DTN Research Group	S.F. Symington
Internet-Draft	The MITRE Corporation
Intended status: Experimental Protocol	S. Farrell
Expires: September 10, 2011	Trinity College Dublin
	H. Weiss
	P. Lovell
	SPARTA, Inc.
	March 09, 2011

Bundle Security Protocol Specification
draft-irtf-dtnrg-bundle-security-18

Abstract

This document defines the bundle security protocol, which provides data integrity and confidentiality services for the bundle protocol. Separate capabilities are provided to protect the bundle payload and additional data that may be included within the bundle. We also describe various bundle security considerations including policy options.
This document is a product of the Delay Tolerant Networking Research Group and has been reviewed by that group. No objections to its publication as an RFC were raised.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

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

This document defines security features for the bundle protocol [DTNBP] intended for use in delay tolerant networks, in order to provide Delay-Tolerant Networking (DTN) security services.

The bundle protocol is used in DTNs which overlay multiple networks, some of which may be challenged by limitations such as intermittent and possibly unpredictable loss of connectivity, long or variable delay, asymmetric data rates, and high error rates. The purpose of the bundle protocol is to support interoperability across such stressed networks. The bundle protocol is layered on top of underlay-network-specific convergence layers, on top of network-specific lower layers, to enable an application in one network to communicate with an application in another network, both of which are spanned by the DTN.

Security will be important for the bundle protocol. The stressed environment of the underlying networks over which the bundle protocol will operate makes it important for the DTN to be protected from unauthorized use, and this stressed environment poses unique challenges for the mechanisms needed to secure the bundle protocol. Furthermore, DTNs may very likely be deployed in environments where a portion of the network might become compromised, posing the usual security challenges related to confidentiality, integrity and availability.

Different security processing applies to the payload and extension blocks that may accompany it in a bundle, and different rules apply to various extension blocks.

This document describes both the base Bundle Security Protocol (BSP) and a set of mandatory ciphersuites. A ciphersuite is a specific collection of various cryptographic algorithms and implementation rules that are used together to provide certain security services.

The Bundle Security Protocol applies, by definition, only to those nodes that implement it, known as "security-aware" nodes. There MAY be other nodes in the DTN that do not implement BSP. All nodes can interoperate with the exception that BSP security operations can only happen at security-aware nodes.

1.1. Related Documents

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

* The Delay-Tolerant Network Architecture [DTNarch] defines the architecture for delay-tolerant networks, but does not discuss security at any length.

*The DTN Bundle Protocol [DTNBP] defines the format and processing of the blocks used to implement the bundle protocol, excluding the security-specific blocks defined here.

1.2. Terminology

We introduce the following terminology for purposes of clarity:

*source - the bundle node from which a bundle originates

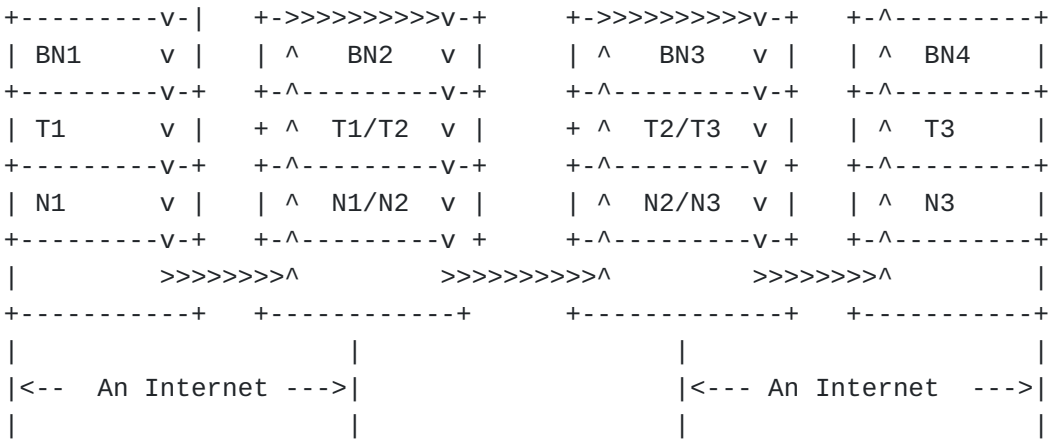
*destination - the bundle node to which a bundle is ultimately destined

*forwarder - the bundle node that forwarded the bundle on its most recent hop

*intermediate receiver or "next hop" - the neighboring bundle node to which a forwarder forwards a bundle.

*path - the ordered sequence of nodes through which a bundle passes on its way from source to destination

In the figure below, which is adapted from figure 1 in the Bundle Protocol Specification, four bundle nodes (denoted BN1, BN2, BN3, and BN4) reside above some transport layer(s). Three distinct transport and network protocols (denoted T1/N1, T2/N2, and T3/N3) are also shown.



BN = "Bundle Node" as defined in the Bundle Protocol Specification

Bundle node BN1 originates a bundle that it forwards to BN2. BN2 forwards the bundle to BN3, and BN3 forwards the bundle to BN4. BN1 is the source of the bundle and BN4 is the destination of the bundle. BN1 is the first forwarder, and BN2 is the first intermediate receiver; BN2 then becomes the forwarder, and BN3 the intermediate receiver; BN3 then

becomes the last forwarder, and BN4 the last intermediate receiver, as well as the destination.
If node BN2 originates a bundle (for example, a bundle status report or a custodial signal), which is then forwarded on to BN3, and then to BN4, then BN2 is the source of the bundle (as well as being the first forwarder of the bundle) and BN4 is the destination of the bundle (as well as being the final intermediate receiver).
We introduce the following security-specific DTN terminology:

- security-source - a bundle node that adds a security block to a bundle

- security-destination - a bundle node that processes a security block of a bundle

- security path - the ordered sequence of security-aware nodes through which a bundle passes on its way from the security-source to the security-destination

Referring to Figure 1 again:
If the bundle that originates at BN1 as source is given a security block by BN1, then BN1 is the security-source of this bundle with respect to that security block, as well as being the source of the bundle.
If the bundle that originates at BN1 as source is given a security block by BN2, then BN2 is the security-source of this bundle with respect to that security block, even though BN1 is the source.
If the bundle that originates at BN1 as source is given a security block by BN1 that is intended to be processed by BN3, then BN1 is the security-source and BN3 is the security destination with respect to this security block. The security path for this block is BN1 to BN3.
A bundle MAY have multiple security blocks. The security-source of a bundle with respect to a given security block in the bundle MAY be the same as or different from the security-source of the bundle with respect to a different security block in the bundle. Similarly, the security-destination of a bundle with respect to each of that bundle's security blocks MAY be the same or different. Therefore the security paths for various blocks MAY be and often will be different.
If the bundle that originates at BN1 as source is given a security block by BN1 that is intended to be processed by BN3, and BN2 adds a security block with security-destination BN4, the security paths for the two blocks overlap but not completely. This problem is discussed further in Section 3.3.
As required in [DTNBP], forwarding nodes MUST transmit blocks in a bundle in the same order in which they were received. This requirement applies to all DTN nodes, not just ones which implement security processing. Blocks in a bundle MAY be added or deleted according to the applicable specification, but those blocks which are both received and

transmitted MUST be transmitted in the same order that they were received.
If a node is not security-aware then it forwards the security blocks in the bundle unchanged unless the bundle's block processing flags specify otherwise. If a network has some nodes that are not security-aware then the block processing flags SHOULD be set such that security blocks are not discarded at those nodes solely because they can not be processed there. Except for this, the non-security-aware nodes are transparent relay points and are invisible as far as security processing is concerned.
The block sequence also indicates the order in which certain significant actions have affected the bundle, and therefore the sequence in which actions MUST occur in order to produce the bundle at its destination.

2. Security Blocks

There are four types of security block that MAY be included in a bundle. These are the Bundle Authentication Block (BAB), the Payload Integrity Block (PIB), the Payload Confidentiality Block (PCB) and the Extension Security Block (ESB).

- The BAB is used to assure the authenticity and integrity of the bundle along a single hop from forwarder to intermediate receiver. Since security blocks are only processed at security-aware nodes, a "single hop" from a security-aware forwarder to the next security-aware intermediate receiver might be more than one actual hop. This situation is discussed further below Section 2.2.

- The PIB is used to assure the authenticity and integrity of the payload from the PIB security-source, which creates the PIB, to the PIB security-destination, which verifies the PIB authenticator. The authentication information in the PIB MAY (if the ciphersuite allows) be verified by any node in between the PIB security-source and the PIB security-destination that has access to the cryptographic keys and revocation status information required to do so.

- Since a BAB protects a bundle on a "hop-by-hop" basis and other security blocks MAY be protecting over several hops or end-to-end, whenever both are present the BAB MUST form the "outer" layer of protection - that is, the BAB MUST always be calculated and added to the bundle after all other security blocks have been calculated and added to the bundle.

- The PCB indicates that the payload has been encrypted, in whole or in part, at the PCB security-source in order to protect the bundle content while in transit to the PCB security-destination.

*PIB and PCB protect the payload and are regarded as "payload-related" for purposes of the security discussion in this document. Other blocks are regarded as "non-payload" blocks. Of course, the primary block is unique and has separate rules.

*The ESB provides security for non-payload blocks in a bundle. ESB therefore is not applied to PIB or PCBs, and of course is not appropriate for either the payload block or primary block.

Each of the security blocks uses the Canonical Bundle Block Format as defined in the Bundle Protocol Specification. That is, each security block is comprised of the following elements:

*- Block type code

*- Block processing control flags

*- Block EID reference list (OPTIONAL)

*- Block data length

*- Block-type-specific data fields

Since the four security blocks have most fields in common, we can shorten the description of the Block-type-specific data fields of each security block if we first define an abstract security block (ASB) and then specify each of the real blocks in terms of the fields which are present/absent in an ASB. Note that no bundle ever contains an actual ASB, which is simply a specification artifact.

2.1. Abstract Security Block

Many of the fields below use the "SDNV" type defined in [DTNBP]. SDNV stands for Self-Delimiting Numeric Value.
An ASB consists of the following mandatory and optional fields:
- Block-type code (one byte) - as in all bundle protocol blocks except the primary bundle block. The block types codes for the security blocks are:

*BundleAuthenticationBlock - BAB: 0x02

*PayloadIntegrityBlock - PIB: 0x03

*PayloadConfidentialityBlock - PCB: 0x04

*ExtensionSecurityBlock - ESB: 0x09

- Block processing control flags (SDNV) - defined as in all bundle protocol blocks except the primary bundle block (as described in the Bundle Protocol [DTNBP]). SDNV encoding is described in the bundle

protocol. There are no general constraints on the use of the block processing flags, and some specific requirements are discussed later.
- EID references - composite field defined in [DTNBP] containing references to one or two EIDs. Presence of the EID-reference field is indicated by the setting of the "block contains an EID-reference field" (EID_REF) bit of the block processing control flags. If one or more references is present, flags in the ciphersuite ID field, described below, specify which.
If no EID fields are present then the composite field itself MUST be omitted entirely and the EID_REF bit MUST be unset. A count field of zero is not permitted.
The possible EIDs are:

    *- (OPTIONAL) Security-source - specifies the security source for the block. If this is omitted, then the source of the bundle is assumed to be the security-source unless otherwise indicated.

