification of the security service in the bundle.

NOTE: The security context identifier and security context parameters of the security block are not included in the IPPT because these parameters, by definition, are required to verify or accept the security service. Successful verification at security verifiers and security acceptors implies that these parameters were unchanged since being specified at the security source.

The scope of the BIB-HMAC-SHA2 security context is configured using an optional security context parameter.

3.3. Parameters

BIB-HMAC-SHA2 can be parameterized to select SHA-2 variants, communicate key information, and define the scope of the IPPT.

3.3.1. SHA Variant

This optional parameter identifies which variant of the SHA-2 algorithm is to be used in the generation of the authentication code.

This value MUST be encoded as a CBOR unsigned integer.

Valid values for this parameter are as follows.

SHA Variant Parameter Values

Value	Description
5	HMAC256/SHA256 as defined in [RFC8152] Table 7: HMAC Algorithm Values
6	HMAC384/SHA384 as defined in [RFC8152] Table 7: HMAC Algorithm Values
7	HMAC512/SHA512 as defined in [RFC8152] Table 7: HMAC Algorithm Values

Table 1

When not provided, implementations SHOULD assume a value of 5 (indicating use of HMAC256/SHA256), unless an alternate default is established by security policy at the security source, verifier, or acceptor of this integrity service.

3.3.2. Encapsulated Key

This optional parameter contains the output of a Key Encapsulation Mechanism (KEM) run at the security source of this security context.

This value MUST be encoded as a CBOR byte string.

If provided, this information is used to retrieve the symmetric HMAC key used in the generation of security results for this security context. If not provided, security verifiers and acceptors MUST determine the proper key as a function of their local BPsec policy and configuration, as discussed in Section 3.5.

3.3.3. Integrity Scope Flags

This optional parameter contains a series of flags that describe what information is to be included with the block-type-specific data when constructing the IPPT value.

This value MUST be represented as a CBOR unsigned integer, the value of which MUST be processed as a bit field containing no more than 16 bits.

Bits in this field represent additional information to be included when generating an integrity signature over the security target. These bits are defined as follows.

- Bit 0 (the low-order bit, 0x1): Primary Block Flag.

- Bit 1 (0x02): Target Header Flag.
- Bit 2 (0x03): Security Header Flag.
- Bits 3-7 are reserved.
- Bits 8-15 are unassigned.

3.3.4. Enumerations

BIB-HMAC-SHA2 defines the following security context parameters.

BIB-HMAC-SHA2 Security Parameters

Id	Name	Encoding Type	Default Value
1	SHA Variant	UINT	256
2	Encapsulated Key	Byte Array	NONE
3	Integrity Scope Flags	UINT	0

Table 2

3.4. Results

BIB-HMAC-SHA2 defines the following security results.

BIB-HMAC-SHA2 Security Results

Result Id	Result Name	CBOR Encoding Type	Description
1	Expected HMAC	byte string	The output of the HMAC calculation at the security source.

Table 3

3.5. Key Considerations

BIB-HMAC-SHA2 does not define or otherwise mandate any method for key exchange, encryption, or encapsulation. The derivation of an appropriate key for use in the integrity service is considered

separate from the application of the integrity service for this context.

HMAC keys used with this context MUST be symmetric and MUST have a key length equal to the output of the HMAC.

It is assumed that any security verifier or security acceptor performing an integrity verification can determine the proper HMAC key to be used. Potential sources of the HMAC key include (but are not limited to) the following:

- Pre-placed keys selected based on local policy.

- Keys extracted from encapsulated key material carried in the BIB.

- Session keys negotiated via a mechanism external to the BIB.

BIB-HMAC-SHA2 provides no explicit requirements on the configuration, storage, or exchange of HMAC keys.

3.6. Canonicalization Algorithms

This section defines the canonicalization algorithm used to prepare the IPPT input to the BIB-HMAC-SHA2 integrity mechanism. The construction of the IPPT depends on the settings of the Integrity Scope Flags that may be provided as part of customizing the behavior of this security context.

In all cases, the canonical form of any portion of an extension block MUST be performed as described in [I-D.ietf-dtn-bpsec]. The canonicalization algorithms defined in [I-D.ietf-dtn-bpsec] adhere to the canonical forms for extension blocks defined in [I-D.ietf-dtn-bpbis] but resolve ambiguities related to how values are represented in CBOR.

The IPPT is constructed using the following process.

1. The canonical form of the IPPT starts as the empty set with length 0.
2. If the Integrity Scope parameter is present and the Primary Block Flag is set to 1, then a canonical form of the bundle's primary block MUST be calculated and the result appended to the IPPT.
3. If the Integrity Scope parameter is present and the Security Header flag is set to 1, then the canonical form of the Block Type Code, Block Number, and Block Processing Control Flags

associated with the BIB MUST be calculated and, in that order, appended to the IPPT.

4. If the Integrity Scope parameter is present and the Target Header flag is set to 1, then the canonical form of the Block Type Code, Block Number, and Block Processing Control Flags associated with the security target MUST be calculated and, in that order, appended to the IPPT.
5. The canonical form of the security target block-type-specific data MUST be calculated and appended to the IPPT.

3.7. Processing

3.7.1. Keyed Hash Generation

During keyed hash generation, two inputs are prepared for the the appropriate HMAC/SHA2 algorithm: the HMAC key and the IPPT. These data items MUST be generated as follows.

The HMAC key MUST have the appropriate length as required by local security policy. The key may be generated specifically for this integrity service, given as part of local security policy, or through some other key management mechanism as discussed in Section 3.5.

The IPPT MUST be generated as discussed in Section 3.6.

Upon successful hash generation the following actions MUST occur.

The keyed hash produced by the HMAC/SHA2 variant MUST be added as a security result for the BIB representing the security operation on this security target, as discussed in Section 3.4).

Finally, the BIB containing information about this security operation MUST be updated as follows. These operations may occur in any order.

The security context ID for the BIB MUST be set to the context identifier for BIB-HMAC-SHA2.

Any local flags used to generated the IPPT SHOULD be placed in the Integrity Scope flags security parameter for the BIB unless these flags are expected to be correctly configured at security verifiers and acceptors in the network.

The HMAC key MAY be encapsulated using some key encapsulation mechanism (to include encrypting with a key encryption key) and

the results of the encapsulation added as the Encapsulated Key security parameter for the BIB.

The SHA Variant used by this security context SHOULD be added as the SHA Variant security parameter for the BIB if it differs from the default key length. Otherwise, this parameter MAY be omitted if doing so provides a useful reduction in message sizes.

Problems encountered in the keyed hash generation MUST be processed in accordance with local BPsec security policy.

3.7.2. Keyed Hash Verification

During keyed hash verification, the input of the security target and a HMAC key are provided to the appropriate HMAC/SHA2 algorithm.

During keyed hash verification, two inputs are prepared for the the appropriate HMAC/SHA2 algorithm: the HMAC key and the IPPT. These data items MUST be generated as follows.

The HMAC key MUST be derived using the Encapsulated Key security parameter if such a parameter is included in the security context parameters of the BIB. Otherwise, this key MUST be derived in accordance with security policy at the verifying node as discussed in Section 3.5.

The IPPT MUST be generated as discussed in Section 3.6 with the value of Integrity Scope flags being taken from the Integrity Scope flags security context parameter. If the Integrity Scope flags parameter is not included in the security context parameters then these flags MAY be derived from local security policy.

The calculated HMAC output MUST be compared to the expected HMAC output encoded in the security results of the BIB for the security target. If the calculated HMAC and expected HMAC are identical, the verification MUST be considered a success. Otherwise, the verification MUST be considered a failure.

If the verification fails or if any needed parameters are missing then the verification MUST be treated as failed and processed in accordance with local security policy.

4. Security Context BCB-AES-GCM

4.1. Overview

The BCB-AES-GCM security context replaces the block-type-specific data field of its security target with cipher text generated using the Advanced Encryption Standard (AES) cipher operating in Galois/Counter Mode (GCM) [AES-GCM].

Additionally, the BCB-AES-GCM security context generates an authentication tag based on the plain text value of the block-type-specific data and other additional authenticated data that may be specified via parameters to this security context.

This security context supports three variants of AES-GCM, based on the supported length of the symmetric key. These variants correspond to A128GCM, A192GCM, and A256GCM as defined in [RFC8152] Table 9: Algorithm Value for AES-GCM.

The BCB-AES-GCM security context shall have the Security Context ID specified in Section 5.1.

4.2. Scope

There are two scopes associated with BCB-AES-GCM: the scope of the confidentiality service and the scope of the authentication service. The first defines the set of information provided to the AES-GCM cipher for the purpose of producing cipher text. The second defines the set of information used to generate an authentication tag.

The scope of the confidentiality service defines the set of information provided to the AES-GCM cipher for the purpose of producing cipher text. This MUST be the full set of plain text contained in the block-type-specific data field of the security target.

The scope of the authentication service defines the set of information used to generate an authentication tag carried with the security block. This information MUST include the plain text of the block-type-specific data field of the security target. This information MAY include other information (additional authenticated data), as follows.

Primary block

The primary block identifies a bundle and, once created, the contents of this block are immutable. Changes to the primary block associated with the security target indicate that the security target (and BCB) may no longer be in the correct bundle.

For example, if a security target and associated BCB are copied from one bundle to another bundle, the BCB may still be able to decrypt the security target even though these blocks were never intended to exist in the copied-to bundle.

Including this information as part of additional authenticated data ensures that security target (and security block) appear in the same bundle at the time of decryption as at the time of encryption.

Security target other fields

The other fields of the security target include block identification and processing information. Changing this information changes how the security target is treated by nodes in the network even when the "user data" of the security target are otherwise unchanged.

For example, if the block processing control flags of a security target are different at a security verifier than they were originally set at the security source then the policy for handling the security target has been modified.

Including this information as part of additional authenticated data ensures that the cipher text in the security target will not be used with a different set of block policy than originally set at the time of encryption.

BCB other fields

The other fields of the BCB include block identification and processing information. Changing this information changes how the BCB is treated by nodes in the network, even when other aspects of the BCB are unchanged.

For example, if the block processing control flags of the BCB are different at a security acceptor than they were originally set at the security source then the policy for handling the BCB has been modified.

Including this information as part of additional authenticated data ensures that the policy and identification of the security service in the bundle has not changed.

NOTE: The security context identifier and security context parameters of the security block are not included as additional authenticated data because these parameters, by definition, are those needed to verify or accept the security service. Therefore, it is expected that changes to these values would result in failures at security verifiers and security acceptors.

The scope of the BCB-AES-GCM security context is configured using an optional security context parameter.

4.3. Parameters

BCB-AES-GCM can be parameterized to specify the AES key length, initialization vector, key information, and identify additional authenticated data.

4.3.1. Initialization Vector (IV)

This optional parameter identifies the initialization vector (IV) used to initialize the AES-GCM cipher.

The length of the initialization vector, prior to any CBOR encoding, MUST be between 8-16 bytes. A value of 12 bytes SHOULD be used unless local security policy requires a different length.

This value MUST be encoded as a CBOR byte string.

The initialization vector may have any value with the caveat that a value MUST NOT be re-used for multiple encryptions using the same encryption key. This value MAY be re-used when encrypting with different keys. For example, if each encryption operation using BCB-AES-GCM uses a newly generated key, then the same IV may be reused.

4.3.2. Key Length

This optional parameter identifies the key length being used for the AES-GCM encryption.

This value MUST be encoded as a CBOR unsigned integer.

Valid values for this parameter are as follows.

Key Length Parameter Values

Value	Description
1	A128GCM as defined in [RFC8152] Table 9: Algorithm Values for AES-GCM
2	A192GCM as defined in [RFC8152] Table 9: Algorithm Values for AES-GCM
3	A256GCM as defined in [RFC8152] Table 9: Algorithm Values for AES-GCM

When not provided, implementations SHOULD assume a value of 3 (indicating use of A256GCM), unless an alternate default is established by security policy at the security source, verifier, or acceptor of this integrity service.

Regardless of the key length, the generated authentication tag MUST always be 128 bits.

4.3.3. Encapsulated Key

This optional parameter contains the output of a Key Encapsulation Mechanism (KEM) run at the security source of this security context.

This value MUST be encoded as a CBOR byte string.

If provided, this information is used to retrieve the symmetric AES key used in the generation of security results for this security context. If not provided, security verifiers and acceptors MUST determine the proper key as a function of their local BPSec policy and configuration, as discussed in Section 4.5.

4.3.4. AAD Scope Flags

This optional parameter contains a series of flags that describe what information is to be included with the block-type-specific data of the security target as part of additional authenticated data (AAD).

This value MUST be represented as a CBOR unsigned integer, the value of which MUST be processed as a bit field containing no more than 16 bits.

Bits in this field represent additional information to be included when generating an integrity signature over the security target. These bits are defined as follows.

- Bit 0 (the low-order bit, 0x1): Primary Block Flag.
- Bit 1 (0x02): Target Header Flag.
- Bit 2 (0x03): Security Header Flag.
- Bits 3-7 are reserved.
- Bits 8-15 are unassigned.

4.3.5. Enumerations

BCB-AES-GCM defines the following security context parameters.

BCB-AES-GCM Security Parameters

Id	Name	Encoding Type	Default Value
1	Initialization Vector	byte string	NONE
2	Key Length	UINT	3
3	Encapsulation Key	Byte Array	NONE
4	AAD Scope Flags	UINT	0

Table 4

4.4. Results

The BCB-AES-GCM security context produces a single security result carried in the security block: the authentication tag.

NOTES:

The cipher text generated by the cipher suite is not considered a security result as it is stored in the block-type-specific data field of the security target block. When operating in GCM mode, AES produces cipher text of the same size as its plain text and, therefore, no additional logic is required to handle padding or overflow caused by the encryption in most cases (see below).

If the generated cipher text contains the authentication tag and the tag can be separated from the cipher text then the tag **MUST** be separated and stored in the Authentication Tag security result field.

If the generated cipher text contains the authentication tag and the tag cannot be separated from the cipher text then the tag **MUST NOT** be included in the Authentication tag security result field. Instead the security target block **MUST** be resized to accommodate the additional 128 bits of authentication tag included in the generated cipher text.

4.4.1. Authentication Tag

The authentication tag is generated by the cipher suite over the security target plain text input to the cipher suite as combined with any optional additional authenticated data. This tag is used to

ensure that the plain text (and important information associated with the plain text) is authenticated prior to decryption.

If the authentication tag is included in the cipher text placed in the security target block-type-specific data field, then this security result **MUST NOT** be included in the BCB for that security target.

The length of the authentication tag, prior to any CBOR encoding, **MUST** be 128 bits.

This value **MUST** be encoded as a CBOR byte string.

4.4.2. Enumerations

BCB-AES-GCM defines the following security context parameters.

BCB-AES-GCM Security Results

Result Id	Result Name	CBOR Encoding Type
1	Authentication Tag	byte string

Table 5

4.5. Key Considerations

BCB-AES-GCM does not define or otherwise mandate any method for key exchange, encryption, or encapsulation. The derivation of an appropriate key is considered separate from the application of the authenticated confidentiality service provided by this context.

Keys used with this context **MUST** be symmetric and **MUST** have a key length equal to the key length defined in the security context parameters or as defined by local security policy at security verifiers and acceptors.

It is assumed that any security verifier or security acceptor can determine the proper key to be used. Potential sources of the key include (but are not limited to) the following.

Pre-placed keys selected based on local policy.

Keys extracted from encapsulated key material carried in the BCB.

Session keys negotiated via a mechanism external to the BCB.

BCB-AES-GCM provides no explicit requirements on the configuration, storage, or exchange of keys.

4.6. Canonicalization Algorithms

This section defines the canonicalization algorithms used to prepare the inputs used to generate both the cipher text and the authentication tag.

In all cases, the canonical form of any portion of an extension block MUST be performed as described in [I-D.ietf-dtn-bpsec]. The canonicalization algorithms defined in [I-D.ietf-dtn-bpsec] adhere to the canonical forms for extension blocks defined in [I-D.ietf-dtn-bpbis] but resolve ambiguities related to how values are represented in CBOR.

4.6.1. Cipher text related calculations

The plain text used during encryption MUST be calculated as the single, definite-length CBOR byte string representing the block-type-specific data field of the security target excluding the CBOR byte string identifying byte and optional CBOR byte string length field.

For example, consider the following two CBOR byte strings and the plain text that would be extracted from them.

CBOR byte string Examples

CBOR Byte String (Hex)	CBOR Part (Hex)	Plain Text Part (Hex)
18ED	18	ED
C24CDEADBEEFDEADBEEFDEADBEEF	C24C	DEADBEEFDEADBEEFDEADBEEF

Table 6

Similarly, the cipher text used during decryption MUST be calculated as the single, definite-length CBOR byte string representing the block-type-specific data field excluding the CBOR byte string identifying byte and optional CBOR byte string length field.

All other fields of the security target (such as the block type code, block number, block processing control flags, or any CRC information) MUST NOT be considered as part of encryption or decryption.

4.6.2. Additional Authenticated Data

The construction of additional authenticated data depends on the AAD Scope flags that may be provided as part of customizing the behavior of this security context.

The canonical form of the AAD input to the BCB-AES-GCM mechanism is constructed using the following process. This process **MUST** be followed when generating AAD for either encryption or decryption.