    *- (OPTIONAL) Security-destination - specifies the security destination for the block. If this is omitted, then the destination of the bundle is assumed to be the security-destination unless otherwise indicated.

If two EIDs are present, security-source is first and security-destination comes second.
- Block data length (SDNV) - as in all bundle protocol blocks except the primary bundle block. SDNV encoding is described in the bundle protocol.
- Block-type-specific data fields as follows:

    *- Ciphersuite ID (SDNV)

    *- Ciphersuite flags (SDNV)

    *- (OPTIONAL) Correlator - when more than one related block is inserted then this field MUST have the same value in each related block instance. This is encoded as an SDNV. See note in Section 3.8 with regard to correlator values in bundle fragments.

    *- (OPTIONAL) Ciphersuite parameters - compound field of next two items

        -- Ciphersuite parameters length - specifies the length of the following Ciphersuite parameters data field and is encoded as an SDNV.

        -- Ciphersuite parameters data - parameters to be used with the ciphersuite in use, e.g. a key identifier or initialization vector (IV). See Section 2.6 for a list of potential parameters and their encoding rules. The particular set of

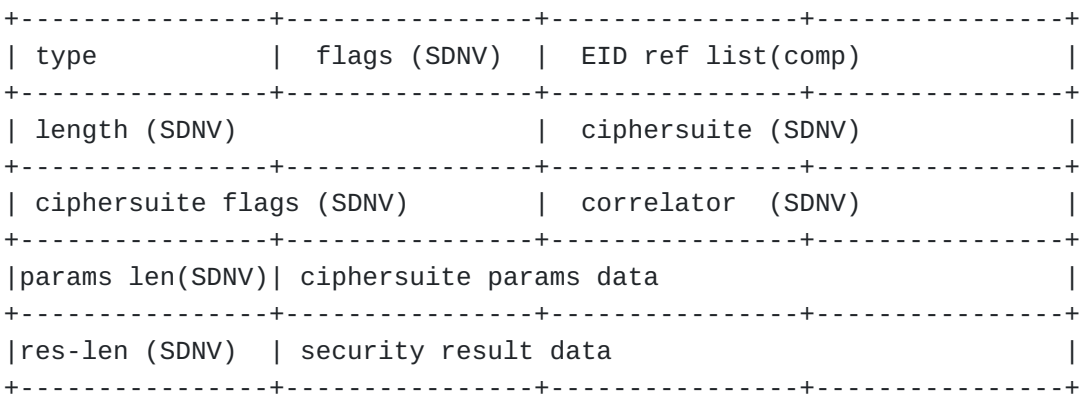
parameters that are included in this field are defined as part of the ciphersuite specification.

*- (OPTIONAL) Security result - compound field of next two items

-- Security result length - contains the length of the next field and is encoded as an SDNV.

-- Security result data - contains the results of the appropriate ciphersuite-specific calculation (e.g., a signature, MAC or ciphertext block key).

Although the diagram hints at a 32-bit layout this is purely for the purpose of exposition. Except for the "type" field, all fields are variable in length.



Some ciphersuites are specified in Section 4, which also specifies the rules which MUST be satisfied by ciphersuite specifications. Additional ciphersuites MAY be defined in separate specifications. Ciphersuite IDs not specified are reserved. Implementations of the bundle security protocol decide which ciphersuites to support, subject to the requirements of Section 4. It is RECOMMENDED that implementations that allow additional ciphersuites permit ciphersuite ID values at least up to and including 127, and they MAY decline to allow larger ID values. The structure of the ciphersuite flags field is shown in Figure 3. In each case the presence of an optional field is indicated by setting the value of the corresponding flag to one. A value of zero indicates the corresponding optional field is missing. Presently there are five flags defined for the field and for convenience these are shown as they would be extracted from a single-byte SDNV. Future additions may cause the field to grow to the left so, as with the flags fields defined in [DTNBP], the description below numbers the bit positions from the right rather than the standard RFC definition which numbers bits from the left.

*src - bit 4 indicates whether the EID-reference field of the ASB contains the optional reference to the security-source.

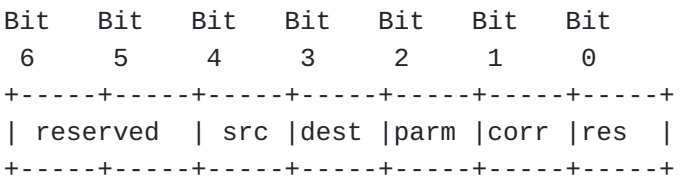
*dest - bit 3 indicates whether the EID-reference field of the ASB contains the optional reference to the security-destination.

*parm - bit 2 indicates whether the ciphersuite-parameters-length and ciphersuite parameters data fields are present or not.

*corr - bit 1 indicates whether or not the ASB contains an optional correlator.

*res - bit 0 indicates whether or not the ASB contains the security result length and security result data fields.

*bits 5-6 are reserved for future use.



A little bit more terminology: if the block is a PIB then when we refer to the "PIB-source", we mean the security source for the PIB as represented by the EID reference in the EID-references field. Similarly we may refer to the PCB-dest, meaning the security-destination of the PCB, again as represented by an EID reference. For example, referring to Figure 1 again, if the bundle that originates at BN1 as source is given a Confidentiality Block (PCB) by BN1 that is protected using a key held by BN3 and it is given a Payload Integrity Block (PIB) by BN1, then BN1 is both the PCB-source and the PIB-source of the bundle, and BN3 is the PCB-dest of the bundle.

The correlator field is used to associate several related instances of a security block. This can be used to place a BAB that contains the ciphersuite information at the "front" of a (probably large) bundle, and another correlated BAB that contains the security result at the "end" of the bundle. This allows even very memory-constrained nodes to be able to process the bundle and verify the BAB. There are similar use cases for multiple related instances of PIB and PCB as will be seen below.

The ciphersuite specification MUST make it clear whether or not multiple block instances are allowed, and if so, under what conditions. Some ciphersuites can of course leave flexibility to the implementation, whereas others might mandate a fixed number of instances.

For convenience, we use the term "first block" to refer to the initial block in a group of correlated blocks, or to the single block if there are no others in the set. Obviously there can be several unrelated

groups in a bundle, each containing only one block or more than one, and each has its own "first block".

2.2. Bundle Authentication Block

In this section we describe typical BAB field values for two scenarios - where a single instance of the BAB contains all the information and where two related instances are used, one "up front" which contains the ciphersuite and another following the payload which contains the security result (e.g. a MAC).
For the case where a single BAB is used:

- The block-type code field value MUST be 0x02.

- The block processing control flags value can be set to whatever values are required by local policy. Ciphersuite 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.

- The ciphersuite ID MUST be documented as a hop-by-hop authentication-ciphersuite which requires one instance of the BAB.

- The correlator field MUST NOT be present.

- The ciphersuite parameters field MAY be present, if so specified in the ciphersuite specification.

- An EID reference to the security-source MAY be present. The security-source can also be specified as part of key information described in Section 2.6 or another block such as the Previous Hop Insertion Block [PHIB]. The security-source might also be inferred from some implementation-specific means such as the convergence layer.

- An EID reference to the security-destination MAY be present and is useful to ensure that the bundle has been forwarded to the correct next-hop node.

- The security result MUST be present as it is effectively the "output" from the ciphersuite calculation (e.g. the MAC or signature) applied to the (relevant parts of) the bundle (as specified in the ciphersuite definition).

For the case using two related BAB instances, the first instance is as defined above, except the ciphersuite ID MUST be documented as a hop-by-hop authentication ciphersuite that requires two instances of the BAB. In addition, the correlator MUST be present and the security result length and security result fields MUST be absent. The second

instance of the BAB MUST have the same correlator value present and MUST contain security result length and security result data fields. The other optional fields MUST NOT be present. Typically, this second instance of a BAB will be the last block of the bundle.
The details of key transport for BAB are specified by the particular ciphersuite. In the absence of conflicting requirements, the following should be noted by implementors:
- the key information item Section 2.6 is OPTIONAL, and if not provided then the key SHOULD be inferred from the source-destination tuple, being the previous key used, a key created from a key-derivation function, or a pre-shared key
- if all the nodes are security-aware, the capabilities of the underlying convergence layer might be useful for identifying the security-source
- depending upon the key mechanism used, bundles can be signed by the sender, or authenticated for one or more recipients, or both.

2.3. Payload Integrity Block

A PIB is an ASB with the following additional restrictions:

*The block type code value MUST be 0x03.

*The block processing control flags value can be set to whatever values are required by local policy. Ciphersuite 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.

*The ciphersuite ID MUST be documented as an end-to-end authentication-ciphersuite or as an end-to-end error-detection-ciphersuite.

*The correlator MUST be present if the ciphersuite requires more than one related instance of a PIB be present in the bundle. The correlator MUST NOT be present if the ciphersuite only requires one instance of the PIB in the bundle.

*The ciphersuite parameters field MAY be present.

*An EID reference to the security-source MAY be present. The security-source can also be specified as part of key information described in Section 2.6.

*An EID reference to the security-destination MAY be present.

*The security result is effectively the "output" from the ciphersuite calculation (e.g. the MAC or signature) applied to the (relevant parts of) the bundle. As in the case of the BAB, this field MUST be present if the correlator is absent. If more

than one related instance of the PIB is required then this is handled in the same way as described for the BAB above.

*The ciphersuite MAY process less than the entire original bundle payload. This might be because it is defined to process some subset of the bundle, or perhaps because the the current payload is a fragment of an original bundle. For whatever reason, if the ciphersuite processes less than the complete, original bundle payload, the ciphersuite parameters of this block MUST specify which bytes of the bundle payload are protected.

For some ciphersuites, (e.g. those using asymmetric keying to produce signatures or those using symmetric keying with a group key), the security information can be checked at any hop on the way to the security destination that has access to the required keying information. This possibility is further discussed in Section 3.6 below.
The use of a generally-available key is RECOMMENDED if custodial transfer is employed and all nodes SHOULD verify the bundle before accepting custody.
Most asymmetric PIB-ciphersuites will use the PIB-source to indicate the signer and will not require the PIB-dest field because the key needed to verify the PIB authenticator will be a public key associated with the PIB-source.

2.4. Payload Confidentiality Block

A typical confidentiality ciphersuite will encrypt the payload using a randomly generated bundle encrypting key (BEK) and will use a key information item in the PCB security parameters to carry the BEK encrypted with some long term key encryption key (KEK) or well-known public key. If neither the destination nor security-destination resolves the key to use for decryption, the key information item in the ciphersuite parameters field can also be used to indicate the decryption key with which the BEK can be recovered. If the bundle already contains PIBs and/or PCBs these SHOULD also be encrypted using this same BEK, as described just below for "super-encryption". The encrypted block is encapsulated into a new PCB that replaces the original block at the same place in the bundle.
It is strongly RECOMMENDED that a data integrity mechanism be used in conjunction with confidentiality, and that encryption-only ciphersuites NOT be used. AES-GCM satisfies this requirement. The "authentication tag" or "integrity check value" is stored into security-result rather than being appended to the payload as is common in some protocols since, as described below, it is important that there be no change in the size of the payload.
The payload is encrypted "in-place", that is, following encryption, the payload block payload field contains ciphertext, not plaintext. The payload block processing flags are unmodified.