1. The canonical form of the AAD starts as the empty set with length 0.
2. If the AAD Scope parameter is present and the Primary Block Flag is set to 1, then a canonical form of the bundle's primary block **MUST** be calculated and the result appended to the AAD.
3. If the AAD Scope parameter is present and the Security Header flag is set to 1, then the canonical form of the Block Type Code, Block Number, and Block Processing Control Flags associated with the BIB **MUST** be calculated and, in that order, appended to the AAD.
4. If the AAD Scope parameter is present and the Target Header flag is set to 1, then the canonical form of the Block Type Code, Block Number, and Block Processing Control Flags associated with the security target **MUST** be calculated and, in that order, appended to the AAD.

If, after this process, the AAD remains at length 0, then no AAD exists to be input to the cipher suite.

4.7. Processing

4.7.1. Encryption

During encryption, four inputs are prepared for input to the AES/GCM cipher: the encryption key, the Initialization Vector (IV), the security target plain text to be encrypted, and any additional authenticated data. These data items **MUST** be generated as follows.

The encryption key **MUST** have the appropriate length as required by local security policy. The key may be generated specifically for this encryption, given as part of local security policy, or through some other key management mechanism as discussed in Section 4.5.

The Initialization Vector (IV) selected MUST be of the appropriate length. Because replaying an IV in counter mode voids the confidentiality of all messages encrypted with said IV, this context also requires a unique IV for every encryption performed with the same key. This means the same key and IV combination MUST NOT be used more than once.

The security target plain text for encryption MUST be generated as discussed in Section 4.6.1.

Additional authenticated data, if present, MUST be generated as discussed in Section 4.6.2 with the value of AAD Scope flags being taken from local security policy.

Upon successful encryption the following actions MUST occur.

The cipher text produced by AES/GCM MUST replace the bytes used to define the plain text in the security target block's block-type-specific data field. The block length of the security target MUST be updated if the generated cipher text is larger than the plain text (which can occur when the authentication tag is included in the cipher text calculation, as discussed in Section 4.4).

The authentication tag calculated by the AES/GCM cipher MUST be added as a security result for the security target in the BCB holding results for this security operation.

Cases where the authentication tag is generated as part of the cipher text MUST be processed as described in Section 4.4.

Finally, the BCB containing information about this security operation MUST be updated as follows. These operations may occur in any order.

The security context ID for the BCB MUST be set to the context identifier for BCB-AES-GCM.

The IV input to the cipher MUST be added as the IV security parameter for the BCB.

Any local flags used to generate AAD for this cipher MUST be added as the AAD Scope flags security parameter for the BCB.

The encryption key MAY be encapsulated using some key encapsulation mechanism (to include encrypting with a key encryption key) and the results of the encapsulation added as the Encapsulated Key security parameter for the BCB.

The key length used by this security context MUST be added as the Key Length security parameter for the BCB if it differs from the default key length. Otherwise, the key length MAY be omitted if doing so provides a useful reduction in message sizes.

Problems encountered in the encryption MUST be processed in accordance with local security policy.

4.7.2. Decryption

During encryption, five inputs are prepared for input to the AES/GCM cipher: the decryption key, the Initialization Vector (IV), the security target cipher text to be decrypted, any additional authenticated data, and the authentication tag generated from the original encryption. These data items MUST be generated as follows.

The decryption key MUST be derived using the Encapsulated Key security parameter if such a parameter is included in the security context parameters of the BCB. Otherwise this key MUST be derived in accordance with security policy at the decrypting node as discussed in Section 4.5.

The Initialization Vector (IV) MUST be set to the value of the IV security parameter included in the BCB. If the IV parameter is not included as a security parameter, an IV MAY be derived from local security policy in cases where IVs are predictable (such as always using an IV of 0 with constantly differing keys). Alternatively, a lack of an IV security parameter MAY be treated as an error by the decrypting node.

The security target cipher text for decryption MUST be generated as discussed in Section 4.6.1.

Additional authenticated data, if present, MUST be generated as discussed in Section 4.6.2 with the value of AAD Scope flags being taken from the AAD Scope flags security context parameter. If the AAD Scope flags parameter is not included in the security context parameters then these flags MAY be derived from local security policy in cases where the set of such flags is determinable in the network.

The authentication tag MUST be present in the BCB security context parameters field if additional authenticated data are defined for the BCB (either in the AAD Scope flags parameter or as specified by local policy). This tag MUST be 128 bits in length.

Upon successful decryption the following actions MUST occur.

The plain text produced by AES/GCM MUST replace the bytes used to define the cipher text in the security target block's block-type-specific data field. Any changes to the security target block length field MUST be corrected in cases where the plain text has a different length than the replaced cipher text.

If the cipher text fails to authenticate, if any needed parameters are missing, or if there are other problems in the decryption then the decryption MUST be treated as failed and processed in accordance with local security policy.

5. IANA Considerations

5.1. Security Context Identifiers

This specification allocates two security context identifiers from the "BPsec Security Context Identifier" registry defined in [I-D.ietf-dtn-bpsec].

Additional Entries for the BPsec Security Context Identifiers Registry:

Value	Description	Reference
TBA	BIB-HMAC-SHA2	This document
TBA	BCB-AES-GCM	This document

Table 7

6. Normative References

- [AES-GCM] Dworkin, M., "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.", November 2007.
- [I-D.ietf-dtn-bpbis] Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-26 (work in progress), July 2020.
- [I-D.ietf-dtn-bpsec] Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-23 (work in progress), October 2020.

- [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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4634] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, DOI 10.17487/RFC4634, July 2006, <<https://www.rfc-editor.org/info/rfc4634>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.

Appendix A. Acknowledgements

The following participants contributed useful review and analysis of these security contexts: Amy Alford and Sarah Heiner of the Johns Hopkins University Applied Physics Laboratory.

Author's Address

Edward J. Birrane, III
The Johns Hopkins University Applied
Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
US

Phone: +1 443 778 7423
Email: Edward.Birrane@jhuapl.edu

Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: April 29, 2021

B. Sipos
RKF Engineering
M. Demmer
UC Berkeley
J. Ott
Aalto University
S. Perreault
October 26, 2020

Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4
draft-ietf-dtn-tcpclv4-22

Abstract

This document describes a TCP-based convergence layer (TCPCL) for Delay-Tolerant Networking (DTN). This version of the TCPCL protocol resolves implementation issues in the earlier TCPCL Version 3 of RFC7242 and updates to the Bundle Protocol (BP) contents, encodings, and convergence layer requirements in BP Version 7. Specifically, the TCPCLv4 uses CBOR-encoded BPv7 bundles as its service data unit being transported and provides a reliable transport of such bundles. This version of TCPCL also includes security and extensibility mechanisms.

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 April 29, 2021.

Copyright Notice

Copyright (c) 2020 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.

Table of Contents

1. Introduction	3
1.1. Scope	4
2. Requirements Language	5
2.1. Definitions Specific to the TCPCL Protocol	5
3. General Protocol Description	9
3.1. Convergence Layer Services	9
3.2. TCPCL Session Overview	11
3.3. TCPCL States and Transitions	13
3.4. PKIX Environments and CA Policy	19
3.5. Session Keeping Policies	20
3.6. Transfer Segmentation Policies	21
3.7. Example Message Exchange	22
4. Session Establishment	23
4.1. TCP Connection	24
4.2. Contact Header	25
4.3. Contact Validation and Negotiation	26
4.4. Session Security	27
4.4.1. Entity Identification	28
4.4.2. TLS Handshake	29
4.4.3. TLS Authentication	30
4.4.4. Example TLS Initiation	32
4.5. Message Header	33
4.6. Session Initialization Message (SESS_INIT)	34
4.7. Session Parameter Negotiation	36
4.8. Session Extension Items	37
5. Established Session Operation	38
5.1. Upkeep and Status Messages	38
5.1.1. Session Upkeep (KEEPALIVE)	38
5.1.2. Message Rejection (MSG_REJECT)	39
5.2. Bundle Transfer	40
5.2.1. Bundle Transfer ID	41
5.2.2. Data Transmission (XFER_SEGMENT)	41
5.2.3. Data Acknowledgments (XFER_ACK)	43
5.2.4. Transfer Refusal (XFER_REFUSE)	44
5.2.5. Transfer Extension Items	47
6. Session Termination	49

6.1.	Session Termination Message (SESS_TERM)	49
6.2.	Idle Session Shutdown	52
7.	Implementation Status	52
8.	Security Considerations	52
8.1.	Threat: Passive Leak of Node Data	53
8.2.	Threat: Passive Leak of Bundle Data	53
8.3.	Threat: TCPCL Version Downgrade	53
8.4.	Threat: Transport Security Stripping	53
8.5.	Threat: Weak TLS Configurations	54
8.6.	Threat: Certificate Validation Vulnerabilities	54
8.7.	Threat: Symmetric Key Limits	54
8.8.	Threat: BP Node Impersonation	54
8.9.	Threat: Denial of Service	55
8.10.	Alternate Uses of TLS	56
8.10.1.	TLS Without Authentication	56
8.10.2.	Non-Certificate TLS Use	56
8.11.	Predictability of Transfer IDs	56
9.	IANA Considerations	57
9.1.	Port Number	57
9.2.	Protocol Versions	57
9.3.	Session Extension Types	58
9.4.	Transfer Extension Types	59
9.5.	Message Types	60
9.6.	XFER_REFUSE Reason Codes	61
9.7.	SESS_TERM Reason Codes	62
9.8.	MSG_REJECT Reason Codes	63
10.	Acknowledgments	64
11.	References	64
11.1.	Normative References	64
11.2.	Informative References	66
	Appendix A. Significant changes from RFC7242	67
	Authors' Addresses	69

1. Introduction

This document describes the TCP-based convergence-layer protocol for Delay-Tolerant Networking. Delay-Tolerant Networking is an end-to-end architecture providing communications in and/or through highly stressed environments, including those with intermittent connectivity, long and/or variable delays, and high bit error rates. More detailed descriptions of the rationale and capabilities of these networks can be found in "Delay-Tolerant Network Architecture" [RFC4838].

An important goal of the DTN architecture is to accommodate a wide range of networking technologies and environments. The protocol used for DTN communications is the Bundle Protocol Version 7 (BPv7) [I-D.ietf-dtn-bpbis], an application-layer protocol that is used to

construct a store-and-forward overlay network. BPv7 requires the services of a "convergence-layer adapter" (CLA) to send and receive bundles using the service of some "native" link, network, or Internet protocol. This document describes one such convergence-layer adapter that uses the well-known Transmission Control Protocol (TCP). This convergence layer is referred to as TCP Convergence Layer Version 4 (TCPCLv4). For the remainder of this document, the abbreviation "BP" without the version suffix refers to BPv7. For the remainder of this document, the abbreviation "TCPCL" without the version suffix refers to TCPCLv4.

The locations of the TCPCL and the BP in the Internet model protocol stack (described in [RFC1122]) are shown in Figure 1. In particular, when BP is using TCP as its bearer with TCPCL as its convergence layer, both BP and TCPCL reside at the application layer of the Internet model.

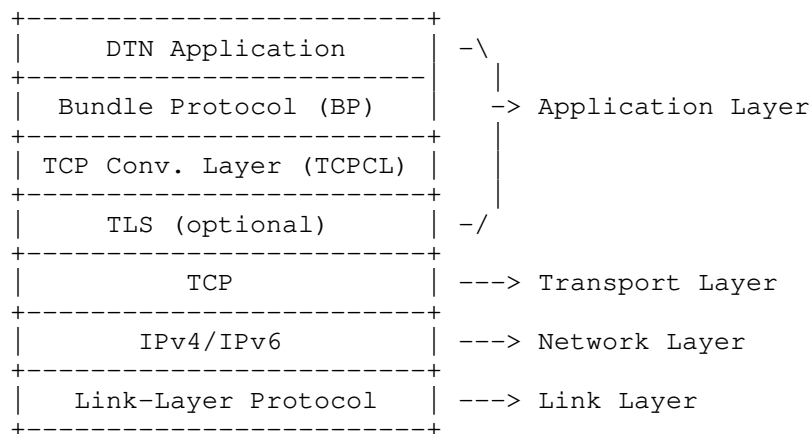


Figure 1: The Locations of the Bundle Protocol and the TCP Convergence-Layer Protocol above the Internet Protocol Stack

1.1. Scope

This document describes the format of the protocol data units passed between entities participating in TCPCL communications. This document does not address:

- o The format of protocol data units of the Bundle Protocol, as those are defined elsewhere in [I-D.ietf-dtn-bpbis]. This includes the concept of bundle fragmentation or bundle encapsulation. The TCPCL transfers bundles as opaque data blocks.

- o Mechanisms for locating or identifying other bundle entities (peers) within a network or across an internet. The mapping of Node ID to potential convergence layer (CL) protocol and network address is left to implementation and configuration of the BP Agent and its various potential routing strategies.
- o Logic for routing bundles along a path toward a bundle's endpoint. This CL protocol is involved only in transporting bundles between adjacent nodes in a routing sequence.
- o Policies or mechanisms for issuing Public Key Infrastructure Using X.509 (PKIX) certificates; provisioning, deploying, or accessing certificates and private keys; deploying or accessing certificate revocation lists (CRLs); or configuring security parameters on an individual entity or across a network.
- o Uses of TLS which are not based on PKIX certificate authentication (see Section 8.10.2) or in which authentication of both entities is not possible (see Section 8.10.1).

Any TCPCL implementation requires a BP agent to perform those above listed functions in order to perform end-to-end bundle delivery.

2. 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.

2.1. Definitions Specific to the TCPCL Protocol

This section contains definitions specific to the TCPCL protocol.

Network Byte Order: Most significant byte first, a.k.a., big endian. All of the integer encodings in this protocol SHALL be transmitted in network byte order.

TCPCL Entity: This is the notional TCPCL application that initiates TCPCL sessions. This design, implementation, configuration, and specific behavior of such an entity is outside of the scope of this document. However, the concept of an entity has utility within the scope of this document as the container and initiator of TCPCL sessions. The relationship between a TCPCL entity and TCPCL sessions is defined as follows:

- * A TCPCL Entity MAY actively initiate any number of TCPCL Sessions and should do so whenever the entity is the initial transmitter of information to another entity in the network.
- * A TCPCL Entity MAY support zero or more passive listening elements that listen for connection requests from other TCPCL Entities operating on other entities in the network.
- * A TCPCL Entity MAY passively initiate any number of TCPCL Sessions from requests received by its passive listening element(s) if the entity uses such elements.

These relationships are illustrated in Figure 2. For most TCPCL behavior within a session, the two entities are symmetric and there is no protocol distinction between them. Some specific behavior, particularly during session establishment, distinguishes between the active entity and the passive entity. For the remainder of this document, the term "entity" without the prefix "TCPCL" refers to a TCPCL entity.

TCP Connection: The term Connection in this specification exclusively refers to a TCP connection and any and all behaviors, sessions, and other states associated with that TCP connection.

TCPCL Session: A TCPCL session (as opposed to a TCP connection) is a TCPCL communication relationship between two TCPCL entities. A TCPCL session operates within a single underlying TCP connection and the lifetime of a TCPCL session is bound to the lifetime of that TCP connection. A TCPCL session is terminated when the TCP connection ends, due either to one or both entities actively closing the TCP connection or due to network errors causing a failure of the TCP connection. Within a single TCPCL session there are two possible transfer streams; one in each direction, with one stream from each entity being the outbound stream and the other being the inbound stream (see Figure 3). From the perspective of a TCPCL session, the two transfer streams do not logically interact with each other. The streams do operate over the same TCP connection and between the same BP agents, so there are logical relationships at those layers (message and bundle interleaving respectively). For the remainder of this document, the term "session" without the prefix "TCPCL" refers to a TCPCL session.

Session parameters: These are a set of values used to affect the operation of the TCPCL for a given session. The manner in which these parameters are conveyed to the bundle entity and thereby to the TCPCL is implementation dependent. However, the mechanism by

which two entities exchange and negotiate the values to be used for a given session is described in Section 4.3.

Transfer Stream: A Transfer stream is a uni-directional user-data path within a TCPCL Session. Transfers sent over a transfer stream are serialized, meaning that one transfer must complete its transmission prior to another transfer being started over the same transfer stream. At the stream layer there is no logical relationship between transfers in that stream; it's only within the BP agent that transfers are fully decoded as bundles. Each uni-directional stream has a single sender entity and a single receiver entity.

Transfer: This refers to the procedures and mechanisms for conveyance of an individual bundle from one node to another. Each transfer within TCPCL is identified by a Transfer ID number which is guaranteed to be unique only to a single direction within a single Session.

Transfer Segment: A subset of a transfer of user data being communicated over a transfer stream.

Idle Session: A TCPCL session is idle while there is no transmission in-progress in either direction. While idle, the only messages being transmitted or received are KEEPALIVE messages.

Live Session: A TCPCL session is live while there is a transmission in-progress in either direction.

Reason Codes: The TCPCL uses numeric codes to encode specific reasons for individual failure/error message types.

The relationship between connections, sessions, and streams is shown in Figure 3.

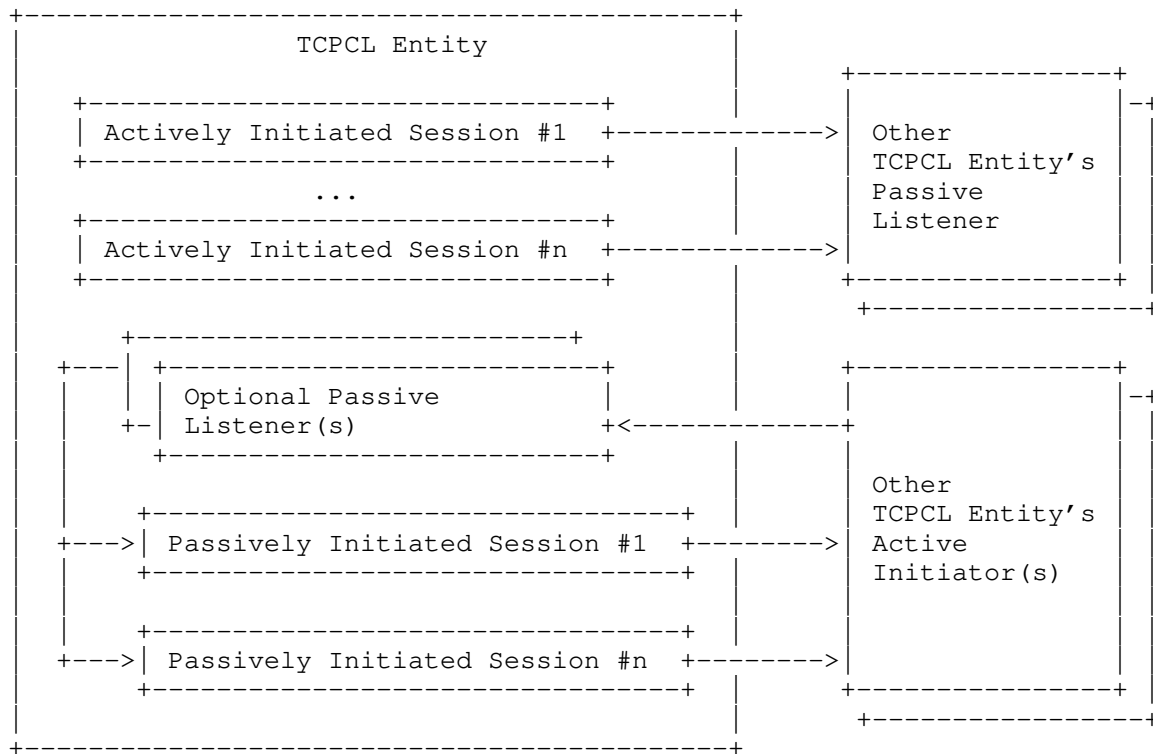


Figure 2: The relationships between TCPCL entities

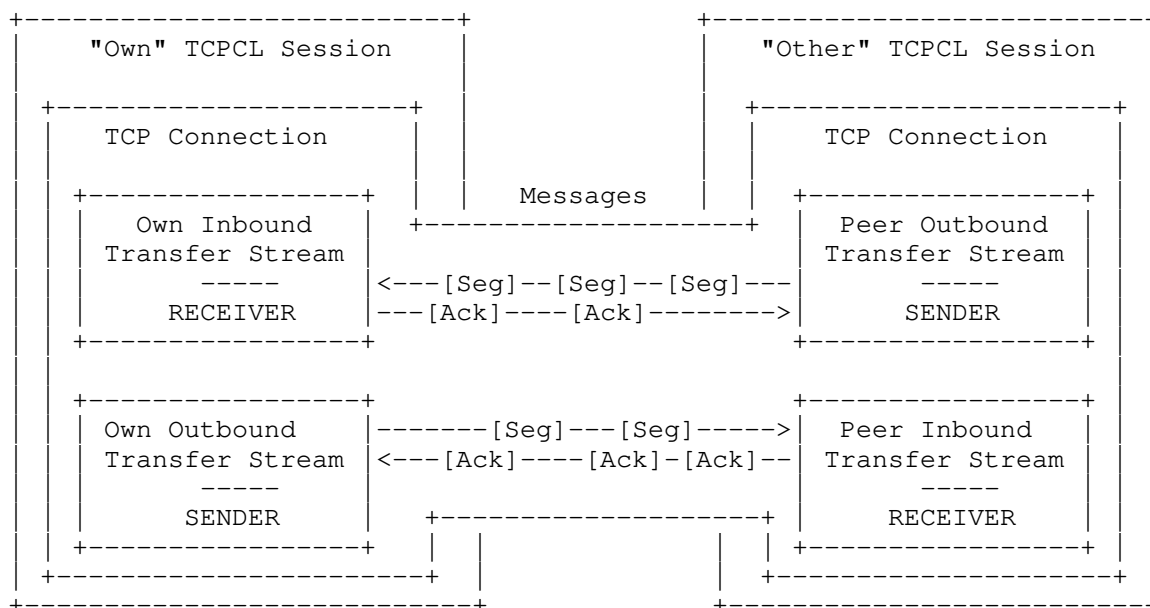


Figure 3: The relationship within a TCPCL Session of its two streams

3. General Protocol Description

The service of this protocol is the transmission of DTN bundles via the Transmission Control Protocol (TCP). This document specifies the encapsulation of bundles, procedures for TCP setup and teardown, and a set of messages and node requirements. The general operation of the protocol is as follows.

3.1. Convergence Layer Services

This version of the TCPCL provides the following services to support the overlaying Bundle Protocol agent. In all cases, this is not an API definition but a logical description of how the CL can interact with the BP agent. Each of these interactions can be associated with any number of additional metadata items as necessary to support the operation of the CL or BP agent.

Attempt Session: The TCPCL allows a BP agent to preemptively attempt to establish a TCPCL session with a peer entity. Each session attempt can send a different set of session negotiation parameters as directed by the BP agent.

Terminate Session: The TCPCL allows a BP agent to preemptively terminate an established TCPCL session with a peer entity. The terminate request is on a per-session basis.

Session State Changed: The TCPCL entity indicates to the BP agent when the session state changes. The top-level session states indicated are:

Connecting: A TCP connection is being established. This state only applies to the active entity.

Contact Negotiating: A TCP connection has been made (as either active or passive entity) and contact negotiation has begun.

Session Negotiating: Contact negotiation has been completed (including possible TLS use) and session negotiation has begun.

Established: The session has been fully established and is ready for its first transfer.

Ending: The entity sent SESS_TERM message and is in the ending state.

Terminated: The session has finished normal termination sequencing.

Failed: The session ended without normal termination sequencing.

Session Idle Changed: The TCPCL entity indicates to the BP agent when the live/idle sub-state of the session changes. This occurs only when the top-level session state is "Established". The session transitions from Idle to Live at the start of a transfer in either transfer stream; the session transitions from Live to Idle at the end of a transfer when the other transfer stream does not have an ongoing transfer. Because TCPCL transmits serially over a TCP connection it suffers from "head of queue blocking," so a transfer in either direction can block an immediate start of a new transfer in the session.

Begin Transmission: The principal purpose of the TCPCL is to allow a BP agent to transmit bundle data over an established TCPCL session. Transmission request is on a per-session basis and the CL does not necessarily perform any per-session or inter-session queueing. Any queueing of transmissions is the obligation of the BP agent.

Transmission Success: The TCPCL entity indicates to the BP agent when a bundle has been fully transferred to a peer entity.

Transmission Intermediate Progress: The TCPCL entity indicates to the BP agent on intermediate progress of transfer to a peer entity. This intermediate progress is at the granularity of each transferred segment.

Transmission Failure: The TCPCL entity indicates to the BP agent on certain reasons for bundle transmission failure, notably when the peer entity rejects the bundle or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

Reception Initialized: The TCPCL entity indicates to the receiving BP agent just before any transmission data is sent. This corresponds to reception of the XFER_SEGMENT message with the START flag of 1.

Interrupt Reception: The TCPCL entity allows a BP agent to interrupt an individual transfer before it has fully completed (successfully or not). Interruption can occur any time after the reception is initialized.

Reception Success: The TCPCL entity indicates to the BP agent when a bundle has been fully transferred from a peer entity.

Reception Intermediate Progress: The TCPCL entity indicates to the BP agent on intermediate progress of transfer from the peer entity. This intermediate progress is at the granularity of each transferred segment. Intermediate reception indication allows a BP agent the chance to inspect bundle header contents before the entire bundle is available, and thus supports the "Reception Interruption" capability.

Reception Failure: The TCPCL entity indicates to the BP agent on certain reasons for reception failure, notably when the local entity rejects an attempted transfer for some local policy reason or when a TCPCL session ends before transfer success. The TCPCL itself does not have a notion of transfer timeout.

3.2. TCPCL Session Overview

First, one node establishes a TCPCL session to the other by initiating a TCP connection in accordance with [RFC0793]. After setup of the TCP connection is complete, an initial Contact Header is exchanged in both directions to establish a shared TCPCL version and negotiate the use of TLS security (as described in Section 4). Once contact negotiation is complete, TCPCL messaging is available and the session negotiation is used to set parameters of the TCPCL session. One of these parameters is a Node ID that each TCPCL Entity is acting

as. This is used to assist in routing and forwarding messages by the BP Agent and is part of the authentication capability provided by TLS.

Once negotiated, the parameters of a TCPCL session cannot change and if there is a desire by either peer to transfer data under different parameters then a new session must be established. This makes CL logic simpler but relies on the assumption that establishing a TCP connection is lightweight enough that TCP connection overhead is negligible compared to TCPCL data sizes.

Once the TCPCL session is established and configured in this way, bundles can be transferred in either direction. Each transfer is performed by segmenting the transfer data into one or more XFER_SEGMENT messages. Multiple bundles can be transmitted consecutively in a single direction on a single TCPCL connection. Segments from different bundles are never interleaved. Bundle interleaving can be accomplished by fragmentation at the BP layer or by establishing multiple TCPCL sessions between the same peers. There is no fundamental limit on the number of TCPCL sessions which a single node can establish beyond the limit imposed by the number of available (ephemeral) TCP ports of the active entity.

A feature of this protocol is for the receiving node to send acknowledgment (XFER_ACK) messages as bundle data segments arrive. The rationale behind these acknowledgments is to enable the sender node to determine how much of the bundle has been received, so that in case the session is interrupted, it can perform reactive fragmentation to avoid re-sending the already transmitted part of the bundle. In addition, there is no explicit flow control on the TCPCL layer.

A TCPCL receiver can interrupt the transmission of a bundle at any point in time by replying with a XFER_REFUSE message, which causes the sender to stop transmission of the associated bundle (if it hasn't already finished transmission) Note: This enables a cross-layer optimization in that it allows a receiver that detects that it already has received a certain bundle to interrupt transmission as early as possible and thus save transmission capacity for other bundles.

For sessions that are idle, a KEEPALIVE message is sent at a negotiated interval. This is used to convey node live-ness information during otherwise message-less time intervals.

A SESS_TERM message is used to initiate the ending of a TCPCL session (see Section 6.1). During termination sequencing, in-progress transfers can be completed but no new transfers can be initiated. A

SESS_TERM message can also be used to refuse a session setup by a peer (see Section 4.3). Regardless of the reason, session termination is initiated by one of the entities and responded-to by the other as illustrated by Figure 13 and Figure 14. Even when there are no transfers queued or in-progress, the session termination procedure allows each entity to distinguish between a clean end to a session and the TCP connection being closed because of some underlying network issue.

Once a session is established, TCPCL is a symmetric protocol between the peers. Both sides can start sending data segments in a session, and one side's bundle transfer does not have to complete before the other side can start sending data segments on its own. Hence, the protocol allows for a bi-directional mode of communication. Note that in the case of concurrent bidirectional transmission, acknowledgment segments MAY be interleaved with data segments.

3.3. TCPCL States and Transitions

The states of a normal TCPCL session (i.e., without session failures) are indicated in Figure 4.

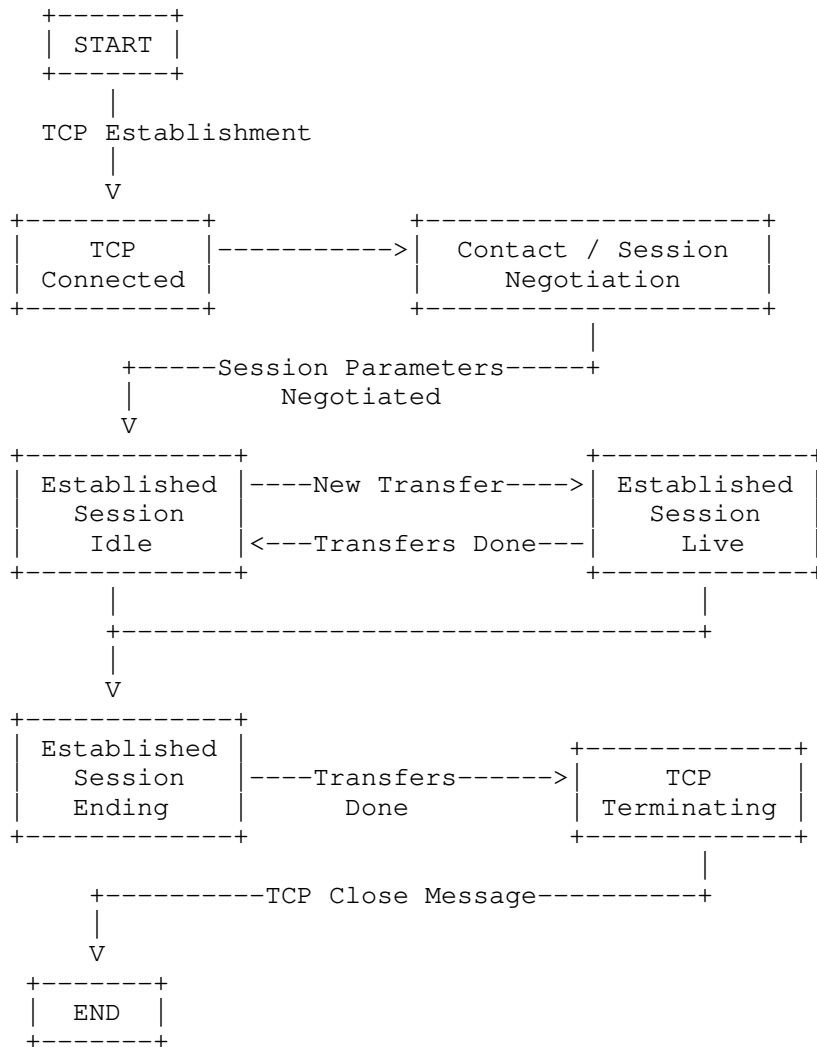


Figure 4: Top-level states of a TCPCL session

Notes on Established Session states:

Session "Live" means transmitting or receiving over a transfer stream.

Session "Idle" means no transmission/reception over a transfer stream.

Session "Ending" means no new transfers will be allowed.

Contact negotiation involves exchanging a Contact Header (CH) in both directions and deriving a negotiated state from the two headers. The contact negotiation sequencing is performed either as the active or passive entity, and is illustrated in Figure 5 and Figure 6 respectively which both share the data validation and negotiation of the Processing of Contact Header "[PCH]" activity of Figure 7 and the "[TCPCLOSE]" activity which indicates TCP connection close. Successful negotiation results in one of the Session Initiation "[SI]" activities being performed. To avoid data loss, a Session Termination "[ST]" exchange allows cleanly finishing transfers before a session is ended.

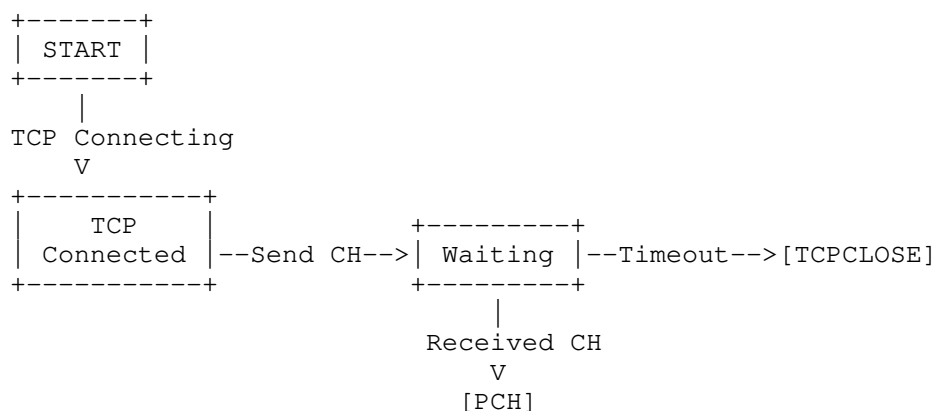


Figure 5: Contact Initiation as Active Entity

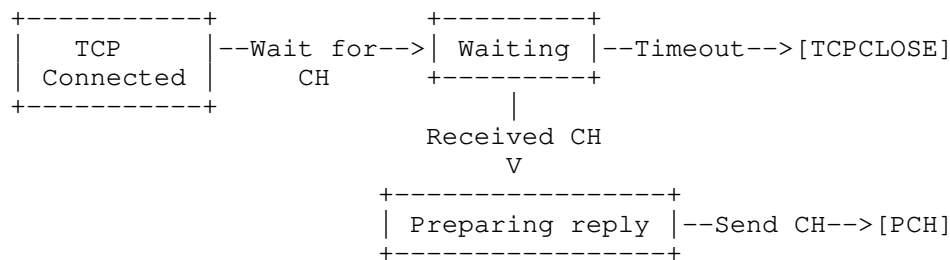


Figure 6: Contact Initiation as Passive Entity

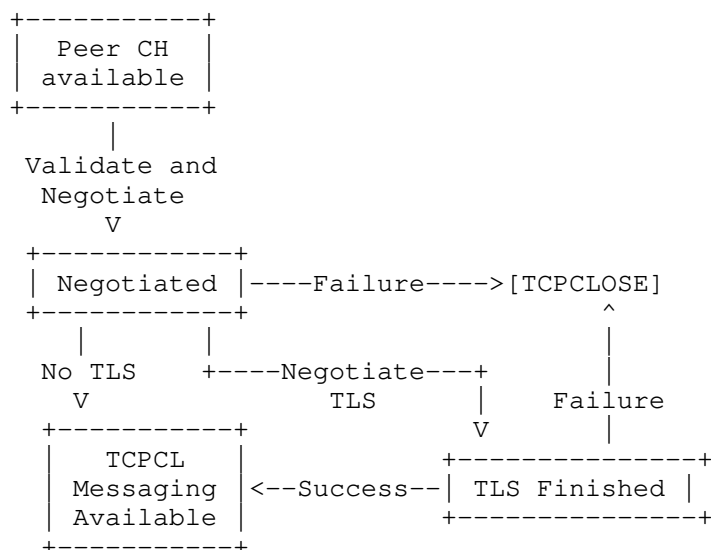


Figure 7: Processing of Contact Header [PCH]

Session negotiation involves exchanging a session initialization (SESS_INIT) message in both directions and deriving a negotiated state from the two messages. The session negotiation sequencing is performed either as the active or passive entity, and is illustrated in Figure 8 and Figure 9 respectively which both share the data validation and negotiation of Figure 10. The validation here includes certificate validation and authentication when TLS is used for the session.