The "in-place" encryption of payload bytes is to allow bundle payload fragmentation and re-assembly, and custody transfer, to operate without knowledge of whether or not encryption has occurred and, if so, how many times.

Fragmentation and reassembly and custody transfer are adversely affected by a change in size of the payload due to ambiguity about what byte range of the original payload is actually in any particular fragment. Ciphersuites SHOULD place any payload expansion, such as authentication tags (integrity check values) and any padding generated by a block-mode cipher, into an "integrity check value" item in the security-result field (see Section 2.6) of the confidentiality block.

Payload super-encryption is allowed; that is, encrypting a payload that has already been encrypted, perhaps more than once. Ciphersuites SHOULD define super-encryption such that, as well as re-encrypting the payload, it also protects the parameters of earlier encryption. Failure to do so may represent a vulnerability in some circumstances.

Confidentiality is normally applied to the payload, and possibly to additional blocks. It is RECOMMENDED to apply a Payload Confidentiality ciphersuite to non-payload blocks only if these SHOULD be super-encrypted with the payload. If super-encryption of the block is not desired then protection of the block SHOULD be done using the Extension Security Block mechanism rather than PCB.

Multiple related PCB instances are required if both the payload and PIBs and PCBs in the bundle are to be encrypted. These multiple PCB instances require correlators to associate them with each other since the key information is provided only in the first PCB.

There are situations where more than one PCB instance is required but the instances are not "related" in the sense which requires correlators. One example is where a payload is encrypted for more than one security-destination so as to be robust in the face of routing uncertainties. In this scenario the payload is encrypted using a BEK. Several PCBs contain the BEK encrypted using different KEKs, one for each destination. These multiple PCB instances, are not "related" and SHOULD NOT contain correlators.

The ciphersuite MAY apply different rules to confidentiality for non-payload blocks.

A PCB is an ASB with the following additional restrictions:

*The block type code value MUST be 0x04.

*The block processing control flags value can be set to whatever values are required by local policy, except that a PCB "first block" MUST have the "replicate in every fragment" flag set. This flag SHOULD NOT be set otherwise. Ciphersuite 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.

*The ciphersuite ID MUST be documented as a confidentiality-ciphersuite.

*The correlator MUST be present if there is more than one related PCB instance. The correlator MUST NOT be present if there are no related PCB instances.

*If a correlator is present, the key information MUST be placed in the PCB "first block".

*Any additional bytes generated as a result of encryption and/or authentication processing of the payload SHOULD be placed in an "integrity check value" field (see Section 2.6) in the security-result of the first PCB.

*The ciphersuite parameters field MAY be present.

*An EID reference to the security-source MAY be present. The security-source can also be specified as part of key information described in Section 2.6.

*An EID reference to the security-destination MAY be present.

*The security result MAY be present and normally contains fields such as an encrypted bundle encryption key, authentication tag or the encrypted versions of bundle blocks other than the payload block.

The ciphersuite MAY process less than the entire original bundle payload, either because the current payload is a fragment of the original bundle or just because it is defined to process some subset. For whatever reason, if the ciphersuite processes less than the complete, original bundle payload the "first" PCB MUST specify, as part of the ciphersuite parameters, which bytes of the bundle payload are protected.
PCB ciphersuites MUST specify which blocks are to be encrypted. The specification MAY be flexible and be dependent upon block type, security policy, various data values and other inputs but it MUST be deterministic. The determination of whether a block is to be encrypted or not MUST NOT be ambiguous.
As was the case for the BAB and PIB, if the ciphersuite requires more than one instance of the PCB, then the "first block" MUST contain any optional fields (e.g., security destination etc.) that apply to all instances with this correlator. These MUST be contained in the first instance and MUST NOT be repeated in other correlated blocks. Fields that are specific to a particular instance of the PCB MAY appear in that PCB. For example, security result fields MAY (and probably will) be included in multiple related PCB instances, with each result being specific to that particular block. Similarly, several PCBs might each

contain a ciphersuite parameters field with an IV specific to that PCB instance.

Put another way: when confidentiality will generate multiple blocks, it MUST create a "first" PCB with the required ciphersuite ID, parameters etc. as specified above. Typically, this PCB will appear early in the bundle. This "first" PCB contains the parameters that apply to the payload and also to the other correlated PCBs. The correlated PCBs follow the "first" PCB and MUST NOT repeat the ciphersuite parameters, security-source, or security-destination fields from the first PCB. These correlated PCBs need not follow immediately after the "first" PCB, and probably will not do so. Each correlated block, encapsulating an encrypted PIB or PCB, is at the same place in the bundle as the original PIB or PCB.

A ciphersuite MUST NOT mix payload data and a non-payload block in a single PCB.

Even if a to-be-encrypted block has the "discard" flag set, whether or not the PCB's "discard" flag is set is an implementation/policy decision for the encrypting node. (The "discard" flag is more properly called the "discard if block cannot be processed" flag.)

Any existing EID-list in the to-be-encapsulated original block remains exactly as-is, and is copied to become the EID-list for the replacing block. The encapsulation process MUST NOT replace or remove the existing EID-list entries. This is critically important for correct updating of entries at the security-destination.

At the security-destination, either specific destination or the bundle destination, the processes described above are reversed. The payload is decrypted in-place using the salt, IV and key values in the first PCB, including verification using the ICV. These values are described below in Section 2.6. Each correlated PCB is also processed at the same destination, using the salt and key values from the first PCB and the block-specific IV item. The "encapsulated block" item in the security-result is decrypted and validated, using also the tag which SHOULD have been appended to the ciphertext of the original block data. Assuming the validation succeeds, the resultant plaintext, which is the entire content of the original block, replaces the PCB at the same place in the bundle. The block type reverts to that of the original block prior to encapsulation, and the other block-specific data fields also return to their original values. Implementors are cautioned that this "replacement" process requires delicate stitchery, as the EID-list contents in the decapsulated block are invalid. As noted above, the EID-list references in the original block were preserved in the replacing PCB, and will have been updated as necessary as the bundle has toured the dtn. The references from the PCB MUST replace the references within the EID-list of the newly-decapsulated block. Caveat implementor.

2.5. Extension Security Block

Extension security blocks provide protection for non-payload-related portions of a bundle. ESBs MUST NOT be used for the primary block or payload, including payload-related security blocks (PIBs and PCBs).
It is sometimes desirable to protect certain parts of a bundle in ways other than those applied to the bundle payload. One such example is bundle metadata that might specify the kind of data in the payload but not the actual payload detail, as described in [DTNMD].
ESBs are typically used to apply confidentiality protection. While it is possible to create an integrity-only ciphersuite, the block protection is not transparent and makes access to the data more difficult. For simplicity, this discussion describes use of a confidentiality ciphersuite.
The protection mechanisms in ESBs are similar to other security blocks with two important differences:

*- different key values are used (using same key as for payload would defeat the purpose)

*- the block is not encrypted or super-encrypted with the payload

A typical ESB ciphersuite will encrypt the extension block using a randomly generated ephemeral key and will use the key information item in the security parameters field to carry the key encrypted with some long term key encryption key (KEK) or well-known public key. If neither the destination nor security-destination resolves the key to use for decryption, the key information item in the ciphersuite parameters field can be used also to indicate the decryption key with which the BEK can be recovered.
It is strongly RECOMMENDED that a data integrity mechanism be used in conjunction with confidentiality, and that encryption-only ciphersuites NOT be used. AES-GCM satisfies this requirement.
The ESB is placed in the bundle in the same position as the block being protected. That is, the entire original block is processed (encrypted, etc) and encapsulated in a "replacing" ESB-type block, and this appears in the bundle at the same sequential position as the original block.
The processed data is placed in the security-result field.
The process is reversed at the security destination with the recovered plaintext block replacing the ESB that had encapsulated it. Processing of EID-list entries, if any, is described above in Section 2.4 and this MUST be followed in order to correctly recover EIDs.
An ESB is an ASB with the following additional restrictions:

*Block type is 0x09.

*Ciphersuite flags indicate which fields are present in this block. Ciphersuite 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.

*EID references MUST be stored in the EID reference list.

*Security-source MAY be present. The security-source can also be specified as part of key information described in Section 2.6. If neither is present then the bundle-source is used as the security-source.

*Security-destination MAY be present. If not present, then the bundle-destination is used as the security-destination.

The security-parameters MAY optionally contain a block-type field to indicate the type of the encapsulated block. Since this replicates a field in the encrypted portion of the block, it is a slight security risk and its use is therefore OPTIONAL.

2.6. Parameters and Result Fields

Various ciphersuites include several items in the security-parameters and/or security-result fields. Which items MAY appear is defined by the particular ciphersuite description. A ciphersuite MAY support several instances of the same type within a single block.
Each item is represented as type-length-value. Type is a single byte indicating which item this is. Length is the count of data bytes to follow, and is an SDNV-encoded integer. Value is the data content of the item.
Item types are

*0: reserved

*1: initialization vector (IV)

*2: reserved

*3: key information

*4: fragment range (offset and length as a pair of SDNVs)

*5: integrity signature

*6: reserved

*7: salt

*8: PCB integrity check value (ICV)

*9: reserved

*10: encapsulated block

*11: block type of encapsulated block

*12 - 191: reserved

*192 - 250: private use

*251 - 255: reserved

The folowing descriptions apply to usage of these items for all ciphersuites. Additional characteristics are noted in the discussion for specific suites.

*- initialization vector(IV): random value, typically eight to sixteen bytes

*- key information: key material encoded or protected by the key management system, and used to transport an ephemeral key protected by a long-term key. This item is discussed further below in Section 2.7

*- fragment range: pair of SDNV values (offset then length) specifying the range of payload bytes to which a particular operation applies. This is termed "fragment range" since that is its typical use, even though sometimes it describes a subset range that is not a fragment. The offset value MUST be the offset within the original bundle, which might not be the offset within the current bundle if the current bundle is already a fragment

*- integrity signature: result of BAB or PIB digest or signing operation. This item is discussed further below in Section 2.7

*- salt: an IV-like value used by certain confidentiality suites

*- PCB integrity check value(ICV): output from certain confidentiality ciphersuite operations to be used at the destination to verify that the protected data has not been modified

*- encapsulated block: result of confidentiality operation on certain blocks, contains the ciphertext of the block and MAY also contain an integrity check value appended to the ciphertext; MAY also contain padding if required by the encryption mode; used for non-payload blocks only

*- block type of encapsulated block: block type code for a block that has been encapsulated in ESB

2.7. Key Transport

This specification endeavours to maintain separation between the security protocol and key management. However, these two interact in the transfer of key information, etc., from security-source to security-destination. The intent of the separation is to facilitate use of a variety of key management systems without a necessity to tailor a ciphersuite to each individually.
The key management process deals with such things as long-term keys, specifiers for long-term keys, certificates for long-term keys and integrity signatures using long-term keys. The ciphersuite itself SHOULD NOT require a knowledge of these, and separation is improved if it treats these as opaque entities, to be handled by the key management process.
The key management process deals specifically with the content of two of the items defined above in Section 2.6:- key information (item type 3) and integrity signature (item type 5). The ciphersuite MUST define the details and format for these items. To facilitate interoperability, it is strongly RECOMMENDED that the implementations use the appropriate definitions from Cryptographic Message Syntax (CMS) [RFC5652] and related RFCs.
Many situations will require several pieces of key information. Again, ciphersuites MUST define whether they accept these packed into a single key information item and/or separated into multiple instances of key information. For interoperability, it is RECOMMENDED that ciphersuites accept these packed into a single key-information item, and that they MAY additionally choose to accept them sent as separate items.

2.8. PIB and PCB combinations

Given the above definitions, nodes are free to combine applications of PIB and PCB in any way they wish - the correlator value allows for multiple applications of security services to be handled separately. Since PIB and PCB apply to the payload and ESB to non-payload blocks, combinations of ESB with PIB and/or PCB are not considered.
There are some obvious security problems that could arise when applying multiple services. For example, if we encrypted a payload but left a PIB security result containing a signature in the clear, payload guesses could be confirmed.
We cannot, in general, prevent all such problems since we cannot assume that every ciphersuite definition takes account of every other ciphersuite definition. However, we can limit the potential for such problems by requiring that any ciphersuite which applies to one instance of a PIB or PCB, MUST be applied to all instances with the same correlator.

We now list the PIB and PCB combinations which we envisage as being useful to support:

* Encrypted tunnels - a single bundle MAY be encrypted many times en-route to its destination. Clearly it has to be decrypted an equal number of times, but we can imagine each encryption as representing the entry into yet another layer of tunnel. This is supported by using multiple instances of PCB, but with the payload encrypted multiple times, "in-place". Depending upon the ciphersuite defintion, other blocks can and should be encrypted, as discussed above and in Section 2.4 to ensure that parameters are protected in the case of super-encryption.

* Multiple parallel authenticators - a single security source might wish to protect the integrity of a bundle in multiple ways. This could be required if the bundle's path is unpredictable, and if various nodes might be involved as security destinations. Similarly, if the security source cannot determine in advance which algorithms to use, then using all might be reasonable. This would result in uses of PIB which presumably all protect the payload, and which cannot in general protect one another. Note that this logic can also apply to a BAB, if the unpredictable routing happens in the convergence layer, so we also envisage support for multiple parallel uses of BAB.

* Multiple sequential authenticators - if some security destination requires assurance about the route that bundles have taken, then it might insist that each forwarding node add its own PIB. More likely, however would be that outbound "bastion" nodes would be configured to sign bundles as a way of allowing the sending "domain" to take accountability for the bundle. In this case, the various PIBs will likely be layered, so that each protects the earlier applications of PIB.

* Authenticated and encrypted bundles - a single bundle MAY require both authentication and confidentiality. Some specifications first apply the authenticator and follow this by encrypting the payload and authenticator. As noted previously in the case where the authenticator is a signature, there are security reasons for this ordering. (See the PCB-RSA-AES128-PAYLOAD-PIB-PCB ciphersuite defined later in Section 4.3.) Others apply the authenticator after encryption, that is, to the ciphertext. This ordering is generally RECOMMENDED and minimizes attacks which, in some cases, can lead to recovery of the encryption key.

There are no doubt other valid ways to combine PIB and PCB instances, but these are the "core" set supported in this specification. Having said that, as will be seen, the mandatory ciphersuites defined here are quite specific and restrictive in terms of limiting the flexibility

offered by the correlator mechanism. This is primarily designed to keep this specification as simple as possible, while at the same time supporting the above scenarios.

3. Security Processing

This section describes the security aspects of bundle processing.

3.1. Nodes as policy enforcement points

All nodes are REQUIRED to have and enforce their own configurable security policies, whether these policies be explicit or default, as defined in Section 6.
All nodes serve as Policy Enforcement Points (PEP) insofar as they enforce polices that MAY restrict the permissions of bundle nodes to inject traffic into the network. Policies MAY apply to traffic originating at the current node, traffic terminating at the current node and traffic to be forwarded by the current node to other nodes. If a particular transmission request, originating either locally or remotely, satisfies the node's policy or policies and is therefore accepted, then an outbound bundle can be created and dispatched. If not, then in its role as a PEP, the node will not create or forward a bundle. Error handling for such cases is currently considered out of scope of this document.
Policy enforcing code MAY override all other processing steps described here and elsewhere in this document. For example, it is valid to implement a node which always attempts to attach a PIB. Similarly it is also valid to implement a node which always rejects all requests which imply the use of a PIB.
Nodes MUST consult their security policy to determine the criteria that a received bundle ought to meet before it will be forwarded. These criteria MUST include a determination of whether or not the received bundle MUST include a valid BAB, PIB, PCB or ESB. If the bundle does not meet the node's policy criteria, then the bundle MUST be discarded and processed no further; in this case, a bundle status report indicating the failure MAY be generated.
The node's policy MAY call for the node to add or subtract some security blocks. For example, it might require the node attempt to encrypt (parts of) the bundle for some security-destination, or that it add a PIB. If the node's policy requires a BAB to be added to the bundle, it MUST be added last so that the calculation of its security result MAY take into consideration the values of all other blocks in the bundle.

3.2. Processing order of security blocks

The processing order of security actions for a bundle is critically important for the actions to complete successfully. In general, the actions performed at the originating node MUST be executed in the

reverse sequence at the destination. There are variations and exceptions, and these are noted below.

The sequence is maintained in the ordering of security blocks in the bundle. It is for this reason that blocks MUST NOT be rearranged at forwarding nodes, whether they support the security protocols or not. The only blocks that participate in this ordering are the primary and payload blocks, and the PIB and PCB security blocks themselves. All other extension blocks, including ESBs, are ignored for purposes of determining the processing order.

The security blocks are added to and removed from a bundle in a last-in-first-out (LIFO) manner, with the top of the stack immediately after the primary block. A newly-created bundle has just the primary and payload blocks, and the stack is empty. As security actions are requested for the bundle, security blocks are pushed onto the stack immediately after the primary block. The early actions have security blocks close to the payload, later actions have blocks nearer to the primary block. The actions deal with only those blocks in the bundle at the time so, for example, the first to be added processes only the payload and primary blocks, the next might process the first if it chooses and the payload and primary, and so on. The last block to be added can process all the blocks.

When the bundle is received, this process is reversed and security processing begins at the top of the stack, immediately after the primary block. The security actions are performed and the block is popped from the stack. Processing continues with the next security block until finally only the payload and primary blocks remain.

The simplicity of this description is undermined by various real-world requirements. Nonetheless it serves as a helpful initial framework for understanding the bundle security process.

The first issue is a very common one and easy to handle. The bundle may be sent indirectly to its destination, requiring several forwarding hops to finally arrive there. Security processing happens at each node, assuming that the node supports bundle security. For the following discussion, we assume that a bundle is created and that confidentiality, then payload integrity and finally bundle authentication are applied to it. The block sequence would therefore be primary-BAB-PIB-PCB-payload. Traveling from source to destination requires going through one intermediate node, so the trip consists of two hops.

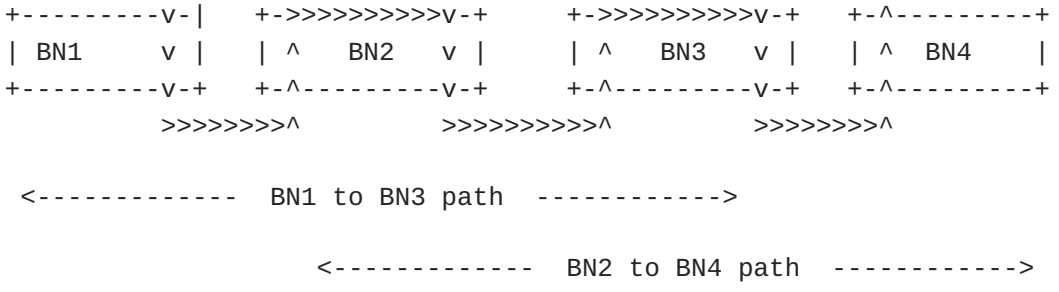
When the bundle is received at the intermediate node, the receive processing validates the BAB and pops it from the stack. However the PIBs and PCBs have the final destination as their security destination, so these can't be processed and removed. The intermediate node then begins the send process with the four remaining blocks in the bundle. The outbound processing adds any security blocks required by local policy, and these are pushed on the stack immediately after the primary block, ahead of the PIB. In this example, the intermediate node adds a PIB as a signature that the bundle has passed through the node.

The receive processing at the destination first handles the intermediate node's PIB and pops it, next is the originator's PIB, also popped, and finally the originator's confidentiality block which allows the payload to be decrypted and the bundle handled for delivery.
DTNs in practice are likely to be more complex. The security policy for a node specifies the security requirements for a bundle. The policy will possibly cause one or more security operations to be applied to the bundle at the current node, each with its own security-destination. Application of policy at subsequent nodes might cause additional security operations, each with a security-destination. The list of security-destinations in the security blocks (BAB, PIB and PCB, not ESB) creates a partial-ordering of nodes that MUST be visited en route to the bundle destination.
The bundle security scheme does not deal with security paths that overlap partially but not completely. The security policy for a node MUST avoid specifying for a bundle a security-destination that causes a conflict with any existing security-destination in that bundle. This is discussed further below in Section 3.3.
The second issue relates to the reversibility of certain security process actions. In general, the actions fall into two categories: those which do not affect other parts of the bundle, and those which are fully reversible. Creating a bundle signature, for example, does not change the bundle content except for the result. The encryption performed as part of the confidentiality processing does change the bundle, but the reverse processing at the destination restores the original content.
The third category is the one where the bundle content has changed slightly and in a non-destructive way, but there is no mechanism to reverse the change. The simplest example is the addition of an EID-reference to a security block. The addition of the reference causes the text to be added to the bundle's dictionary. The text may be used also by other references so removal of the block and this specific EID-reference does not cause removal of the text from the dictionary. This shortcoming is of no impact to the "sequential" or "wrapping" security schemes described above, but does cause failures with "parallel" authentication mechanisms. Solutions for this problem are implementation-specific and typically involve multi-pass processing such that blocks are added at one stage and the security results calculated at a later stage of the overall process.
Certain ciphersuites have sequence requirements for their correct operation, most notably the bundle authentication ciphersuites. Processing for bundle authentication is required to happen after all other sending operations, and prior to any receive operations at the next hop node. It follows therefore that BABs MUST always be pushed onto the stack after all others.
Although we describe the security block list as a stack, there are some blocks which are placed after the payload and therefore are not part of the stack. The BundleAuthentication ciphersuite #1 ("BA1") requires a second, correlated block to contain the security-result and this block

is placed after the payload, usually as the last block in the bundle. We can apply the stack rules even to these blocks by specifying that they be added to the end of the bundle at the same time that their "owner" or "parent" block is pushed on the stack. In fact, they form a stack beginning at the payload but growing in the other direction. Also, not all blocks in the main stack have a corresponding entry in the trailing stack. The only blocks which MUST follow the payload are those mandated by ciphersuites as correlated blocks for holding a security-result. No other blocks are required to follow the payload block and it is NOT RECOMMENDED that they do so.
ESBs are effectively placeholders for the blocks they encapsulate and, since those do not form part of the processing sequence described above, ESBs themselves do not either. ESBs MAY be correlated, however, so the "no reordering" requirement applies to them as well.