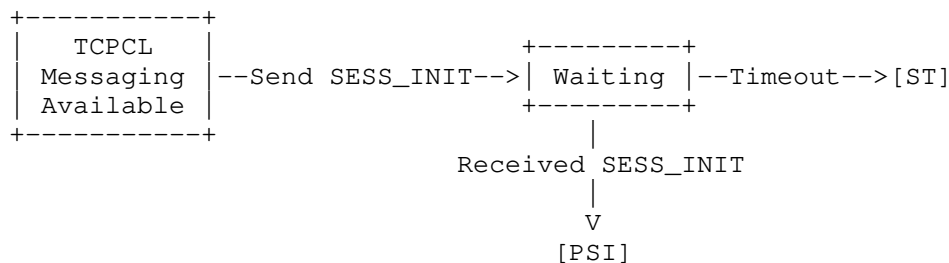


Figure 8: Session Initiation [SI] as Active Entity

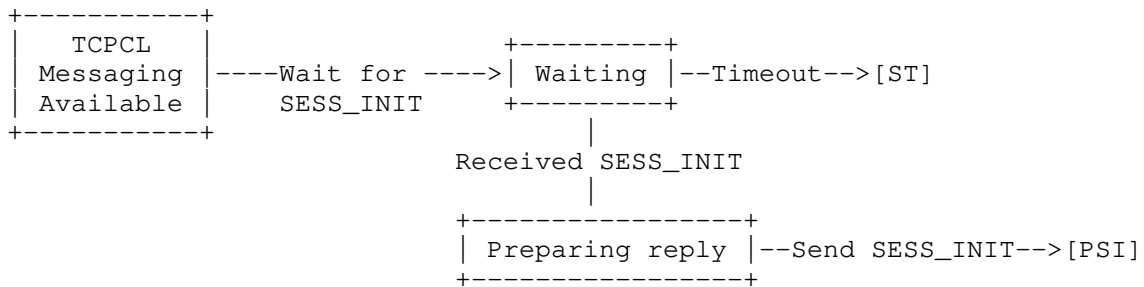


Figure 9: Session Initiation [SI] as Passive Entity

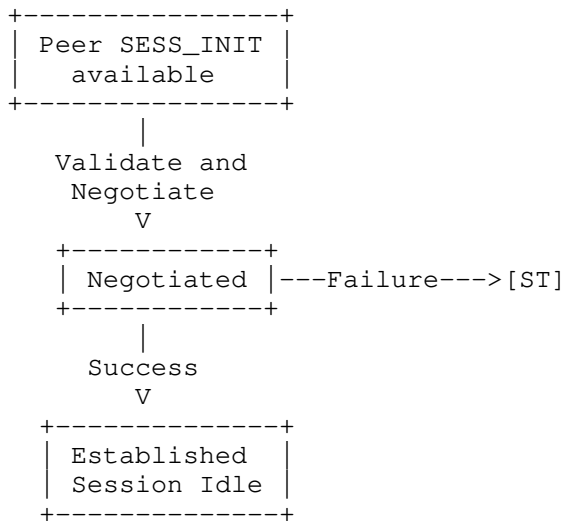


Figure 10: Processing of Session Initiation [PSI]

Transfers can occur after a session is established and it's not in the Ending state. Each transfer occurs within a single logical transfer stream between a sender and a receiver, as illustrated in Figure 11 and Figure 12 respectively.

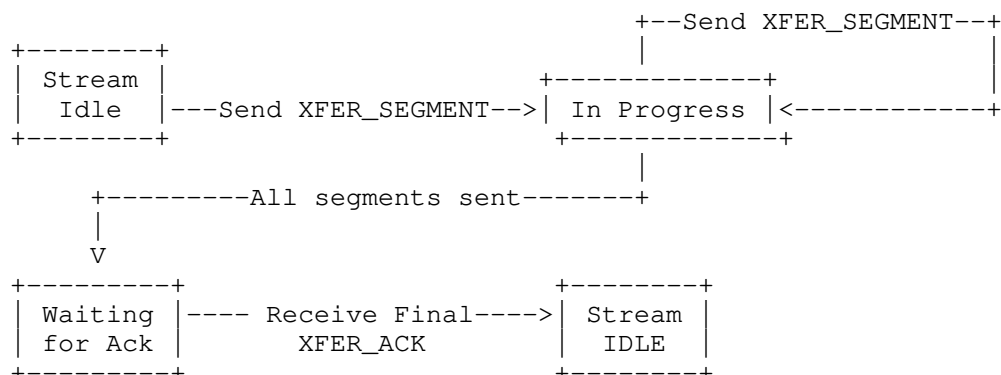


Figure 11: Transfer sender states

Notes on transfer sending:

Pipelining of transfers can occur when the sending entity begins a new transfer while in the "Waiting for Ack" state.

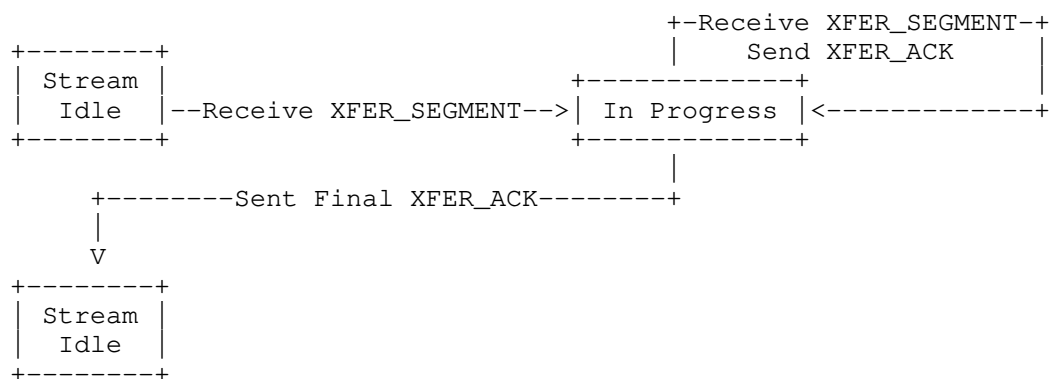


Figure 12: Transfer receiver states

Session termination involves one entity initiating the termination of the session and the other entity acknowledging the termination. For either entity, it is the sending of the SESS_TERM message which transitions the session to the Ending substate. While a session is in the Ending state only in-progress transfers can be completed and no new transfers can be started.

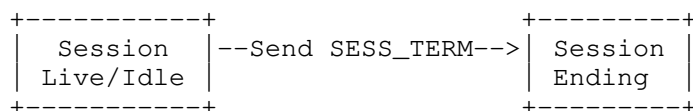


Figure 13: Session Termination [ST] from the Initiator

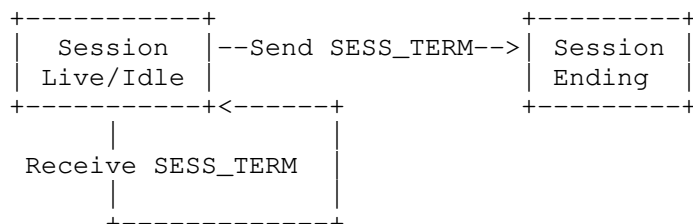


Figure 14: Session Termination [ST] from the Responder

3.4. PKIX Environments and CA Policy

This specification gives requirements about how to use PKIX certificates issued by a Certificate Authority (CA), but does not define any mechanisms for how those certificates come to be. The requirements about TCPCL certificate use are broad to support two quite different PKIX environments:

DTN-Aware CAs: In the ideal case, the CA(s) issuing certificates for TCPCL entities are aware of the end use of the certificate, have a mechanism for verifying ownership of a Node ID, and are issuing certificates directly for that Node ID. In this environment, the ability to authenticate a peer entity Node ID directly avoids the need to authenticate a network name or address and then implicitly trust Node ID of the peer. The TCPCL authenticates the Node ID whenever possible and this is preferred over lower-level PKIX identifiers.

DTN-Ignorant CAs: It is expected that Internet-scale "public" CAs will continue to focus on DNS names as the preferred PKIX identifier. There are large infrastructures already in-place for managing network-level authentication and protocols to manage identity verification in those environments [RFC8555]. The TCPCL allows for this type of environment by authenticating a lower-level identifier for a peer and requiring the entity to trust that the Node ID given by the peer (during session initialization) is valid. This situation not ideal, as it allows vulnerabilities described in Section 8.8, but still provides some amount of mutual authentication to take place for a TCPCL session.

Even within a single TCPCL session, each entity may operate within different PKI environments and with different identifier limitations. The requirements related to identifiers in a PKIX certificate are in Section 4.4.1.

It is important for interoperability that a TCPCL entity have its own security policy tailored to accommodate the peers with which it is expected to operate. A strict TLS security policy is appropriate for a private network with a single shared CA. Operation on the Internet (such as inter-site BP gateways) could trade more lax TCPCL security with the use of encrypted bundle encapsulation [I-D.ietf-dtn-bibect] to ensure strong bundle security.

3.5. Session Keeping Policies

This specification gives requirements about how to initiate, sustain, and terminate a TCPCL session but does not impose any requirements on how sessions need to be managed by a BP agent. It is a network administration matter to determine an appropriate session keeping policy, but guidance given here can be used to steer policy toward performance goals.

Persistent Session: This policy preemptively establishes a single session to known entities in the network and keeps the session active using KEEPALIVES. Benefits of this policy include reducing the total amount of TCP data needing to be exchanged for a set of transfers (assuming KEEPALIVE size is significantly smaller than transfer size), and allowing the session state to indicate peer connectivity. Drawbacks include wasted network resources when a session is mostly idle or when the network connectivity is inconsistent (which requires re-establishing failed sessions), and potential queueing issues when multiple transfers are requested simultaneously. This policy assumes that there is agreement between pairs of entities as to which of the peers will initiate sessions; if there is no such agreement, there is potential for duplicate sessions to be established between peers.

Ephemeral Sessions: This policy only establishes a session when an outgoing transfer is needed to be sent. Benefits of this policy include not wasting network resources on sessions which are idle for long periods of time, and avoids queueing issues of a persistent session. Drawbacks include the TCP and TLS overhead of establish a new session for each transfer. This policy assumes that each entity can function in a passive role to listen for session requests from any peer which needs to send a transfer; when that is not the case the Polling behavior below needs to happen. This policy can be augmented to keep the session established as long as any transfers are queued.

Active-Only Polling Sessions: When naming and/or addressing of one entity is variable (i.e. dynamically assigned IP address or domain name) or when firewall or routing rules prevent incoming TCP connections, that entity can only function in the active role. In these cases, sessions also need to be established when an incoming transfer is expected from a peer or based on a periodic schedule. This polling behavior causes inefficiencies compared to as-needed ephemeral sessions.

Many other policies can be established in a TCPCL network between the two extremes of single persistent sessions and only ephemeral sessions. Different policies can be applied to each peer entity and to each bundle as it needs to be transferred (e.g for quality of service). Additionally, future session extension types can apply further nuance to session policies and policy negotiation.

3.6. Transfer Segmentation Policies

Each TCPCL session allows a negotiated transfer segmentation policy to be applied in each transfer direction. A receiving node can set the Segment MRU in its SESS_INIT message to determine the largest acceptable segment size, and a transmitting node can segment a transfer into any sizes smaller than the receiver's Segment MRU. It is a network administration matter to determine an appropriate segmentation policy for entities operating TCPCL, but guidance given here can be used to steer policy toward performance goals. It is also advised to consider the Segment MRU in relation to chunking/packetization performed by TLS, TCP, and any intermediate network-layer nodes.

Minimum Overhead: For a simple network expected to exchange relatively small bundles, the Segment MRU can be set to be identical to the Transfer MRU which indicates that all transfers can be sent with a single data segment (i.e., no actual segmentation). If the network is closed and all transmitters are known to follow a single-segment transfer policy, then receivers can avoid the necessity of segment reassembly. Because this CL operates over a TCP stream, which suffers from a form of head-of-queue blocking between messages, while one node is transmitting a single XFER_SEGMENT message it is not able to transmit any XFER_ACK or XFER_REFUSE for any associated received transfers.

Predictable Message Sizing: In situations where the maximum message size is desired to be well-controlled, the Segment MRU can be set to the largest acceptable size (the message size less XFER_SEGMENT header size) and transmitters can always segment a transfer into maximum-size chunks no larger than the Segment MRU. This guarantees that any single XFER_SEGMENT will not monopolize the

TCP stream for too long, which would prevent outgoing XFER_ACK and XFER_REFUSE associated with received transfers.

Dynamic Segmentation: Even after negotiation of a Segment MRU for each receiving node, the actual transfer segmentation only needs to guarantee that any individual segment is no larger than that MRU. In a situation where TCP throughput is dynamic, the transfer segmentation size can also be dynamic in order to control message transmission duration.

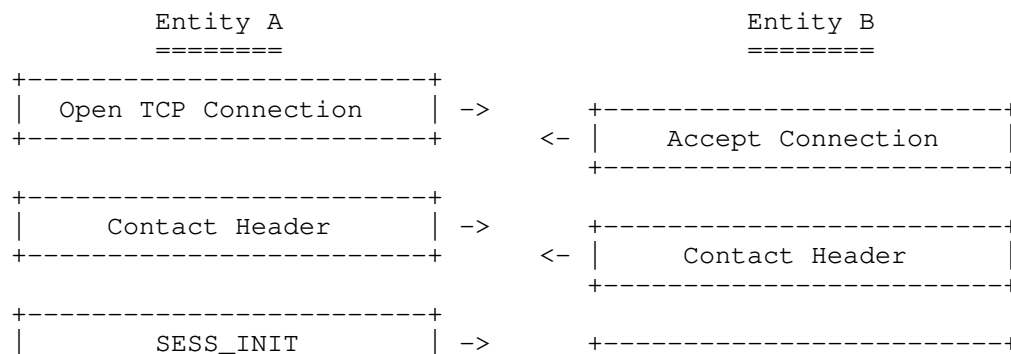
Many other policies can be established in a TCPCL network between the two extremes of minimum overhead (large MRU, single-segment) and predictable message sizing (small MRU, highly segmented). Different policies can be applied to each transfer stream to and from any particular node. Additionally, future session extension and transfer extension types can apply further nuance to transfer policies and policy negotiation.

3.7. Example Message Exchange

The following figure depicts the protocol exchange for a simple session, showing the session establishment and the transmission of a single bundle split into three data segments (of lengths "L1", "L2", and "L3") from Entity A to Entity B.

Note that the sending node can transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream. Although this example only demonstrates a single bundle transmission, it is also possible to pipeline multiple XFER_SEGMENT messages for different bundles without necessarily waiting for XFER_ACK messages to be returned for each one. However, interleaving data segments from different bundles is not allowed.

No errors or rejections are shown in this example.