3.3. Security Regions

Each security block has a security path, as described in the discussion for Figure 1, and the paths for various blocks are often different. BABs are always for a single hop and these restricted paths never cause conflict.
The paths for PIBs and PCBs are often from bundle source to bundle destination, to provide end-to-end protection. A bundle-source-to-bundle-destination path likewise never causes a problem.
Another common scenario is for gateway-to-gateway protection of traffic between two sub-networks ("tunnel-mode").
Looking at Figure 1 and the simplified version shown in Figure 4, we can regard BN2 and BN3 as gateways connecting the two subnetworks labeled "An internet". As long as they provide security for the BN2-BN3 path, all is well. Problems begin, for example, when BN2 adds blocks with BN4 as the security-destination, and originating node BN1 has created blocks with BN3 as security-destination. We now have two paths and neither is a subset of the other.
This scenario should be prevented by node BN2's security policy being aware of the already-existing block with BN3 as the security destination. This policy SHOULD NOT specify a security-dest that is further distant than any existing security-dest.



Consider the case where the security concern is for data integrity, so the blocks are PIBs. BN1 creates one ("PIa") along with the new bundle, and BN2 pushes its own PIB "PIb" on the stack, with security-destination BN4. When this bundle arrives at BN3, the bundle blocks are The situation would be worse if the security concern is confidentiality, and PCBs are employed, using the confidentiality ciphersuite #3 ("PC3") described in Section 4.3. In this scenario, BN1 would encrypt the bundle with BN3 as security-destination, BN2 would create an overlapping security path by super-encrypting the payload and encapsulating the PC3 block for security-destination BN4. BN3 forwards all the blocks without change. BN4 decrypts the payload from its super-encryption and decapsulates the PC3 block, only to find that it should have been processed earlier. Assuming that BN4 has no access to BN3's key store, BN4 has no way to decrypt the bundle and recover the original content.
As mentioned above, authors of security policy need to use care to ensure that their policies do not cause overlaps. These guidelines should prove helpful:

- the originator of a bundle can always specify the bundle-dest as the security-dest, and should be cautious about doing otherwise

- in the "tunnel-mode" scenario where two sub-networks are connected by a tunnel through a network, the gateways can each specify the other as security-dest, and should be cautious about doing otherwise

- BAB is never a problem because it is always only a single hop

- PIB for a bundle without PCB will usually specify the bundle destination as security-dest

- PIB for a bundle containing a PCB should specify as its security-dest the security-dest of the PCB (outermost PCB if there are more than one)

3.4. Canonicalisation of bundles

In order to verify a signature or MAC on a bundle the exact same bits, in the exact same order, MUST be input to the calculation upon verification as were input upon initial computation of the original signature or MAC value. Consequently, a node MUST NOT change the encoding of any URI [RFC3986] in the dictionary field, e.g., changing the DNS part of some HTTP URL from lower case to upper case. Because bundles MAY be modified while in transit (either correctly or due to implementation errors), a canonical form of any given bundle (that contains a BAB or PIB) MUST be defined.

This section defines bundle canonicalisation algorithms used in the Section 4.1 and Section 4.2 ciphersuites. Other ciphersuites can use these or define their own canonicalization procedures.

3.4.1. Strict canonicalisation

The first algorithm that can be used permits no changes at all to the bundle between the security-source and the security-destination. It is mainly intended for use in BAB ciphersuites. This algorithm conceptually catenates all blocks in the order presented, but omits all security result data fields in blocks of this ciphersuite type. That is, when a BAB ciphersuite specifies this algorithm then we omit all BAB security results for all BAB ciphersuites, when a PIB ciphersuite specifies this algorithm then we omit all PIB security results for all PIB ciphersuites. All security result length fields are included, even though their corresponding security result data fields are omitted.
Notes:

*- In the above we specify that security result data is omitted. This means that no bytes of the security result data are input. We do not set the security result length to zero. Rather, we assume that the security result length will be known to the module that implements the ciphersuite before the security result is calculated, and require that this value be in the security result length field even though the security result data itself will be omitted.

*- The 'res' bit of the ciphersuite ID, which indicates whether or not the security result length and security result data field are present, is part of the canonical form.

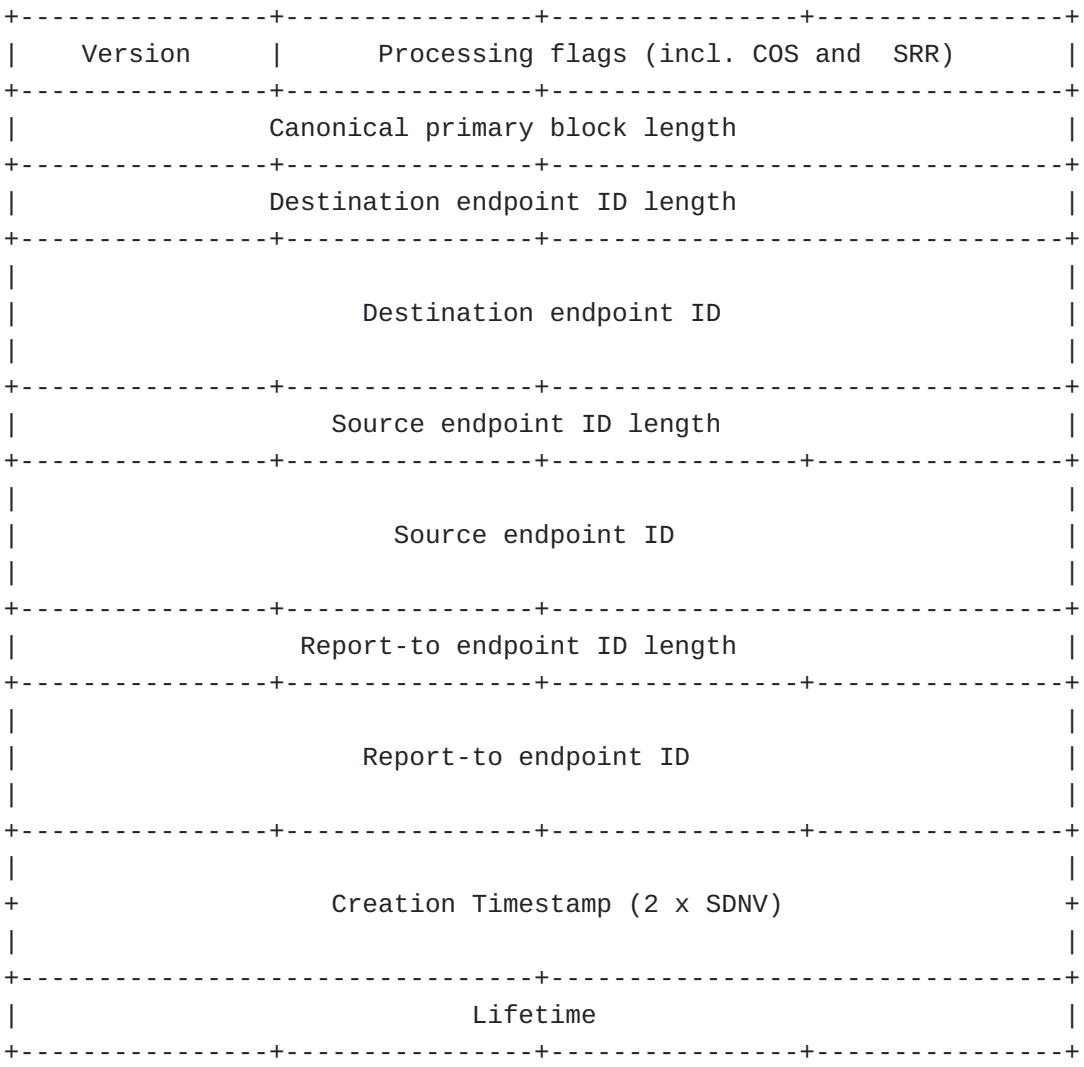
*- The value of the block data length field, which indicates the length of the block, is also part of the canonical form. Its value indicates the length of the entire bundle when the bundle includes the security result data field.

*- BABs are always added to bundles after PIBs, so when a PIB ciphersuite specifies this strict canonicalisation algorithm and the PIB is received with a bundle that also includes one or more BABs, application of strict canonicalisation as part of the PIB security result verification process requires that all BABs in the bundle be ignored entirely.

3.4.2. Mutable canonicalisation

This algorithm is intended to protect parts of the bundle which SHOULD NOT be changed in-transit. Hence it omits the mutable parts of the bundle.
The basic approach is to define a canonical form of the primary block and catenate it with the security (PIBs and PCBs only) and payload

blocks in the order that they will be transmitted. This algorithm ignores all other blocks, including ESBs, because it cannot be determined whether or not they will change as the bundle transits the network. In short, this canonicalization protects the payload, payload-related security blocks and parts of the primary block.
Many fields in various blocks are stored as variable-length SDNVs. These are canonicalized in unpacked form, as eight-byte fixed-width fields in network byte order. The size of eight bytes is chosen because implementations MAY handle larger values as invalid, as noted in [DTNBP].
The canonical form of the primary block is shown in Figure 6. Essentially, it de-references the dictionary block, adjusts lengths where necessary and ignores flags that MAY change in transit.



The fields shown in Figure 6 are:

*Version is the single-byte value in the primary block.

* Processing flags in the primary block is an SDNV, and includes the class-of-service (COS) and status report request (SRR) fields. For purposes of canonicalization, the SDNV is unpacked into a fixed-width field and some bits are masked out. The unpacked field is ANDed with mask 0x0000 0000 0007 C1BE to set to zero all reserved bits and the "bundle is a fragment" bit.

* Length - a four-byte value containing the length (in bytes) of this structure, in network byte order.

* Destination endpoint ID length and value - are the length (as a four byte value in network byte order) and value of the destination endpoint ID from the primary bundle block. The URI is simply copied from the relevant part(s) of the dictionary block and is not itself canonicalised. Although the dictionary entries contain null-terminators, the null-terminators are not included in the length or the canonicalization.

* Source endpoint ID length and value are handled similarly to the destination.

* Report-to endpoint ID length and value are handled similarly to the destination.

* Creation time (2 x SDNV) and Lifetime (SDNV) are simply copied from the primary block, with the SDNV values being represented as eight-byte unpacked values.

* Fragment offset and Total application data unit length are ignored, as is the case for the "bundle is a fragment" bit mentioned above. If the payload data to be canonicalized is less than the complete, original bundle payload, the offset and length are specified in the security-parameters.

For non-primary blocks being included in the canonicalization, the block processing flags value used for canonicalization is the unpacked SDNV value with reserved and mutable bits masked to zero. The unpacked value is ANDed with mask 0x0000 0000 0000 0077 to zero reserved bits and the "last block" flag. The "last block" flag is ignored because BABs and other security blocks MAY be added for some parts of the journey but not others so the setting of this bit might change from hop to hop.
Endpoint ID references in security blocks are canonicalized using the de-referenced text form in place of the reference pair. The reference count is not included, nor is the length of the endpoint ID text.
The block-length is canonicalized as an eight-byte unpacked value in network byte order. If the payload data to be canonicalized is less than the complete, original bundle payload, this field contain the size

of the data being canonicalized (the "effective block") rather that the actual size of the block.
Payload blocks are generally canonicalized as-is with the exception that in some instances only a portion of the payload data is to be protected. In such a case, only those bytes are included in the canonical form, and additional ciphersuite parameters are required to specify which part of the payload is protected, as discussed further below.
Security blocks are handled likewise, except that the ciphersuite will likely specify that the "current" security block security result field not be considered part of the canonical form. This differs from the strict canonicalisation case since we might use the mutable canonicalisation algorithm to handle sequential signatures such that signatures cover earlier ones.
ESBs MUST NOT be included in the canonicalization.
Notes:

*- The canonical form of the bundle is not transmitted. It is simply an artifact used as input to digesting.

*- We omit the reserved flags because we cannot determine if they will change in transit. The masks specified above will have to be revised if additional flags are defined and they need to be protected.

*- Our URI encoding does not preserve the "null-termination" convention from the dictionary field, nor do we separate the scheme and the scheme-specific part (SSP) as is done there.

*- The URI encoding will cause errors if any node rewrites the dictionary content (e.g. changing the DNS part of an HTTP URL from lower-case to upper case). This could happen transparently when a bundle is synched to disk using one set of software and then read from disk and forwarded by a second set of software. Because there are no general rules for canonicalising URIs (or IRIs), this problem may be an unavoidable source of integrity failures.

*- All SDNV fields here are canonicalized as eight-byte unpacked values in network byte order. Length fields are canonicalized as four-byte values in network byte order. Encoding does not need optimization since the values are never sent over the network.

*If a bundle is fragmented before the PIB is applied then the PIB applies to a fragment and not the entire bundle. However, the protected fragment could be subsequently further fragmented, which would leave the verifier unable to know which bytes were protected by the PIB. Even in the absence of fragmentation the

same situation applies if the ciphersuite is defined to allow protection of less than the entire, original bundle payload.

*For this reason, PIB ciphersuites which support applying a PIB to less than the complete, original bundle payload MUST specify, as part of the ciphersuite parameters, which bytes of the bundle payload are protected. When verification occurs, only the specified range of the payload bytes are input to PIB verification. It is valid for a ciphersuite to be specified so as to only apply to entire bundles and not to fragments. A ciphersuite MAY be specified to apply to only a portion of the payload, regardless of whether the payload is a fragment or the complete original bundle payload.

*The same fragmentation issue applies equally to PCB ciphersuites. Ciphersuites which support applying confidentiality to fragments MUST specify, as part of the ciphersuite parameters, which bytes of the bundle payload are protected. When decrypting a fragment, only the specified bytes are processed. It is also valid for a confidentiality ciphersuite to be specified so as to only apply to entire bundles and not to fragments.

This definition of mutable canonicalization assumes that endpoint IDs themselves are immutable and is unsuitable for use in environments where that assumption might be violated.
The canonicalization applies to a specific bundle and not a specific payload. If a bundle is forwarded in some way, the recipient is not able to verify the original integrity signature since the the source EID will be different, and possibly other fields.
The solution for either of these issues is to define and use a PIB ciphersuite having an alternate version of mutable canonicalization any fields from the primary block.

3.5. Endpoint ID confidentiality

Every bundle MUST contain a primary block that contains the source and destinations endpoint IDs, and others, and that cannot be encrypted. If endpoint ID confidentiality is required, then bundle-in-bundle encapsulation can solve this problem in some instances.
Similarly, confidentiality requirements MAY also apply to other parts of the primary block (e.g. the current-custodian) and that is supported in the same manner.

3.6. Bundles received from other nodes

Nodes implementing this specification SHALL consult their security policy to determine whether or not a received bundle is required by policy to include a BAB. If the bundle has no BAB and one is not required then BAB processing on the received bundle is complete and the

bundle is ready to be further processed for PIB/PCB/ESB handling or delivery or forwarding.

If the bundle is required to have a BAB but does not, then the bundle MUST be discarded and processed no further. If the bundle is required to have a BAB but all of its BABs identify a different node other than the receiving node as the BAB security destination, then the bundle MUST be discarded and processed no further.

If the bundle is required to have a BAB and has one or more BABs that identify the receiving node as the BAB security destination, or for which there is no security destination, then the value in the security result field(s) of the BAB(s) MUST be verified according to the ciphersuite specification. If for all such BABs in the bundle either the BAB security source cannot be determined or the security result value check fails, the bundle has failed to authenticate and the bundle MUST be discarded and processed no further. If any of the BABs present verify, or if a BAB is not required, the bundle is ready for further processing as determined by extension blocks and/or policy.

BABs received in a bundle MUST be stripped before the bundle is forwarded. New BABs MAY be added as required by policy. This MAY require correcting the "last block" field of the to-be-forwarded bundle.

Further processing of the bundle MUST take place in the order indicated by the various blocks from the primary block to the payload block, except as defined by an applicable specification.

If the bundle has a PCB and the receiving node is the PCB destination for the bundle (either because the node is listed as the bundle's PCB-dest or because the node is listed as the bundle's destination and there is no PCB-dest), the node MUST decrypt the relevant parts of the bundle in accordance with the ciphersuite specification. The PCB SHALL be deleted. If the relevant parts of the bundle cannot be decrypted (i.e. the decryption key cannot be deduced or decryption fails), then the bundle MUST be discarded and processed no further; in this case a bundle deletion status report (see the Bundle Protocol [DTNBP]) indicating the decryption failure MAY be generated. If the PCB security result included the ciphertext of a block other than the payload block, the recovered plaintext block MUST be placed in the bundle at the location from which the PCB was deleted.

If the bundle has one or more PIBs for which the receiving node is the bundle's PIB destination (either because the node is listed in the bundle's PIB-dest or because the node is listed as the bundle's destination and there is no PIB-dest), the node MUST verify the value in the PIB security result field(s) in accordance with the ciphersuite specification. If all the checks fail, the bundle has failed to authenticate and the bundle SHALL be processed according to the security policy. A bundle status report indicating the failure MAY be generated. Otherwise, if the PIB verifies, the bundle is ready to be processed for either delivery or forwarding. Before forwarding the bundle, the node SHOULD remove the PIB from the bundle, subject to the

requirements of Section 3.2, unless it is likely that some downstream node will also be able to verify the PIB.
If the bundle has a PIB and the receiving node is not the bundle's PIB-dest the receiving node MAY attempt to verify the value in the security result field. If it is able to check and the check fails, the node SHALL discard the bundle and it MAY send a bundle status report indicating the failure.
If the bundle has an ESB and the receiving node is the ESB destination for the bundle (either because the node is listed as the bundle's ESB-dest or because the node is listed as the bundle's destination and there is no ESB-dest), the node MUST decrypt and/or decapsulate the encapsulated block in accordance with the ciphersuite specification. The decapsulated block replaces the ESB in the bundle block sequence, and the ESB is thereby deleted. If the content cannot be decrypted (i.e., the decryption key cannot be deduced or decryption fails), then the bundle MAY be discarded and processed no further unless the security policy specifies otherwise. In this case a bundle deletion status report (see the Bundle Protocol [DTNBP]) indicating the decryption failure MAY be generated.

3.7. The At-Most-Once-Delivery Option

An application MAY request (in an implementation specific manner) that a node be registered as a member of an endpoint and that received bundles destined for that endpoint be delivered to that application. An option for use in such cases is known as "at-most-once-delivery". If this option is chosen, the application indicates that it wants the node to check for duplicate bundles, discard duplicates, and deliver at most one copy of each received bundle to the application. If this option is not chosen, the application indicates that it wants the node to deliver all received bundle copies to the application. If this option is chosen, the node SHALL deliver at most one copy of each received bundle to the application. If the option is not chosen, the node SHOULD, subject to policy, deliver all bundles.
To enforce this the node MUST look at the source/timestamp pair value of each complete (reassembled, if necessary) bundle received and determine if this pair, which uniquely identifies a bundle, has been previously received. If it has, then the bundle is a duplicate. If it has not, then the bundle is not a duplicate. The source/timestamp pair SHALL be added to the list of pair values already received by that node.
Each node implementation MAY decide how long to maintain a table of pair value state.

3.8. Bundle Fragmentation and Reassembly

If it is necessary for a node to fragment a bundle and security services have been applied to that bundle, the fragmentation rules described in [DTNBP] MUST be followed. As defined there and repeated

here for completeness, only the payload MAY be fragmented; security blocks, like all extension blocks, can never be fragmented. In addition, the following security-specific processing is REQUIRED:
The security policy requirements for a bundle MUST be applied individually to all the bundles resulting from a fragmentation event.
If the original bundle contained a PIB, then each of the PIB instances MUST be included in some fragment.
If the original bundle contained one or more PCBs, then any PCB instances containing a key information item MUST have the "replicate in every fragment" flag set, and thereby be replicated in every fragment. This is to ensure that the canonical block-sequence can be recovered during reassembly.
If the original bundle contained one or more correlated PCBs not containing a key information item, then each of these MUST be included in some fragment, but SHOULD NOT be sent more than once. They MUST be placed in a fragment in accordance with the fragmentation rules described in [DTNBP].
Note: various fragments MAY have additional security blocks added at this or later stages and it is possible that correlators will collide. In order to facilitate uniqueness, ciphersuites SHOULD include the fragment-offset of the fragment as a high-order component of the correlator.