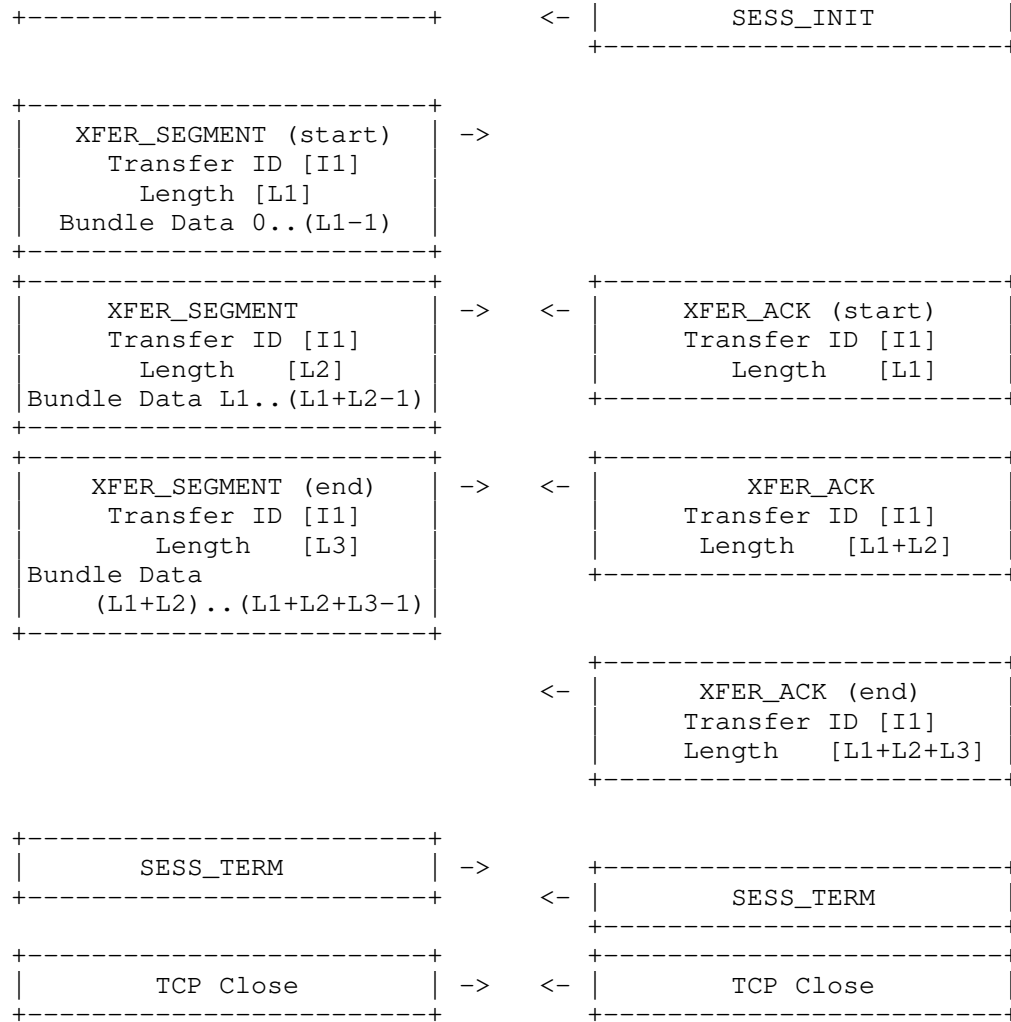


Figure 15: An example of the flow of protocol messages on a single TCP Session between two entities

4. Session Establishment

For bundle transmissions to occur using the TCPCL, a TCPCL session MUST first be established between communicating entities. It is up to the implementation to decide how and when session setup is triggered. For example, some sessions can be opened proactively and maintained for as long as is possible given the network conditions, while other sessions are be opened only when there is a bundle that

is queued for transmission and the routing algorithm selects a certain next-hop node.

4.1. TCP Connection

To establish a TCPCL session, an entity **MUST** first establish a TCP connection with the intended peer entity, typically by using the services provided by the operating system. Destination port number 4556 has been assigned by IANA as the Registered Port number for the TCP convergence layer. Other destination port numbers **MAY** be used per local configuration. Determining a peer's destination port number (if different from the registered TCPCL port number) is up to the implementation. Any source port number **MAY** be used for TCPCL sessions. Typically an operating system assigned number in the TCP Ephemeral range (49152-65535) is used.

If the entity is unable to establish a TCP connection for any reason, then it is an implementation matter to determine how to handle the connection failure. An entity **MAY** decide to re-attempt to establish the connection. If it does so, it **MUST NOT** overwhelm its target with repeated connection attempts. Therefore, the entity **MUST NOT** retry the connection setup earlier than some delay time from the last attempt, and it **SHOULD** use a (binary) exponential back-off mechanism to increase this delay in case of repeated failures. The upper limit on a re-attempt back-off is implementation defined but **SHOULD** be no longer than one minute (60 seconds) before signaling to the BP agent that a connection cannot be made.

Once a TCP connection is established, the active entity **SHALL** immediately transmit its Contact Header. Once a TCP connection is established, the passive entity **SHALL** wait for the peer's Contact Header. If the passive entity does not receive a Contact Header after some implementation-defined time duration after TCP connection is established, the entity **SHALL** close the TCP connection. Entities **SHOULD** choose a Contact Header reception timeout interval no longer than one minute (60 seconds). Upon reception of a Contact Header, the passive entity **SHALL** transmit its Contact Header. The ordering of the Contact Header exchange allows the passive entity to avoid allocating resources to a potential TCPCL session until after a valid Contact Header has been received from the active entity. This ordering also allows the passive peer to adapt to alternate TCPCL protocol versions.

The format of the Contact Header is described in Section 4.2. Because the TCPCL protocol version in use is part of the initial Contact Header, nodes using TCPCL version 4 can coexist on a network with nodes using earlier TCPCL versions (with some negotiation needed for interoperation as described in Section 4.3).

4.2. Contact Header

This section describes the format of the Contact Header and the meaning of its fields.

If an entity is capable of exchanging messages according to TLS 1.3 [RFC8446] or any successors which are compatible with that TLS ClientHello, the the CAN_TLS flag within its Contact Header SHALL be set to 1. This behavior prefers the use of TLS when possible, even if security policy does not allow or require authentication. This follows the opportunistic security model of [RFC7435], though an active attacker could interfere with the exchange in such cases.

Upon receipt of the Contact Header, both entities perform the validation and negotiation procedures defined in Section 4.3. After receiving the Contact Header from the other entity, either entity MAY refuse the session by sending a SESS_TERM message with an appropriate reason code.

The format for the Contact Header is as follows:

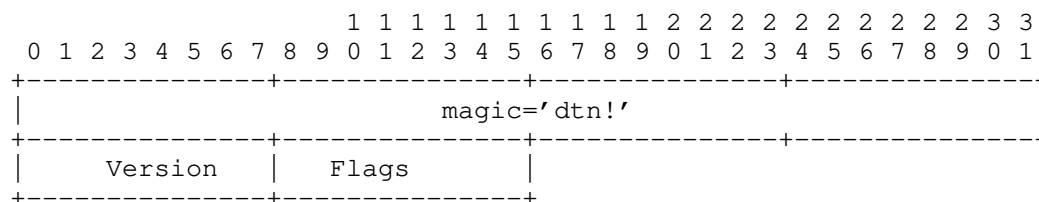


Figure 16: Contact Header Format

See Section 4.3 for details on the use of each of these Contact Header fields.

The fields of the Contact Header are:

magic: A four-octet field that always contains the octet sequence 0x64 0x74 0x6E 0x21, i.e., the text string "dtn!" in US-ASCII (and UTF-8).

Version: A one-octet field value containing the value 4 (current version of the TCPCL).

Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 1. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Name	Code	Description
CAN_TLS	0x01	If bit is set, indicates that the sending peer is capable of TLS security.
Reserved	others	

Table 1: Contact Header Flags

4.3. Contact Validation and Negotiation

Upon reception of the Contact Header, each node follows the following procedures to ensure the validity of the TCPCL session and to negotiate values for the session parameters.

If the magic string is not present or is not valid, the connection MUST be terminated. The intent of the magic string is to provide some protection against an inadvertent TCP connection by a different protocol than the one described in this document. To prevent a flood of repeated connections from a misconfigured application, a passive entity MAY deny new TCP connections from a specific peer address for a period of time after one or more connections fail to provide a decodable Contact Header.

The first negotiation is on the TCPCL protocol version to use. The active entity always sends its Contact Header first and waits for a response from the passive entity. During contact initiation, the active TCPCL node SHALL send the highest TCPCL protocol version on a first session attempt for a TCPCL peer. If the active entity receives a Contact Header with a lower protocol version than the one sent earlier on the TCP connection, the TCP connection SHALL be closed. If the active entity receives a SESS_TERM message with reason of "Version Mismatch", that node MAY attempt further TCPCL sessions with the peer using earlier protocol version numbers in decreasing order. Managing multi-TCPCL-session state such as this is an implementation matter.

If the passive entity receives a Contact Header containing a version that is not a version of the TCPCL that the entity implements, then the entity SHALL send its Contact Header and immediately terminate the session with a reason code of "Version mismatch". If the passive entity receives a Contact Header with a version that is lower than the latest version of the protocol that the entity implements, the entity MAY either terminate the session (with a reason code of "Version mismatch") or adapt its operation to conform to the older

version of the protocol. The decision of version fall-back is an implementation matter.

The negotiated contact parameters defined by this specification are described in the following paragraphs.

TCPCL Version: Both Contact Headers of a successful contact negotiation have identical TCPCL Version numbers as described above. Only upon response of a Contact Header from the passive entity is the TCPCL protocol version established and session negotiation begun.

Enable TLS: Negotiation of the Enable TLS parameter is performed by taking the logical AND of the two Contact Headers' CAN_TLS flags. A local security policy is then applied to determine if the negotiated value of Enable TLS is acceptable. It can be a reasonable security policy to require or disallow the use of TLS depending upon the desired network flows. Because this state is negotiated over an unsecured medium, there is a risk of a TLS Stripping as described in Section 8. If the Enable TLS state is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note that this contact failure reason is different than a failure of TLS handshake or TLS authentication after an agreed-upon and acceptable Enable TLS state. If the negotiated Enable TLS value is true and acceptable then TLS negotiation feature (described in Section 4.4) begins immediately following the Contact Header exchange.

4.4. Session Security

This version of the TCPCL supports establishing a Transport Layer Security (TLS) session within an existing TCP connection. When TLS is used within the TCPCL it affects the entire session. Once TLS is established, there is no mechanism available to downgrade the TCPCL session to non-TLS operation.

Once established, the lifetime of a TLS connection SHALL be bound to the lifetime of the underlying TCP connection. Immediately prior to actively ending a TLS connection after TCPCL session termination, the peer which sent the original (non-reply) SESS_TERM message SHOULD follow the Closure Alert procedure of [RFC8446] to cleanly terminate the TLS connection. Because each TCPCL message is either fixed-length or self-indicates its length, the lack of a TLS Closure Alert will not cause data truncation or corruption.

Subsequent TCPCL session attempts to the same passive entity MAY attempt use the TLS connection resumption feature. There is no guarantee that the passive entity will accept the request to resume a

TLS session, and the active entity cannot assume any resumption outcome.

4.4.1. Entity Identification

The TCPCL uses TLS for certificate exchange in both directions to identify each entity and to allow each entity to authenticate its peer. Each certificate can potentially identify multiple entities and there is no problem using such a certificate as long as the identifiers are sufficient to meet authentication policy (as described in later sections) for the entity which presents it.

Because the PKIX environment of each TCPCL entity are likely not controlled by the certificate end users (see Section 3.4), the TCPCL defines a prioritized list of what a certificate can identify about a TCPCL entity:

Node ID: The ideal certificate identity is the Node ID of the entity using the NODE-ID definition below. When the Node ID is identified, there is no need for any lower-level identification to take place.

DNS Name: If CA policy forbids a certificate to contain an arbitrary NODE-ID but allows a DNS-ID to be identified then one or more stable DNS names can be identified in the certificate. The use of wildcard DNS-ID is discouraged due to the complex rules for matching and dependence on implementation support for wildcard matching (see Section 6.4.3 of [RFC6125]).

Network Address: If no stable DNS name is available but a stable network address is available and CA policy allows a certificate to contain a IPADDR-ID (as defined below) then one or more network addresses can be identified in the certificate.

When only a DNS-ID or IPADDR-ID can be identified by a certificate, it is implied that an entity which authenticates using that certificate is trusted to provide a valid Node ID in its SESS_INIT; the certificate itself does not actually authenticate that Node ID.

The RECOMMENDED security policy of an entity is to validate a peer which authenticates its Node ID regardless of an authenticated DNS name or address, and only consider the host/address authentication in the absence of an authenticated Node ID.

This specification defines a NODE-ID of a certificate as being the subjectAltName entry of type uniformResourceIdentifier whose value is a URI consistent with the requirements of [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Type" registry. This

is similar to the URI-ID of [RFC6125] but does not require any structure to the scheme-specific-part of the URI. Unless specified otherwise by the definition of the URI scheme being authenticated, URI matching of a NODE-ID SHALL use the URI comparison logic of [RFC3986] and scheme-based normalization of those schemes specified in [I-D.ietf-dtn-bpbis]. A URI scheme can refine this "exact match" logic with rules about how Node IDs within that scheme are to be compared with the certificate-authenticated NODE-ID.

This specification defines a IPADDR-ID of a certificate as being the subjectAltName entry of type ipAddress whose value is encoded according to [RFC5280].

4.4.2. TLS Handshake

The use of TLS is negotiated using the Contact Header as described in Section 4.3. After negotiating an Enable TLS parameter of true, and before any other TCPCL messages are sent within the session, the session entities SHALL begin a TLS handshake in accordance with [RFC8446]. By convention, this protocol uses the entity which initiated the underlying TCP connection (the active peer) as the "client" role of the TLS handshake request.

The TLS handshake, if it occurs, is considered to be part of the contact negotiation before the TCPCL session itself is established. Specifics about sensitive data exposure are discussed in Section 8.

The parameters within each TLS negotiation are implementation dependent but any TCPCL node SHALL follow all recommended practices of BCP 195 [RFC7525], or any updates or successors that become part of BCP 195. Within each TLS handshake, the following requirements apply (using the rough order in which they occur):

Client Hello: When a resolved DNS name was used to establish the TCP connection, the TLS ClientHello SHOULD include a Server Name Indication (SNI) in accordance with [RFC6066]. When present, the "server_name" extension SHALL contain a "HostName" value taken from the DNS name (of the passive entity) which was resolved. Note: The 'HostName' in the "server_name" extension is the network name for the passive entity, not the Node ID of that entity.

Server Certificate: The passive entity SHALL supply a certificate within the TLS handshake to allow authentication of its side of the session. Unless prohibited by CA policy, the passive entity certificate SHALL contain a NODE-ID which authenticates the Node ID of the peer. When assigned a stable DNS name, the passive entity certificate SHOULD contain a DNS-ID which authenticates that (fully qualified) name. When assigned a stable network

address, the passive entity certificate MAY contain a IPADDR-ID which authenticates that address. The passive entity MAY use the SNI DNS name to choose an appropriate server-side certificate which authenticates that DNS name.

Certificate Request: During TLS handshake, the passive entity SHALL request a client-side certificate.

Client Certificate: The active entity SHALL supply a certificate chain within the TLS handshake to allow authentication of its side of the session. Unless prohibited by CA policy, the active entity certificate SHALL contain a NODE-ID which authenticates the Node ID of the peer. When assigned a stable DNS name, the active entity certificate SHOULD contain a DNS-ID which authenticates that (fully qualified) name. When assigned a stable network address, the active entity certificate MAY contain a IPADDR-ID which authenticates that address.

All certificates supplied during TLS handshake SHALL conform to [RFC5280], or any updates or successors to that profile. When a certificate is supplied during TLS handshake, the full certification chain SHOULD be included unless security policy indicates that is unnecessary.

If a TLS handshake cannot negotiate a TLS connection, both entities of the TCPCL session SHALL close the TCP connection. At this point the TCPCL session has not yet been established so there is no TCPCL session to terminate.

After a TLS connection is successfully established, the active entity SHALL send a SESS_INIT message to begin session negotiation. This session negotiation and all subsequent messaging are secured.

4.4.3. TLS Authentication

Using PKIX certificates exchanged during the TLS handshake, each of the entities can authenticate a peer Node ID directly or authenticate the peer DNS name or network address. The logic for attempting authentication is separate from the logic for handling the result of authentication, which is based on local security policy.

By using the SNI DNS name (see Section 4.4.2) a single passive entity can act as a convergence layer for multiple BP agents with distinct Node IDs. When this "virtual host" behavior is used, the DNS name is used as the indication of which BP Node the active entity is attempting to communicate with. A virtual host CL entity can be authenticated by a certificate containing all of the DNS names and/or Node IDs being hosted or by several certificates each authenticating

a single DNS name and/or Node ID, using the SNI value from the peer to select which certificate to use.

Any certificate received during TLS handshake SHALL be validated up to one or more trusted CA certificates. If certificate validation fails or if security policy disallows a certificate for any reason, the entity SHALL terminate the session (with a reason code of "Contact Failure").

The result of authenticating a peer value against a certificate claim is one of:

Absent: Indicating that the claim type is not present in the certificate and cannot be authenticated.

Success: Indicating the claim type is present and matches the peer (Node ID / DNS name / address) value.

Failure: Indicating the claim type is present but did not match the peer value.

Either during or immediately after the TLS handshake, the active entity SHALL authenticate the DNS name (of the passive entity) used to initiate the TCP connection using any DNS-ID of the peer certificate. If the DNS name authentication result is Failure or if the result is Absent and security policy requires an authenticated DNS name, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Either during or immediately after the TLS handshake, the active entity SHALL authenticate the IP address of the other side of the TCP connection using any IPADDR-ID of the peer certificate. Either during or immediately after the TLS handshake, the passive entity SHALL authenticate the IP address of the other side of the TCP connection using any IPADDR-ID of the peer certificate. If the address authentication result is Failure or if the result is Absent and security policy requires an authenticated network address, the entity SHALL terminate the session (with a reason code of "Contact Failure").