3.9. Reactive fragmentation

When a partial bundle has been received, the receiving node SHALL consult its security policy to determine if it MAY fragment the bundle, converting the received portion into a bundle fragment for further forwarding. Whether or not reactive fragmentation is permitted SHALL depend on the security policy and the ciphersuite used to calculate the BAB authentication information, if required. (Some BAB ciphersuites, i.e., the mandatory BAB-HMAC ciphersuite defined in Section 4.1, do not accommodate reactive fragmentation because the security result in the BAB requires that the entire bundle be signed. It is conceivable, however, that a BAB ciphersuite could be defined such that multiple security results are calculated, each on a different segment of a bundle, and that these security results could be interspersed between bundle payload segments such that reactive fragmentation could be accommodated.)
If the bundle is reactively fragmented by the intermediate receiver and the BAB-ciphersuite is of an appropriate type (e.g. with multiple security results embedded in the payload), the bundle MUST be fragmented immediately after the last security result value in the partial payload that is received. Any data received after the last security result value MUST be dropped.
If a partial bundle is received at the intermediate receiver and is reactively fragmented and forwarded, only the part of the bundle that was not received MUST be retransmitted, though more of the bundle MAY be retransmitted. Before retransmitting a portion of the bundle, it

SHALL be changed into a fragment and, if the original bundle included a BAB, the fragmented bundle MUST also, and its BAB SHALL be recalculated.
This specification does not currently define any ciphersuite which can handle this reactive fragmentation case.
An interesting possibility is a ciphersuite definition such that the transmission of a follow-up fragment would be accompanied by the signature for the payload up to the restart point.

3.10. Attack Model

An evaluation of resilience to cryptographic attack necessarily depends upon the algorithms chosen for bulk data protection and for key transport. The mandatory ciphersuites described in the following section use AES, RSA and SHA algorithms in ways that are believed to be reasonably secure against ciphertext-only, chosen-ciphertext, known-plaintext and chosen-plaintext attacks.
The design has been careful to preserve the resilience of the algorithms against attack. For example, if a message is encrypted then any message integrity signature is also encrypted so that guesses cannot be confirmed.

4. Mandatory Ciphersuites

This section defines the mandatory ciphersuites for this specification. There is currently one mandatory ciphersuite for use with each of the security block types BAB, PIB, PCB and ESB. The BAB ciphersuite is based on shared secrets using HMAC. The PIB ciphersuite is based on digital signatures using RSA with SHA-256. The PCB and ESB ciphersuites are based on using RSA for key transport and AES for bulk encryption. The ciphersuites use the mechanisms defined in Cryptographic Message Syntax (CMS) [RFC5652] for packaging the keys, signatures, etc for transport in the appropriate security block. The data in the CMS object is not the bundle data, as would be the typical usage for CMS. Rather, the "message data" packaged by CMS is the ephemeral key, message digest, etc used in the core code of the ciphersuite.
In all cases where we use CMS, implementations SHOULD NOT include additional attributes whether signed or unsigned, authenticated or unauthenticated.

4.1. BAB-HMAC

The BAB-HMAC ciphersuite has ciphersuite ID value 0x001.
BAB-HMAC uses the strict canonicalisation algorithm in Section 3.4.1. Strict canonicalization supports digesting of a fragment-bundle. It does not permit the digesting of only a subset of the payload, but only the complete contents of the payload of the current bundle, which might be a fragment. The "fragment range" item for security-parameters is not

used to indicate a fragment, as this information is digested within the primary block.
The variant of HMAC to be used is HMAC-SHA1 as defined in [RFC2104].
This ciphersuite requires the use of two related instances of the BAB. It involves placing the first BAB instance (as defined in Section 2.2) just after the primary block. The second (correlated) instance of the BAB MUST be placed after all other blocks (except possibly other BAB blocks) in the bundle.
This means that normally, the BAB will be the second and last blocks of the bundle. If a forwarder wishes to apply more than one correlated BAB pair, then this can be done. There is no requirement that each application "wrap" the others, but the forwarder MUST insert all the "up front" BABs, and their "at back" "partners" (without any security result), before canonicalising.
Inserting more than one correlated BAB pair would be useful if the bundle could be routed to more than one potential "next-hop" or if both an old or a new key were valid at sending time, with no certainty about the situation that will obtain at reception time.
The security result is the output of the HMAC-SHA1 calculation with input being the result of running the entire bundle through the strict canonicalisation algorithm. Both required BAB instances MUST be included in the bundle before canonicalisation.
Security parameters are OPTIONAL with this scheme, but if used then the only field that can be present is key information (see Section 2.6).
Implementations MUST support use of "Enveloped-data" type as defined in [RFC5652] section 6, with RecipientInfo type KeyTransRecipientInfo containing the issuer and serial number of a suitable certificate. They MAY support additional RecipientInfo types. The "encryptedContent" field in EncryptedContentInfo contains the encrypted BEK that protects the payload and certain security blocks of the bundle.
In the absence of key information the receiver is expected to be able to find the correct key based on the sending identity. The sending identity MAY be known from the security-source field or the content of a previous-hop block in the bundle. It MAY also be determined using implementation-specific means such as the convergence layer.

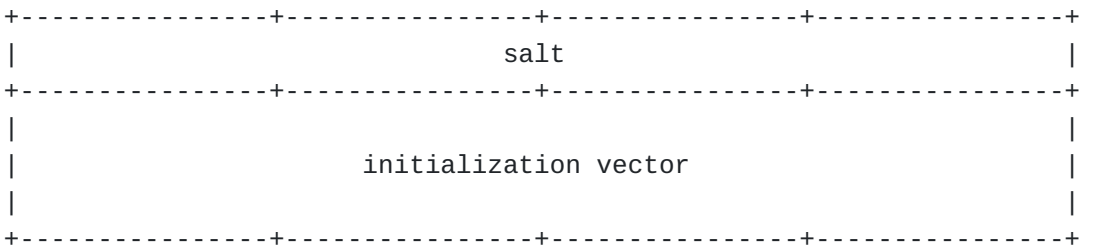
4.2. PIB-RSA-SHA256

The PIB-RSA-SHA256 ciphersuite has ciphersuite ID value 0x02.
PIB-RSA-SHA256 uses the mutable canonicalisation algorithm Section 3.4.2, with the security-result data field for only the "current" block being excluded from the canonical form. The resulting canonical form of the bundle is the input to the signing process. This ciphersuite requires the use of a single instance of the PIB.
Because the signature field in SignedData SignatureValue is a security-result field, the entire key information item MUST be placed in the block's security-result field, rather than security-parameters.
If the bundle being signed has been fragmented before signing, then we have to specify which bytes were signed in case the signed bundle is

subsequently fragmented for a second time. If the bundle is a fragment, then the ciphersuite parameters MUST include a fragment-range field, as described in Section 2.6, specifying the offset and length of the signed fragment. If the entire bundle is signed then these numbers MUST be omitted.
Implementations MUST support use of "SignedData" type as defined in [RFC5652] section 5.1, with SignerInfo type SignerIdentifier containing the issuer and serial number of a suitable certificate. The data to be signed is the output of the SHA256 mutable canonicalization process.

4.3. PCB-RSA-AES128-PAYLOAD-PIB-PCB

The PCB-RSA-AES128-PAYLOAD-PIB-PCB ciphersuite has ciphersuite ID value 0x003.
This scheme encrypts PIBs, PCBs and the payload. The key size for this ciphersuite is 128 bits.
Encryption is done using the AES algorithm in Galois/Counter Mode (GCM) as described in [RFC5084] Note: parts of the following description are borrowed from [RFC4106].
The choice of GCM avoids expansion of the payload, which causes problems with fragmentation/reassembly and custody transfer. GCM also includes authentication, essential in preventing attacks that can alter the decrypted plaintext or even recover the encryption key.
GCM is a block cipher mode of operation providing both confidentiality and data integrity. The GCM encryption operation has four inputs: a secret key, an initialization vector (IV), a plaintext, and an input for additional authenticated data (AAD) which is not used here. It has two outputs, a ciphertext whose length is identical to the plaintext, and an authentication tag, also known as the Integrity Check Value (ICV).
For consistency with the description in [RFC5084], we refer to the GCM IV as a nonce. The same key and nonce combination MUST NOT be used more than once. The nonce has the following layout



The salt field is a four-octet value, usually chosen at random. It MUST be the same for all PCBs which have the same correlator value. The salt need not be kept secret.
The initialization vector (IV) is an eight-octet value, usually chosen at random. It MUST be different for all PCBs which have the same correlator value. The value need not be kept secret.

The key (bundle encryption key, BEK) is a sixteen-octet (128 bits) value, usually chosen at random. The value MUST be kept secret, as described below.
The integrity check value is a sixteen-octet value used to verify that the protected data has not been altered. The value need not be kept secret.
This ciphersuite requires the use of a single PCB instance to deal with payload confidentiality. If the bundle already contains PIBs or PCBs then the ciphersuite will create additional correlated blocks to protect these PIBs and PCBs. These "additional" blocks replace the original blocks on a one-for-one basis, so the number of blocks remains unchanged. All these related blocks MUST have the same correlator value. The term "first PCB" in this section refers to the single PCB if there is only one or, if there are several, then to the one containing the key information. This MUST be the first of the set.
First PCB - the first PCB MAY contain a correlator value, and MAY specify security-source and/or security-destination in the EID-list. If not specified, the bundle-source and bundle-destination respectively are used for these values, as with other ciphersuites. The block MUST contain security-parameters and security-result fields. Each field MAY contain several items formatted as described in Section 2.6.
Security-parameters

- *key information
- *salt
- *IV (this instance applies only to payload)
- *fragment offset and length, if bundle is a fragment

Security-result

- *ICV

Subsequent PCBs MUST contain a correlator value to link them to the first PCB. Security-source and security-destination are implied from the first PCB, however see the discussion in Section 2.4 concerning EID-list entries. They MUST contain security-parameters and security-result fields as follows:
Security-parameters