Immediately before Session Parameter Negotiation, each side of the session SHALL authenticate the Node ID of the peer's SESS_INIT message using any NODE-ID of the peer certificate. If the Node ID authentication result is Failure or if the result is Absent and security policy requires an authenticated Node ID, the entity SHALL terminate the session (with a reason code of "Contact Failure").

4.4.4. Example TLS Initiation

A summary of a typical TLS use is shown in the sequence in Figure 17 below. In this example the active peer terminates the session but termination can be initiated from either peer.

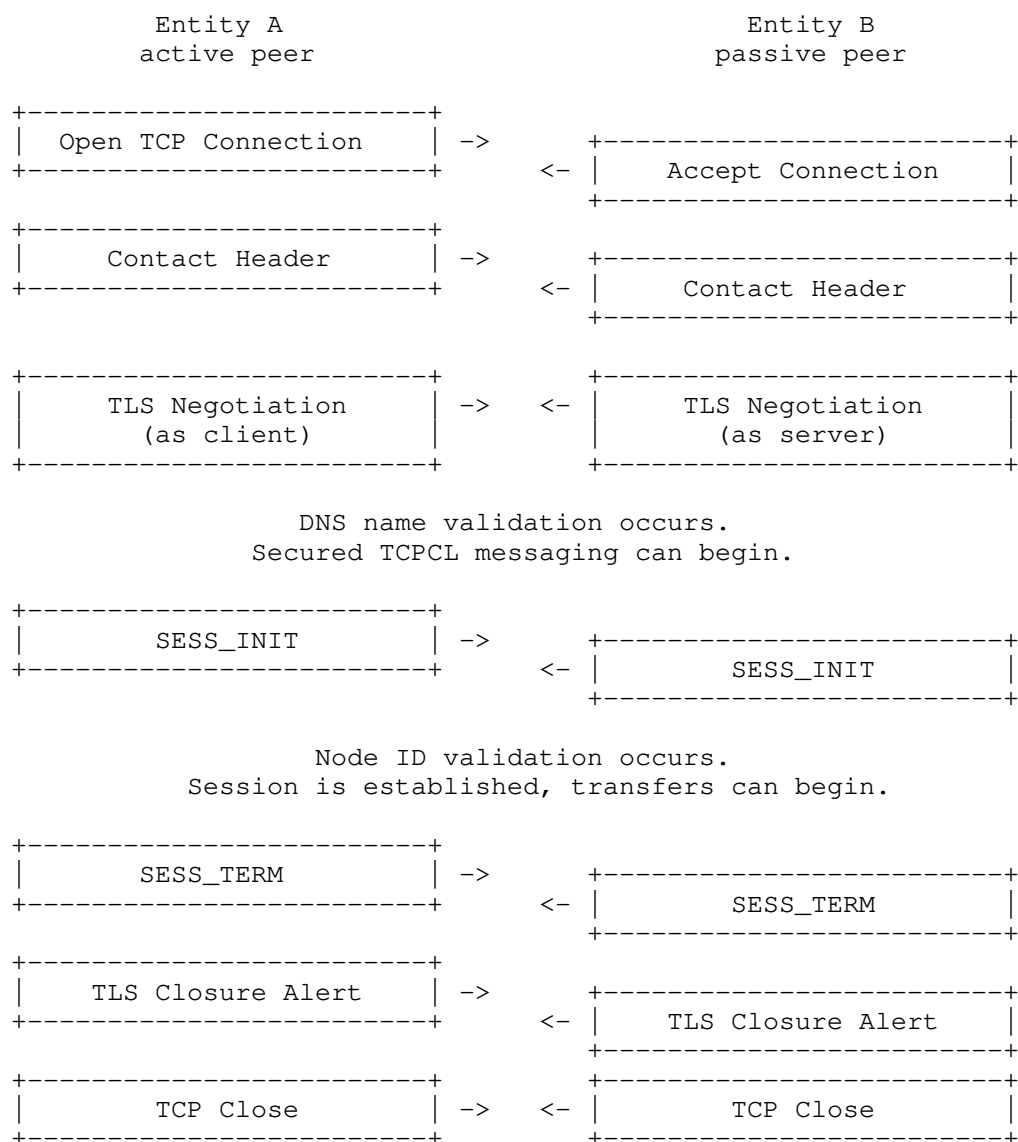


Figure 17: A simple visual example of TCPCL TLS Establishment between two entities

4.5. Message Header

After the initial exchange of a Contact Header, all messages transmitted over the session are identified by a one-octet header with the following structure:

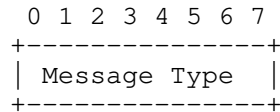


Figure 18: Format of the Message Header

The message header fields are as follows:

Message Type: Indicates the type of the message as per Table 2 below. Encoded values are listed in Section 9.5.

Name	Code	Description
SESS_INIT	0x07	Contains the session parameter inputs from one of the entities, as described in Section 4.6.
SESS_TERM	0x05	Indicates that one of the entities participating in the session wishes to cleanly terminate the session, as described in Section 6.1.
XFER_SEGMENT	0x01	Indicates the transmission of a segment of bundle data, as described in Section 5.2.2.
XFER_ACK	0x02	Acknowledges reception of a data segment, as described in Section 5.2.3.
XFER_REFUSE	0x03	Indicates that the transmission of the current bundle SHALL be stopped, as described in Section 5.2.4.
KEEPALIVE	0x04	Used to keep TCPCL session active, as described in Section 5.1.1.
MSG_REJECT	0x06	Contains a TCPCL message rejection, as described in Section 5.1.2.

Table 2: TCPCL Message Types

4.6. Session Initialization Message (SESS_INIT)

Before a session is established and ready to transfer bundles, the session parameters are negotiated between the connected entities. The SESS_INIT message is used to convey the per-entity parameters which are used together to negotiate the per-session parameters as described in Section 4.7.

The format of a SESS_INIT message is as follows in Figure 19.

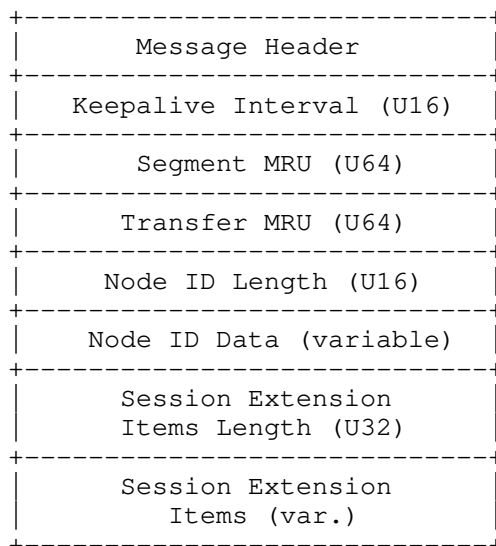


Figure 19: SESS_INIT Format

The fields of the SESS_INIT message are:

Keepalive Interval: A 16-bit unsigned integer indicating the minimum interval, in seconds, to negotiate as the Session Keepalive using the method of Section 4.7.

Segment MRU: A 64-bit unsigned integer indicating the largest allowable single-segment data payload size to be received in this session. Any XFER_SEGMENT sent to this peer SHALL have a data payload no longer than the peer's Segment MRU. The two entities of a single session MAY have different Segment MRUs, and no relation between the two is required.

Transfer MRU: A 64-bit unsigned integer indicating the largest allowable total-bundle data size to be received in this session. Any bundle transfer sent to this peer SHALL have a Total Bundle Length payload no longer than the peer's Transfer MRU. This value can be used to perform proactive bundle fragmentation. The two entities of a single session MAY have different Transfer MRUs, and no relation between the two is required.

Node ID Length and Node ID Data: Together these fields represent a variable-length text string. The Node ID Length is a 16-bit unsigned integer indicating the number of octets of Node ID Data to follow. A zero-length Node ID SHALL be used to indicate the lack of Node ID rather than a truly empty Node ID. This case

allows an entity to avoid exposing Node ID information on an untrusted network. A non-zero-length Node ID Data SHALL contain the UTF-8 encoded Node ID of the Entity which sent the SESS_INIT message. Every Node ID SHALL be a URI consistent with the requirements of [RFC3986] and the URI schemes of the IANA "Bundle Protocol URI Scheme Type" registry. The Node ID itself can be authenticated as described in Section 4.4.3.

Session Extension Length and Session Extension Items:

Together these fields represent protocol extension data not defined by this specification. The Session Extension Length is the total number of octets to follow which are used to encode the Session Extension Item list. The encoding of each Session Extension Item is within a consistent data container as described in Section 4.8. The full set of Session Extension Items apply for the duration of the TCPCL session to follow. The order and multiplicity of these Session Extension Items is significant, as defined in the associated type specification(s). If the content of the Session Extension Items data disagrees with the Session Extension Length (e.g., the last Item claims to use more octets than are present in the Session Extension Length), the reception of the SESS_INIT is considered to have failed.

When the active entity initiates a TCPCL session, it is likely based on routing information which binds a Node ID to CL parameters. If the active entity receives a SESS_INIT with different Node ID than was intended for the TCPCL session, the session MAY be allowed to be established. If allowed, such a session SHALL be associated with the Node ID provided in the SESS_INIT message rather than any intended value.

4.7. Session Parameter Negotiation

An entity calculates the parameters for a TCPCL session by negotiating the values from its own preferences (conveyed by the SESS_INIT it sent to the peer) with the preferences of the peer node (expressed in the SESS_INIT that it received from the peer). The negotiated parameters defined by this specification are described in the following paragraphs.

Transfer MTU and Segment MTU: The maximum transmit unit (MTU) for whole transfers and individual segments are identical to the Transfer MRU and Segment MRU, respectively, of the received SESS_INIT message. A transmitting peer can send individual segments with any size smaller than the Segment MTU, depending on local policy, dynamic network conditions, etc. Determining the size of each transmitted segment is an implementation matter. If either the Transfer MRU or Segment MRU is unacceptable, the entity

SHALL terminate the session with a reason code of "Contact Failure".

Session Keepalive: Negotiation of the Session Keepalive parameter is performed by taking the minimum of the two Keepalive Interval values from the two SESS_INIT messages. The Session Keepalive interval is a parameter for the behavior described in Section 5.1.1. If the Session Keepalive interval is unacceptable, the entity SHALL terminate the session with a reason code of "Contact Failure". Note: a negotiated Session Keepalive of zero indicates that KEEPALIVES are disabled.

Once this process of parameter negotiation is completed (which includes a possible completed TLS handshake of the connection to use TLS), this protocol defines no additional mechanism to change the parameters of an established session; to effect such a change, the TCPCL session MUST be terminated and a new session established.

4.8. Session Extension Items

Each of the Session Extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 20.

The fields of the Session Extension Item are:

Item Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 3. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a TCPCL entity receives a Session Extension Item with an unknown Item Type and the CRITICAL flag of 1, the entity SHALL close the TCPCL session with SESS_TERM reason code of "Contact Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification does not define any extension types directly, but does create an IANA registry for such codes (see Section 9.3).

Item Length: A 16-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data

lengths, as no bundle transfers can begin until the full extension data is sent.

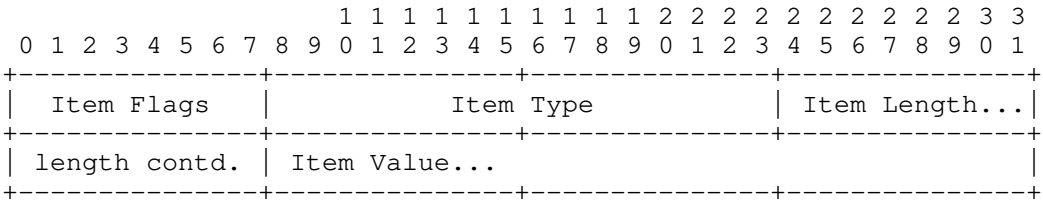


Figure 20: Session Extension Item Format

Name	Code	Description
CRITICAL	0x01	If bit is set, indicates that the receiving peer must handle the extension item.
Reserved	others	

Table 3: Session Extension Item Flags

5. Established Session Operation

This section describes the protocol operation for the duration of an established session, including the mechanism for transmitting bundles over the session.

5.1. Upkeep and Status Messages

5.1.1. Session Upkeep (KEEPALIVE)

The protocol includes a provision for transmission of KEEPALIVE messages over the TCPCL session to help determine if the underlying TCP connection has been disrupted.

As described in Section 4.3, a negotiated parameter of each session is the Session Keepalive interval. If the negotiated Session Keepalive is zero (i.e., one or both contact headers contains a zero Keepalive Interval), then the keepalive feature is disabled. There is no logical minimum value for the keepalive interval (within the minimum imposed by the positive-value encoding), but when used for many sessions on an open, shared network a short interval could lead to excessive traffic. For shared network use, entities SHOULD choose a keepalive interval no shorter than 30 seconds. There is no logical maximum value for the keepalive interval (within the maximum imposed

by the fixed-size encoding), but an idle TCP connection is liable for closure by the host operating system if the keepalive time is longer than tens-of-minutes. Entities SHOULD choose a keepalive interval no longer than 10 minutes (600 seconds).

Note: The Keepalive Interval SHOULD NOT be chosen too short as TCP retransmissions MAY occur in case of packet loss. Those will have to be triggered by a timeout (TCP retransmission timeout (RTO)), which is dependent on the measured RTT for the TCP connection so that KEEPALIVE messages can experience noticeable latency.

The format of a KEEPALIVE message is a one-octet message type code of KEEPALIVE (as described in Table 2) with no additional data. Both sides SHALL send a KEEPALIVE message whenever the negotiated interval has elapsed with no transmission of any message (KEEPALIVE or other).

If no message (KEEPALIVE or other) has been received in a session after some implementation-defined time duration, then the entity SHALL terminate the session by transmitting a SESS_TERM message (as described in Section 6.1) with reason code "Idle Timeout". If configurable, the idle timeout duration SHOULD be no shorter than twice the keepalive interval. If not configurable, the idle timeout duration SHOULD be exactly twice the keepalive interval.

5.1.2. Message Rejection (MSG_REJECT)

This message type is not expected to be seen in a well-functioning session. Its purpose is to aid in troubleshooting bad entity behavior by allowing the peer to observe why an entity is not responding as expected to its messages.

If a TCPCL entity receives a message type which is unknown to it (possibly due to an unhandled protocol version mismatch or a incorrectly-negotiated session extension which defines a new message type), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Type Unknown" and close the TCP connection. If a TCPCL entity receives a message type which is known but is inappropriate for the negotiated session parameters (possibly due to incorrectly-negotiated session extension), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Unsupported". If a TCPCL entity receives a message which is inappropriate for the current session state (e.g., a SESS_INIT after the session has already been established or an XFER_ACK message with an unknown Transfer ID), the entity SHALL send a MSG_REJECT message with a Reason Code of "Message Unexpected".

The format of a MSG_REJECT message is as follows in Figure 21.

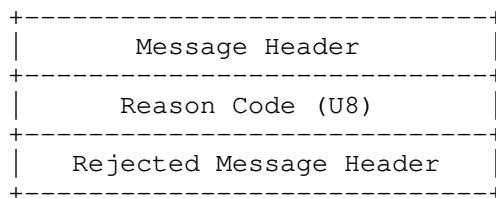


Figure 21: Format of MSG_REJECT Messages

The fields of the MSG_REJECT message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 4.

Rejected Message Header: The Rejected Message Header is a copy of the Message Header to which the MSG_REJECT message is sent as a response.

Name	Code	Description
Message Type Unknown	0x01	A message was received with a Message Type code unknown to the TCPCL node.
Message Unsupported	0x02	A message was received but the TCPCL entity cannot comply with the message contents.
Message Unexpected	0x03	A message was received while the session is in a state in which the message is not expected.

Table 4: MSG_REJECT Reason Codes

5.2. Bundle Transfer

All of the messages in this section are directly associated with transferring a bundle between TCPCL entities.

A single TCPCL transfer results in a bundle (handled by the convergence layer as opaque data) being exchanged from one node to the other. In TCPCL a transfer is accomplished by dividing a single bundle up into "segments" based on the receiving-side Segment MRU (see Section 4.2). The choice of the length to use for segments is an implementation matter, but each segment MUST NOT be larger than

the receiving node's maximum receive unit (MRU) (see the field Segment MRU of Section 4.2). The first segment for a bundle is indicated by the 'START' flag and the last segment is indicated by the 'END' flag.

A single transfer (and by extension a single segment) SHALL NOT contain data of more than a single bundle. This requirement is imposed on the agent using the TCPCL rather than TCPCL itself.

If multiple bundles are transmitted on a single TCPCL connection, they MUST be transmitted consecutively without interleaving of segments from multiple bundles.

5.2.1. Bundle Transfer ID

Each of the bundle transfer messages contains a Transfer ID which is used to correlate messages (from both sides of a transfer) for each bundle. A Transfer ID does not attempt to address uniqueness of the bundle data itself and has no relation to concepts such as bundle fragmentation. Each invocation of TCPCL by the bundle protocol agent, requesting transmission of a bundle (fragmentary or otherwise), results in the initiation of a single TCPCL transfer. Each transfer entails the sending of a sequence of some number of XFER_SEGMENT and XFER_ACK messages; all are correlated by the same Transfer ID. The sending entity originates a transfer ID and the receiving entity uses that same Transfer ID in acknowledgements.

Transfer IDs from each node SHALL be unique within a single TCPCL session. Upon exhaustion of the entire 64-bit Transfer ID space, the sending node SHALL terminate the session with SESS_TERM reason code "Resource Exhaustion". For bidirectional bundle transfers, a TCPCL node SHOULD NOT rely on any relation between Transfer IDs originating from each side of the TCPCL session.

Although there is not a strict requirement for Transfer ID initial values or ordering (see Section 8.11), in the absence of any other mechanism for generating Transfer IDs an entity SHALL use the following algorithm: The initial Transfer ID from each node is zero and subsequent Transfer ID values are incremented from the prior Transfer ID value by one.

5.2.2. Data Transmission (XFER_SEGMENT)

Each bundle is transmitted in one or more data segments. The format of a XFER_SEGMENT message follows in Figure 22.

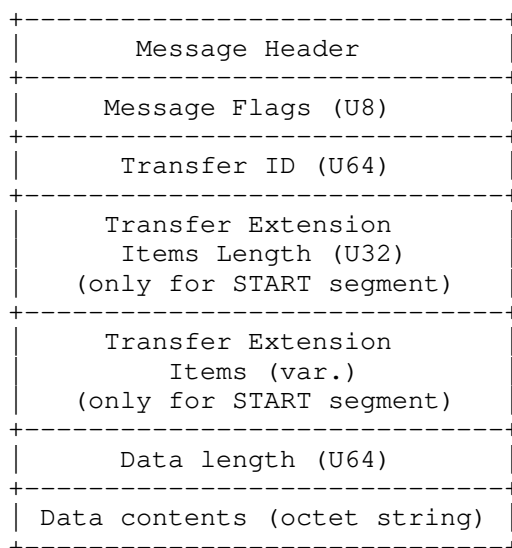


Figure 22: Format of XFER_SEGMENT Messages

The fields of the XFER_SEGMENT message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being made.

Transfer Extension Length and Transfer Extension Items:
Together these fields represent protocol extension data for this specification. The Transfer Extension Length and Transfer Extension Item fields SHALL only be present when the 'START' flag is set to 1 on the message. The Transfer Extension Length is the total number of octets to follow which are used to encode the Transfer Extension Item list. The encoding of each Transfer Extension Item is within a consistent data container as described in Section 5.2.5. The full set of transfer extension items apply only to the associated single transfer. The order and multiplicity of these transfer extension items is significant, as defined in the associated type specification(s). If the content of the Transfer Extension Items data disagrees with the Transfer Extension Length (e.g., the last Item claims to use more octets than are present in the Transfer Extension Length), the reception of the XFER_SEGMENT is considered to have failed.

Data length: A 64-bit unsigned integer indicating the number of octets in the Data contents to follow.

Data contents: The variable-length data payload of the message.

Name	Code	Description
END	0x01	If bit is set, indicates that this is the last segment of the transfer.
START	0x02	If bit is set, indicates that this is the first segment of the transfer.
Reserved	others	

Table 5: XFER_SEGMENT Flags

The flags portion of the message contains two flag values in the two low-order bits, denoted 'START' and 'END' in Table 5. The 'START' flag SHALL be set to 1 when transmitting the first segment of a transfer. The 'END' flag SHALL be set to 1 when transmitting the last segment of a transfer. In the case where an entire transfer is accomplished in a single segment, both the 'START' and 'END' flags SHALL be set to 1.

Once a transfer of a bundle has commenced, the entity MUST only send segments containing sequential portions of that bundle until it sends a segment with the 'END' flag set to 1. No interleaving of multiple transfers from the same node is possible within a single TCPCL session. Simultaneous transfers between two entities MAY be achieved using multiple TCPCL sessions.

5.2.3. Data Acknowledgments (XFER_ACK)

Although the TCP transport provides reliable transfer of data between transport peers, the typical BSD sockets interface provides no means to inform a sending application of when the receiving application has processed some amount of transmitted data. Thus, after transmitting some data, the TCPCL needs an additional mechanism to determine whether the receiving agent has successfully received and fully processed the segment. To this end, the TCPCL protocol provides feedback messaging whereby a receiving node transmits acknowledgments of reception of data segments.

The format of an XFER_ACK message follows in Figure 23.

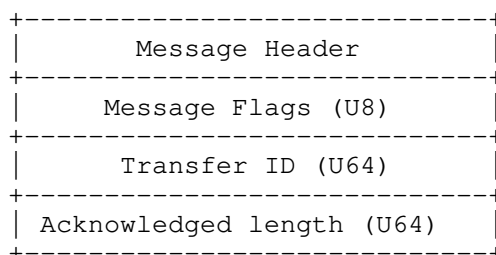


Figure 23: Format of XFER_ACK Messages

The fields of the XFER_ACK message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 5. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Transfer ID: A 64-bit unsigned integer identifying the transfer being acknowledged.

Acknowledged length: A 64-bit unsigned integer indicating the total number of octets in the transfer which are being acknowledged.

A receiving TCPCL node SHALL send an XFER_ACK message in response to each received XFER_SEGMENT message after the segment has been fully processed. The flags portion of the XFER_ACK header SHALL be set to match the corresponding XFER_SEGMENT message being acknowledged (including flags not decodable to the entity). The acknowledged length of each XFER_ACK contains the sum of the data length fields of all XFER_SEGMENT messages received so far in the course of the indicated transfer. The sending node SHOULD transmit multiple XFER_SEGMENT messages without waiting for the corresponding XFER_ACK responses. This enables pipelining of messages on a transfer stream.

For example, suppose the sending node transmits four segments of bundle data with lengths 100, 200, 500, and 1000, respectively. After receiving the first segment, the entity sends an acknowledgment of length 100. After the second segment is received, the entity sends an acknowledgment of length 300. The third and fourth acknowledgments are of length 800 and 1800, respectively.

5.2.4. Transfer Refusal (XFER_REFUSE)

The TCPCL supports a mechanism by which a receiving node can indicate to the sender that it does not want to receive the corresponding bundle. To do so, upon receiving an XFER_SEGMENT message, the entity

MAY transmit a XFER_REFUSE message. As data segments and acknowledgments can cross on the wire, the bundle that is being refused SHALL be identified by the Transfer ID of the refusal.

There is no required relation between the Transfer MRU of a TCPCL node (which is supposed to represent a firm limitation of what the node will accept) and sending of a XFER_REFUSE message. A XFER_REFUSE can be used in cases where the agent's bundle storage is temporarily depleted or somehow constrained. A XFER_REFUSE can also be used after the bundle header or any bundle data is inspected by an agent and determined to be unacceptable.

A transfer receiver MAY send an XFER_REFUSE message as soon as it receives any XFER_SEGMENT message. The transfer sender MUST be prepared for this and MUST associate the refusal with the correct bundle via the Transfer ID fields.

The TCPCL itself does not have any required behavior to respond to an XFER_REFUSE based on its Reason Code; the refusal is passed up as an indication to the BP agent that the transfer has been refused. If a transfer refusal has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the refusal as having an Unknown reason.

The format of the XFER_REFUSE message is as follows in Figure 24.

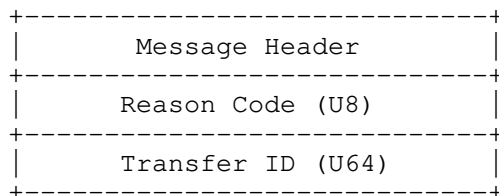


Figure 24: Format of XFER_REFUSE Messages

The fields of the XFER_REFUSE message are:

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 6.

Transfer ID: A 64-bit unsigned integer identifying the transfer being refused.

Name	Code	Description
Unknown	0x00	Reason for refusal is unknown or not specified.
Completed	0x01	The receiver already has the complete bundle. The sender MAY consider the bundle as completely received.
No Resources	0x02	The receiver's resources are exhausted. The sender SHOULD apply reactive bundle fragmentation before retrying.
Retransmit	0x03	The receiver has encountered a problem that requires the bundle to be retransmitted in its entirety.
Not Acceptable	0x04	Some issue with the bundle data or the transfer extension data was encountered. The sender SHOULD NOT retry the same bundle with the same extensions.
Extension Failure	0x05	A failure processing the Transfer Extension Items has occurred.

Table 6: XFER_REFUSE Reason Codes

The receiver MUST, for each transfer preceding the one to be refused, have either acknowledged all XFER_SEGMENT messages or refused the bundle transfer.

The bundle transfer refusal MAY be sent before an entire data segment is received. If a sender receives a XFER_REFUSE message, the sender MUST complete the transmission of any partially sent XFER_SEGMENT message. There is no way to interrupt an individual TCPCL message partway through sending it. The sender MUST NOT commence transmission of any further segments of the refused bundle subsequently. Note, however, that this requirement does not ensure that an entity will not receive another XFER_SEGMENT for the same bundle after transmitting a XFER_REFUSE message since messages can cross on the wire; if this happens, subsequent segments of the bundle SHALL also be refused with a XFER_REFUSE message.

Note: If a bundle transmission is aborted in this way, the receiver does not receive a segment with the 'END' flag set to 1 for the aborted bundle. The beginning of the next bundle is identified by

the 'START' flag set to 1, indicating the start of a new transfer, and with a distinct Transfer ID value.

5.2.5. Transfer Extension Items

Each of the Transfer Extension Items SHALL be encoded in an identical Type-Length-Value (TLV) container form as indicated in Figure 25.

The fields of the Transfer Extension Item are:

Item Flags: A one-octet field containing generic bit flags about the Item, which are listed in Table 7. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver. If a TCPCL node receives a Transfer Extension Item with an unknown Item Type and the CRITICAL flag is 1, the entity SHALL refuse the transfer with an XFER_REFUSE reason code of "Extension Failure". If the CRITICAL flag is 0, an entity SHALL skip over and ignore any item with an unknown Item Type.

Item Type: A 16-bit unsigned integer field containing the type of the extension item. This specification creates an IANA registry for such codes (see Section 9.4).

Item Length: A 16-bit unsigned integer field containing the number of Item Value octets to follow.

Item Value: A variable-length data field which is interpreted according to the associated Item Type. This specification places no restrictions on an extension's use of available Item Value data. Extension specifications SHOULD avoid the use of large data lengths, as the associated transfer cannot begin until the full extension data is sent.

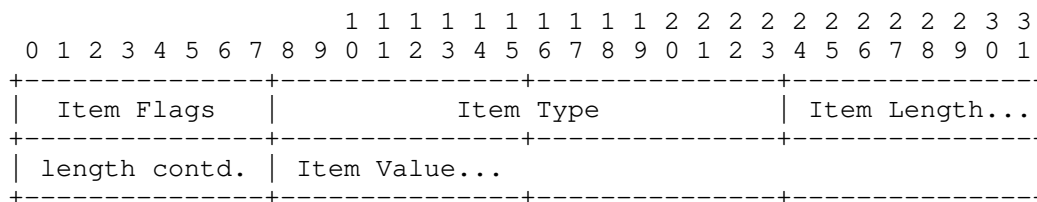


Figure 25: Transfer Extension Item Format

Name	Code	Description
CRITICAL	0x01	If bit is set, indicates that the receiving peer must handle the extension item.
Reserved	others	

Table 7: Transfer Extension Item Flags

5.2.5.1. Transfer Length Extension

The purpose of the Transfer Length extension is to allow entities to preemptively refuse bundles that would exceed their resources or to prepare storage on the receiving node for the upcoming bundle data.

Multiple Transfer Length extension items SHALL NOT occur within the same transfer. The lack of a Transfer Length extension item in any transfer SHALL NOT imply anything about the potential length of the transfer. The Transfer Length extension SHALL be assigned transfer extension type ID 0x0001.

If a transfer occupies exactly one segment (i.e., both START and END flags are 1) the Transfer Length extension SHOULD NOT be present. The extension does not provide any additional information for single-segment transfers.

The format of the Transfer Length data is as follows in Figure 26.

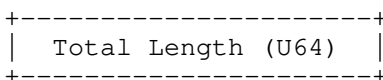


Figure 26: Format of Transfer Length data

The fields of the Transfer Length extension are:

Total Length: A 64-bit unsigned integer indicating the size of the data-to-be-transferred. The Total Length field SHALL be treated as authoritative by the receiver. If, for whatever reason, the actual total length of bundle data received differs from the value indicated by the Total Length value, the receiver SHALL treat the transmitted data as invalid and send an XFER_REFUSE with a Reason Code of "Not Acceptable".

6. Session Termination

This section describes the procedures for terminating a TCPCL session. The purpose of terminating a session is to allow transfers to complete before the session is closed but not allow any new transfers to start. A session state change is necessary for this to happen because transfers can be in-progress in either direction (transfer stream) within a session. Waiting for a transfer to complete in one direction does not control or influence the possibility of a transfer in the other direction. Either peer of a session can terminate an established session at any time.

6.1. Session Termination Message (SESS_TERM)

To cleanly terminate a session, a SESS_TERM message SHALL be transmitted by either node at any point following complete transmission of any other message. When sent to initiate a termination, the REPLY flag of a SESS_TERM message SHALL be 0. Upon receiving a SESS_TERM message after not sending a SESS_TERM message in the same session, an entity SHALL send an acknowledging SESS_TERM message. When sent to acknowledge a termination, a SESS_TERM message SHALL have identical data content from the message being acknowledged except for the REPLY flag, which is set to 1 to indicate acknowledgement.

Once a SESS_TERM message is sent the state of that TCPCL session changes to Ending. While the session is in the Ending state, an entity MAY finish an in-progress transfer in either direction. While the session is in the Ending state, an entity SHALL NOT begin any new outgoing transfer for the remainder of the session. While the session is in the Ending state, an entity SHALL NOT accept any new incoming transfer for the remainder of the session.

Instead of following a clean termination sequence, after transmitting a SESS_TERM message an entity MAY immediately close the associated TCP connection. When performing an unclean termination, a receiving node SHOULD acknowledge all received XFER_SEGMENTS with an XFER_ACK before closing the TCP connection. Not acknowledging received segments can result in unnecessary bundle or bundle fragment retransmission. When performing an unclean termination, a transmitting node SHALL treat either sending or receiving a SESS_TERM message (i.e., before the final acknowledgment) as a failure of the transfer. Any delay between request to close the TCP connection and actual closing of the connection (a "half-closed" state) MAY be ignored by the TCPCL entity.

The TCPCL itself does not have any required behavior to respond to an SESS_TERM based on its Reason Code; the termination is passed up as

an indication to the BP agent that the session state has changed. If a termination has a Reason Code which is not decodable to the BP agent, the agent SHOULD treat the termination as having an Unknown reason.

The format of the SESS_TERM message is as follows in Figure 27.

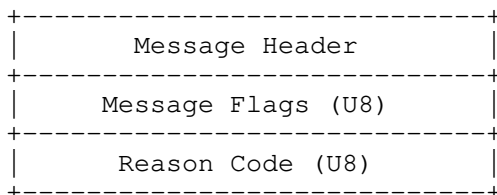


Figure 27: Format of SESS_TERM Messages

The fields of the SESS_TERM message are:

Message Flags: A one-octet field of single-bit flags, interpreted according to the descriptions in Table 8. All reserved header flag bits SHALL be set to 0 by the sender. All reserved header flag bits SHALL be ignored by the receiver.

Reason Code: A one-octet refusal reason code interpreted according to the descriptions in Table 9.

Name	Code	Description
REPLY	0x01	If bit is set, indicates that this message is an acknowledgement of an earlier SESS_TERM message.
Reserved	others	

Table 8: SESS_TERM Flags

Name	Code	Description
Unknown	0x00	A termination reason is not available.
Idle timeout	0x01	The session is being closed due to idleness.
Version mismatch	0x02	The node cannot conform to the specified TCPCL protocol version.
Busy	0x03	The node is too busy to handle the current session.
Contact Failure	0x04	The node cannot interpret or negotiate a Contact Header or SESS_INIT option.
Resource Exhaustion	0x05	The node has run into some resource limit and cannot continue the session.

Table 9: SESS_TERM Reason Codes

The earliest a TCPCL session termination MAY occur is immediately after transmission of a Contact Header (and prior to any further message transmit). This can, for example, be used to notify that the entity is currently not able or willing to communicate. However, an entity MUST always send the Contact Header to its peer before sending a SESS_TERM message.

Termination of the TCP connection MAY occur prior to receiving the Contact header as discussed in Section 4.1. If reception of the Contact Header itself somehow fails (e.g., an invalid "magic string" is received), an entity SHALL close the TCP connection without sending a SESS_TERM message.

If a session is to be terminated before a protocol message has completed being sent, then the entity MUST NOT transmit the SESS_TERM message but still SHALL close the TCP connection. Each TCPCL message is contiguous in the octet stream and has no ability to be cut short and/or preempted by an other message. This is particularly important when large segment sizes are being transmitted; either entire XFER_SEGMENT is sent before a SESS_TERM message or the connection is simply terminated mid-XFER_SEGMENT.

6.2. Idle Session Shutdown

The protocol includes a provision for clean termination of idle sessions. Determining the length of time to wait before ending idle sessions, if they are to be closed at all, is an implementation and configuration matter.

If there is a configured time to close idle links and if no TCPCL messages (other than KEEPALIVE messages) has been received for at least that amount of time, then either node MAY terminate the session by transmitting a SESS_TERM message indicating the reason code of "Idle timeout" (as described in Table 9).

7. Implementation Status

[NOTE to the RFC Editor: please remove this section before publication, as well as the reference to [RFC7942] and [github-dtn-bpbis-tcpcl].]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations can exist.

An example implementation of the this draft of TCPCLv4 has been created as a GitHub project [github-dtn-bpbis-tcpcl] and is intended to use as a proof-of-concept and as a possible source of interoperability testing. This example implementation uses D-Bus as the CL-BP Agent interface, so it only runs on hosts which provide the Python "dbus" library.

8. Security Considerations

This section separates security considerations into threat categories based on guidance of BCP 72 [RFC3552].

8.1. Threat: Passive Leak of Node Data

When used without TLS security, the TCPCL exposes the Node ID and other configuration data to passive eavesdroppers. This occurs even when no transfers occur within a TCPCL session. This can be avoided by always using TLS, even if authentication is not available (see Section 8.10).

8.2. Threat: Passive Leak of Bundle Data

TCPCL can be used to provide point-to-point transport security, but does not provide security of data-at-rest and does not guarantee end-to-end bundle security. The bundle security mechanisms defined in [I-D.ietf-dtn-bpsec] are to be used instead.

When used without TLS security, the TCPCL exposes all bundle data to passive eavesdroppers. This can be avoided by always using TLS, even if authentication is not available (see Section 8.10).

8.3. Threat: TCPCL Version Downgrade

When a TCPCL entity supports multiple versions of the protocol it is possible for a malicious or misconfigured peer to use an older version of TCPCL which does not support transport security. A man-in-the-middle attacker can also manipulate a Contact Header to present a lower protocol version than desired.

It is up to security policies within each TCPCL node to ensure that the negotiated TCPCL version meets transport security requirements.

8.4. Threat: Transport Security Stripping

When security policy allows non-TLS sessions, TCPCL does not protect against active network attackers. It is possible for a man-in-the-middle attacker to set the CAN_TLS flag to 0 on either side of the Contact Header exchange. This leads to the "SSL Stripping" attack described in [RFC7457].

The purpose of the CAN_TLS flag is to allow the use of TCPCL on entities which simply do not have a TLS implementation available. When TLS is available on an entity, it is strongly encouraged that the security policy disallow non-TLS sessions. This requires that the TLS handshake occurs, regardless of the policy-driven parameters of the handshake and policy-driven handling of the handshake outcome.

The negotiated use of TLS is identical behavior to STARTTLS use in [RFC2595] and [RFC4511].

8.5. Threat: Weak TLS Configurations

Even when using TLS to secure the TCPCL session, the actual ciphersuite negotiated between the TLS peers can be insecure. Recommendations for ciphersuite use are included in BCP 195 [RFC7525]. It is up to security policies within each TCPCL node to ensure that the negotiated TLS ciphersuite meets transport security requirements.

8.6. Threat: Certificate Validation Vulnerabilities

Even when TLS itself is operating properly an attacker can attempt to exploit vulnerabilities within certificate check algorithms or configuration to establish a secure TCPCL session using an invalid certificate. A BP agent treats the peer Node ID within a TCPCL session as authoritative and an invalid certificate exploit could lead to bundle data leaking and/or denial of service to the Node ID being impersonated. There are many reasons, described in [RFC5280], why a certificate can fail to validate, including using the certificate outside of its valid time interval, using purposes for which it was not authorized, or using it after it has been revoked by its CA. Validating a certificate is a complex task and can require network connectivity outside of the primary TCPCL network path(s) if a mechanism such as the Online Certificate Status Protocol (OCSP) is used by the CA. The configuration and use of particular certificate validation methods are outside of the scope of this document.

8.7. Threat: Symmetric Key Limits

Even with a secure block cipher and securely-established session keys, there are limits to the amount of plaintext which can be safely encrypted with a given set of keys as described in [AEAD-LIMITS]. When permitted by the negotiated TLS version (see [RFC8446]), it is advisable to take advantage of session key updates to avoid those limits.

8.8. Threat: BP Node Impersonation

The certificates exchanged by TLS enable authentication of peer DNS name and Node ID, but it is possible that a peer either not provide a valid certificate or that the certificate does not validate either the DNS name or Node ID of the peer (see Section 3.4). Having a CA-validated certificate does not alone guarantee the identity of the network host or BP node from which the certificate is provided; additional validation procedures in Section 4.4.2 bind the DNS name or node ID based on the contents of the certificate.

The DNS name validation is a weaker form of authentication, because even if a peer is operating on an authenticated network DNS name it can provide an invalid Node ID and cause bundles to be "leaked" to an invalid node. Especially in DTN environments, network names and addresses of nodes can be time-variable so binding a certificate to a Node ID is a more stable identity.

Node ID validation ensures that the peer to which a bundle is transferred is in fact the node which the BP Agent expects it to be. It is a reasonable policy to skip DNS name validation if certificates can be guaranteed to validate the peer's Node ID. In circumstances where certificates can only be issued to DNS names, Node ID validation is not possible but it could be reasonable to assume that a trusted host is not going to present an invalid Node ID. Determining of when a DNS name authentication can be trusted to validate a Node ID is also a policy matter outside the scope of this document.

8.9. Threat: Denial of Service

The behaviors described in this section all amount to a potential denial-of-service to a TCPCL entity. The denial-of-service could be limited to an individual TCPCL session, could affect other well-behaving sessions on an entity, or could affect all sessions on a host.

A malicious entity can continually establish TCPCL sessions and delay sending of protocol-required data to trigger timeouts. The victim entity can block TCP connections from network peers which are thought to be incorrectly behaving within TCPCL.

An entity can send a large amount of data over a TCPCL session, requiring the receiving entity to handle the data. The victim entity can attempt to stop the flood of data by sending an XFER_REFUSE message, or forcibly terminate the session.

There is the possibility of a "data dribble" attack in which an entity presents a very small Segment MRU which causes transfers to be split among an large number of very small segments and causes the segmentation overhead to overwhelm the actual bundle data segments. Similarly, an entity can present a very small Transfer MRU which will cause resources to be wasted on establishment and upkeep of a TCPCL session over which a bundle could never be transferred. The victim entity can terminate the session during the negotiation of Section 4.7 if the MRUs are unacceptable.

The keepalive mechanism can be abused to waste throughput within a network link which would otherwise be usable for bundle

transmissions. Due to the quantization of the Keepalive Interval parameter the smallest Session Keepalive is one second, which should be long enough to not flood the link. The victim entity can terminate the session during the negotiation of Section 4.7 if the Keepalive Interval is unacceptable.

Finally, an attacker or a misconfigured entity can cause issues at the TCP connection which will cause unnecessary TCP retransmissions or connection resets, effectively denying the use of the overlying TCPCL session.

8.10. Alternate Uses of TLS

This specification makes use of PKIX certificate validation and authentication within TLS. There are alternate uses of TLS which are not necessarily incompatible with the security goals of this specification, but are outside of the scope of this document. The following subsections give examples of alternate TLS uses.

8.10.1. TLS Without Authentication

In environments where PKI is available but there are restrictions on the issuance of certificates (including the contents of certificates), it may be possible to make use of TLS in a way which authenticates only the passive entity of a TCPCL session or which does not authenticate either entity. Using TLS in a way which does not successfully authenticate some claim of both peer entities of a TCPCL session is outside of the scope of this document but does have similar properties to the opportunistic security model of [RFC7435].

8.10.2. Non-Certificate TLS Use

In environments where PKI is unavailable, alternate uses of TLS which do not require certificates such as pre-shared key (PSK) authentication [RFC5489] and the use of raw public keys [RFC7250] are available and can be used to ensure confidentiality within TCPCL. Using non-PKI node authentication methods is outside of the scope of this document.

8.11. Predictability of Transfer IDs

The only requirement on Transfer IDs is that they be unique with each session from the sending peer only. The trivial algorithm of the first transfer starting at zero and later transfers incrementing by one causes absolutely predictable Transfer IDs. Even when a TCPCL session is not TLS secured and there is a man-in-the-middle attacker causing denial of service with XFER_REFUSE messages, it is not

possible to preemptively refuse a transfer so there is no benefit in having unpredictable Transfer IDs within a session.

9. IANA Considerations

Registration procedures referred to in this section are defined in [RFC8126].

Some of the registries have been defined as version specific to TCPCLv4, and imports some or all codepoints from TCPCLv3. This was done to disambiguate the use of these codepoints between TCPCLv3 and TCPCLv4 while preserving the semantics of some of the codepoints.

9.1. Port Number

Within the port registry of [IANA-PORTS], TCP port number 4556 has been previously assigned as the default port for the TCP convergence layer in [RFC7242]. This assignment is unchanged by TCPCL version 4, but the assignment reference is updated to this specification. Each TCPCL entity identifies its TCPCL protocol version in its initial contact (see Section 9.2), so there is no ambiguity about what protocol is being used. The related assignments for UDP and DCCP port 4556 (both registered by [RFC7122]) are unchanged.

Parameter	Value
Service Name:	dtn-bundle
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IESG <iesg@ietf.org>
Description:	DTN Bundle TCP CL Protocol
Reference:	This specification.
Port Number:	4556

9.2. Protocol Versions

IANA has created, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version Numbers". The version number table is updated to include this specification. The registration procedure is RFC Required.

Value	Description	Reference
0	Reserved	[RFC7242]
1	Reserved	[RFC7242]
2	Reserved	[RFC7242]
3	TCPCL	[RFC7242]
4	TCPCLv4	This specification.
5-255	Unassigned	

9.3. Session Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Session Extension Types" and initialize it with the contents of Table 10. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new session extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received during session negotiation.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Session Extension Type
0x0000	Reserved
0x0001--0x7FFF	Unassigned
0x8000--0xFFFF	Private/Experimental Use

Table 10: Session Extension Type Codes

9.4. Transfer Extension Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Transfer Extension Types" and initialize it with the contents of Table 11. The registration procedure is Expert Review within the lower range 0x0001--0x7FFF. Values in the range 0x8000--0xFFFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new transfer extension types need to define the encoding of the Item Value data as well as any meaning or restriction on the number of or order of instances of the type within an extension item list. Specifications need to define how the extension functions when no instance of the new extension type is received in a transfer.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Transfer Extension Type
0x0000	Reserved
0x0001	Transfer Length Extension
0x0002--0x7FFF	Unassigned
0x8000--0xFFFF	Private/Experimental Use

Table 11: Transfer Extension Type Codes

9.5. Message Types

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 Message Types" and initialize it with the contents of Table 12. The registration procedure is RFC Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new message types need to define the encoding of the message data as well as the purpose and relationship of the new message to existing session/transfer state within the baseline message sequencing. The use of new message types need to be negotiated between TCPCL entities within a session (using the session extension mechanism) so that the receiving entity can properly decode all message types used in the session.

Expert(s) are encouraged to favor new session/transfer extension types over new message types. TCPCL messages are not self-delimiting, so care must be taken in introducing new message types. If an entity receives an unknown message type the only thing that can be done is to send a MSG_REJECT and close the TCP connection; not even a clean termination can be done at that point.

Code	Message Type
0x00	Reserved
0x01	XFER_SEGMENT
0x02	XFER_ACK
0x03	XFER_REFUSE
0x04	KEEPALIVE
0x05	SESS_TERM
0x06	MSG_REJECT
0x07	SESS_INIT
0x08--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 12: Message Type Codes

9.6. XFER_REFUSE Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 XFER_REFUSE Reason Codes" and initialize it with the contents of Table 13. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new XFER_REFUSE reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each refusal reason needs to be usable by the receiving BP Agent to make retransmission or re-routing decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Refusal Reason
0x00	Unknown
0x01	Completed
0x02	No Resources
0x03	Retransmit
0x04	Not Acceptable
0x05	Extension Failure
0x06--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 13: XFER_REFUSE Reason Codes

9.7. SESS_TERM Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 SESS_TERM Reason Codes" and initialize it with the contents of Table 14. The registration procedure is Specification Required within the lower range 0x00--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new SESS_TERM reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each termination reason needs to be usable by the receiving BP Agent to make re-connection decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Termination Reason
0x00	Unknown
0x01	Idle timeout
0x02	Version mismatch
0x03	Busy
0x04	Contact Failure
0x05	Resource Exhaustion
0x06--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 14: SESS_TERM Reason Codes

9.8. MSG_REJECT Reason Codes

EDITOR NOTE: sub-registry to-be-created upon publication of this specification.

IANA will create, under the "Bundle Protocol" registry [IANA-BUNDLE], a sub-registry titled "Bundle Protocol TCP Convergence-Layer Version 4 MSG_REJECT Reason Codes" and initialize it with the contents of Table 15. The registration procedure is Specification Required within the lower range 0x01--0xEF. Values in the range 0xF0--0xFF are reserved for use on private networks for functions not published to the IANA.

Specifications of new MSG_REJECT reason codes need to define the meaning of the reason and disambiguate it with pre-existing reasons. Each rejection reason needs to be usable by the receiving TCPCL Entity to make message sequencing and/or session termination decisions.

Expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Code	Rejection Reason
0x00	reserved
0x01	Message Type Unknown
0x02	Message Unsupported
0x03	Message Unexpected
0x04--0xEF	Unassigned
0xF0--0xFF	Private/Experimental Use

Table 15: MSG_REJECT Reason Codes

10. Acknowledgments

This specification is based on comments on implementation of [RFC7242] provided from Scott Burleigh.

11. References

11.1. Normative References

- [I-D.ietf-dtn-bpbis]
Burleigh, S., Fall, K., and E. Birrane, "Bundle Protocol Version 7", draft-ietf-dtn-bpbis-26 (work in progress), July 2020.
- [IANA-BUNDLE]
IANA, "Bundle Protocol",
<<https://www.iana.org/assignments/bundle/>>.
- [IANA-PORTS]
IANA, "Service Name and Transport Protocol Port Number Registry", <<https://www.iana.org/assignments/service-names-port-numbers/>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981,
<<https://www.rfc-editor.org/info/rfc793>>.

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [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>>.

- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

11.2. Informative References

- [AEAD-LIMITS]
Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", August 2017, <<http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>>.
- [github-dtn-bpbis-tcpcl]
Sipos, B., "TCPCL Example Implementation", <<https://github.com/BSipos-RKF/dtn-bpbis-tcpcl/>>.
- [I-D.ietf-dtn-bibect]
Burleigh, S., "Bundle-in-Bundle Encapsulation", draft-ietf-dtn-bibect-03 (work in progress), February 2020.
- [I-D.ietf-dtn-bpsec]
Birrane, E. and K. McKeever, "Bundle Protocol Security Specification", draft-ietf-dtn-bpsec-23 (work in progress), October 2020.
- [RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <<https://www.rfc-editor.org/info/rfc2595>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006, <<https://www.rfc-editor.org/info/rfc4511>>.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <<https://www.rfc-editor.org/info/rfc4838>>.
- [RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, DOI 10.17487/RFC5489, March 2009, <<https://www.rfc-editor.org/info/rfc5489>>.

- [RFC7122] Kruse, H., Jero, S., and S. Ostermann, "Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)", RFC 7122, DOI 10.17487/RFC7122, March 2014, <<https://www.rfc-editor.org/info/rfc7122>>.
- [RFC7242] Demmer, M., Ott, J., and S. Perreault, "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <<https://www.rfc-editor.org/info/rfc7242>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <<https://www.rfc-editor.org/info/rfc7435>>.
- [RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <<https://www.rfc-editor.org/info/rfc7457>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <<https://www.rfc-editor.org/info/rfc8555>>.

Appendix A. Significant changes from RFC7242

The areas in which changes from [RFC7242] have been made to existing headers and messages are:

- o Split Contact Header into pre-TLS protocol negotiation and SESS_INIT parameter negotiation. The Contact Header is now fixed-length.
- o Changed Contact Header content to limit number of negotiated options.

- o Added session option to negotiate maximum segment size (per each direction).
- o Renamed "Endpoint ID" to "Node ID" to conform with BPv7 terminology.
- o Added session extension capability.
- o Added transfer extension capability. Moved transfer total length into an extension item.
- o Defined new IANA registries for message / type / reason codes to allow renaming some codes for clarity.
- o Segments of all new IANA registries are reserved for private/experimental use.
- o Expanded Message Header to octet-aligned fields instead of bit-packing.
- o Added a bundle transfer identification number to all bundle-related messages (XFER_SEGMENT, XFER_ACK, XFER_REFUSE).
- o Use flags in XFER_ACK to mirror flags from XFER_SEGMENT.
- o Removed all uses of SDNV fields and replaced with fixed-bit-length (network byte order) fields.
- o Renamed SHUTDOWN to SESS_TERM to deconflict term "shutdown" related to TCP connections.
- o Removed the notion of a re-connection delay parameter.

The areas in which extensions from [RFC7242] have been made as new messages and codes are:

- o Added contact negotiation failure SESS_TERM reason code.
- o Added MSG_REJECT message to indicate an unknown or unhandled message was received.
- o Added TLS connection security mechanism.
- o Added Resource Exhaustion SESS_TERM reason code.

Authors' Addresses

Brian Sipos
RKF Engineering Solutions, LLC
7500 Old Georgetown Road
Suite 1275
Bethesda, MD 20814-6198
United States of America

Email: BSipos@rkf-eng.com

Michael Demmer
University of California, Berkeley
Computer Science Division
445 Soda Hall
Berkeley, CA 94720-1776
United States of America

Email: demmer@cs.berkeley.edu

Joerg Ott
Aalto University
Department of Communications and Networking
PO Box 13000
Aalto 02015
Finland

Email: ott@in.tum.de

Simon Perreault

Quebec, QC
Canada

Email: simon@per.reau.lt