- *IV for this specific block

Security-result

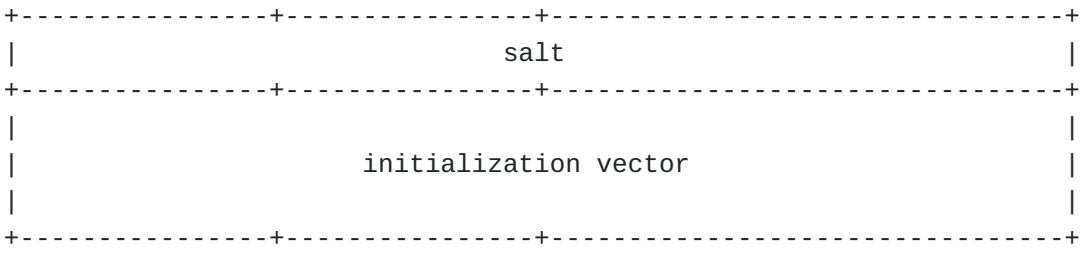
- *encapsulated block

The security-parameters and security-result fields in the subsequent PCBs MUST NOT contain any items other than these two. Items such as key and salt are supplied in the first PCB and MUST NOT be repeated.
Implementations MUST support use of "Enveloped-data" type as defined in [RFC5652] section 6, with RecipientInfo type KeyTransRecipientInfo containing the issuer and serial number of a suitable certificate. They MAY support additional RecipientInfo types. The "encryptedContent" field in EncryptedContentInfo contains the encrypted BEK that protects the payload and certain security blocks of the bundle.
The Integrity Check Value from the AES-GCM encryption of the payload is placed in the security-result field of the first PCB.
If the bundle being encrypted is a fragment-bundle we have to specify which bytes are encrypted in case the bundle is subsequently fragmented again. If the bundle is a fragment the ciphersuite parameters MUST include a fragment-range field, as described in Section 2.6, specifying the offset and length of the encrypted fragment. Note that this is not the same pair of fields which appear in the primary block as "offset and length". The "length" in this case is the length of the fragment, not the original length. If the bundle is not a fragment then this field MUST be omitted.
The confidentiality processing for payload and other blocks is different, mainly because the payload might be fragmented later at some other node.
For the payload, only the bytes of the bundle payload field are affected, being replaced by ciphertext. The salt, IV and key values specified in the first PCB are used to encrypt the payload, and the resultant authentication tag (ICV) is placed in an ICV item in the security-result field of that first PCB. The other bytes of the payload block, such as type, flags and length, are not modified.
For each PIB or PCB to be protected, the entire original block is encapsulated in a "replacing" PCB. This replacing PCB is placed in the outgoing bundle in the same position as the original block, PIB or PCB. As mentioned above, this is one-for-one replacement and there is no consolidation of blocks or mixing of data in any way.
The encryption process uses AES-GCM with the salt and key values from the first PCB, and an IV unique to this PCB. The process creates ciphertext for the entire original block, and an authentication tag for validation at the security destination. For this encapsulation process, unlike the processing of the bundle payload, the authentication tag is appended to the ciphertext for the block and the combination is stored into the "encapsulated block" item in security-result.
The replacing block, of course, also has the same correlator value as the first PCB with which it is associated. It also contains the block-specific IV in security-parameters, and the combination of original-block-ciphertext and authentication tag, stored as an "encapsulated block" item in security-result.
If the payload was fragmented after encryption then all those fragments MUST be present and reassembled before decryption. This process might

be repeated several times at different destinations if multiple fragmentation actions have occurred.
The size of the GCM counter field limits the payload size to 2^39 - 256 bytes, about half a terabyte. A future revision of this specification will address the issue of handling payloads in excess of this size.

4.4. ESB-RSA-AES128-EXT

The ESB-RSA-AES128-EXT ciphersuite has ciphersuite ID value 0x004.
This scheme encrypts non-payload-related blocks. It MUST NOT be used to encrypt PIBs, PCBs or primary or payload blocks. The key size for this ciphersuite is 128 bits.
Encryption is done using the AES algorithm in Galois/Counter Mode (GCM) as described in [RFC5084] Note: parts of the following description are borrowed from [RFC4106].
GCM is a block cipher mode of operation providing both confidentiality and data origin authentication. The GCM authenticated encryption operation has four inputs: a secret key, an initialization vector (IV), a plaintext, and an input for additional authenticated data (AAD) which is not used here. It has two outputs, a ciphertext whose length is identical to the plaintext, and an authentication tag, also known as the Integrity Check Value (ICV).
For consistency with the description in [RFC5084], we refer to the GCM IV as a nonce. The same key and nonce combination MUST NOT be used more than once. The nonce has the following layout



The salt field is a four-octet value, usually chosen at random. It MUST be the same for all ESBs which have the same correlator value. The salt need not be kept secret.
The initialization vector (IV) is an eight-octet value, usually chosen at random. It MUST be different for all ESBs which have the same correlator value. The value need not be kept secret.
The data encryption key is a sixteen-octet (128 bits) value, usually chosen at random. The value MUST be kept secret, as described below.
The integrity check value is a sixteen-octet value used to verify that the protected data has not been altered. The value need not be kept secret.
This ciphersuite replaces each BP extension block to be protected with a "replacing" ESB, and each can be individually specified.

If a number of related BP extension blocks are to be protected they can be grouped as a correlated set and protected using a single key. These blocks replace the original blocks on a one-for-one basis, so the number of blocks remains unchanged. All these related blocks MUST have the same correlator value. The term "first ESB" in this section refers to the single ESB if there is only one or, if there are several, then to the one containing the key or key-identifier. This MUST be the first of the set. If the blocks are individually specified then there is no correlated set and each block is its own "first ESB".
First ESB - the first ESB MAY contain a correlator value, and MAY specify security-source and/or security-destination in the EID-list. If not specified, the bundle-source and bundle-destination respectively are used for these values, as with other ciphersuites. The block MUST contain security-parameters and security-result fields. Each field MAY contain several items formatted as described in Section 2.6.
Security-parameters

*key information

*salt

*IV for this specific block

*block type of encapsulated block (OPTIONAL)

Security-result

*encapsulated block

Subsequent ESBs MUST contain a correlator value to link them to the first ESB. Security-source and security-destination are implied from the first ESB, however see the discussion in Section 2.4 concerning EID-list entries. Subsequent ESBs MUST contain security-parameters and security-result fields as follows:
Security-parameters

*IV for this specific block

*block type of encapsulated block (OPTIONAL)

Security-result

*encapsulated block

The security-parameters and security-result fields in the subsequent ESBs MUST NOT contain any items other than those listed. Items such as key and salt are supplied in the first ESB and MUST NOT be repeated. Implementations MUST support use of "Enveloped-data" type as defined in [RFC5652] section 6, with RecipientInfo type KeyTransRecipientInfo containing the issuer and serial number of a suitable certificate. They

MAY support additional RecipientInfo types. The "encryptedContent" field in EncryptedContentInfo contains the encrypted BEK used to encrypt the content of the block being protected.
For each block to be protected, the entire original block is encapsulated in a "replacing" ESB. This replacing ESB is placed in the outgoing bundle in the same position as the original block. As mentioned above, this is one-for-one replacement and there is no consolidation of blocks or mixing of data in any way.
The encryption process uses AES-GCM with the salt and key values from the first ESB, and an IV unique to this ESB. The process creates ciphertext for the entire original block, and an authentication tag for validation at the security destination. The authentication tag is appended to the ciphertext for the block and the combination is stored into the "encapsulated block" item in security-result.
The replacing block, of course, also has the same correlator value as the first ESB with which it is associated. It also contains the block-specific IV in security-parameters, and the combination of original-block-ciphertext and authentication tag, stored as an "encapsulated block" item in security-result.

5. Key Management

Key management in delay tolerant networks is recognized as a difficult topic and is one that this specification does not attempt to solve. However, solely in order to support implementation and testing, implementations SHOULD support:

*- The use of well-known RSA public keys for all ciphersuites.

*- Long-term pre-shared-symmetric keys for the BAB-HMAC ciphersuite.

Since endpoint IDs are URIs and URIs can be placed in X.509 [RFC5280] public key certificates (in the subjectAltName extension) implementations SHOULD support this way of distributing public keys. RFC 5280 does not insist that implementations include revocation checking. In the context of a DTN, it is reasonably likely that some nodes would not be able to use revocation checking services (either CRLs or OCSP) and deployments SHOULD take this into account when planning any public key infrastructure to support this specification.

6. Default Security Policy

Every node serves as a Policy Enforcement Point insofar as it enforces some policy that controls the forwarding and delivery of bundles via one or more convergence layer protocol implementation. Consequently, every node SHALL have and operate according to its own configurable

security policy, whether the policy be explicit or default. The policy SHALL specify:

- Under what conditions received bundles SHALL be forwarded.
- Under what conditions received bundles SHALL be required to include valid BABs.
- Under what conditions the authentication information provided in a bundle's BAB SHALL be deemed adequate to authenticate the bundle.
- Under what conditions received bundles SHALL be required to have valid PIBs and/or PCBs.
- Under what conditions the authentication information provided in a bundle's PIB SHALL be deemed adequate to authenticate the bundle.
- Under what conditions a BAB SHALL be added to a received bundle before that bundle is forwarded.
- Under what conditions a PIB SHALL be added to a received bundle before that bundle is forwarded.
- Under what conditions a PCB SHALL be added to a received bundle before that bundle is forwarded.
- Under what conditions an ESB SHALL be applied to one or more blocks in a received bundle before that bundle is forwarded.
- The actions that SHALL be taken in the event that a received bundle does not meet the receiving node's security policy criteria.

This specification does not address how security policies get distributed to nodes. It only REQUIRES that nodes have and enforce security policies.
If no security policy is specified at a given node, or if a security policy is only partially specified, that node's default policy regarding unspecified criteria SHALL consist of the following:

- Bundles that are not well-formed do not meet the security policy criteria.
- The mandatory ciphersuites MUST be used.
- All bundles received MUST have a BAB which MUST be verified to contain a valid security result. If the bundle does not have a BAB, then the bundle MUST be discarded and processed no further;

a bundle status report indicating the authentication failure MAY be generated.

*No received bundles SHALL be required to have a PIB; if a received bundle does have a PIB, however, the PIB can be ignored unless the receiving node is the PIB-dest, in which case the PIB MUST be verified.

*No received bundles SHALL be required to have a PCB; if a received bundle does have a PCB, however, the PCB can be ignored unless the receiving node is the PCB-dest, in which case the PCB MUST be processed. If processing of a PCB yields a PIB, that PIB SHALL be processed by the node according to the node's security policy.

*A PIB SHALL NOT be added to a bundle before sourcing or forwarding it.

*A PCB SHALL NOT be added to a bundle before sourcing or forwarding it.

*A BAB MUST always be added to a bundle before that bundle is forwarded.

*If a destination node receives a bundle that has a PIB-dest but the value in that PIB-dest is not the EID of the destination node, the bundle SHALL be delivered at that destination node.

*If a destination node receives a bundle that has an ESB-dest but the value in that ESB-dest is not the EID of the destination node, the bundle SHALL be delivered at that destination node.

*If a received bundle does not satisfy the node's security policy for any reason, then the bundle MUST be discarded and processed no further; in this case, a bundle deletion status report (see the Bundle Protocol [DTNBP]) indicating the failure MAY be generated.

7. Security Considerations

The Bundle Security Protocol builds upon much work of others, in particular the Cryptographic Message Syntax (CMS) [RFC5652] and Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [RFC5280]. The security considerations in these two documents apply here as well.
Several documents specifically consider the use of Galois/Counter Mode(GCM) and of AES and are important to consider when building ciphersuites. These are The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP) [RFC4106] and Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax

(CMS) [RFC5084]. Although the BSP is not identical, many of the security issues considered in these documents also apply here.
Certain applications of DTN need to both sign and encrypt a message and there are security issues to consider with this.
If the intent is to provide an assurance that a message did in fact come from a specific source and has not been changed then it should be signed first and then encrypted. A signature on an encrypted message does not establish any relationship between the signer and the original plaintext message.
On the other hand, if the intent is reduce the threat of denial-of-service attacks then signing the encrypted message is appropriate. A message that fails the signature check will not be processed through the computationally-intensive decryption pass. A more extensive discussion of these points is in S/MIME 3.2 Message Specification [RFC5751], especially in section 3.6.
Additional details relating to these combinations can be found at Section 2.8 where it is RECOMMENDED that the encrypt-then-sign combination is usually appropriate for usage in a DTN.
In a DTN encrypt-then-sign potentially allows intermediate nodes to verify a signature (over the ciphertext) and thereby apply policy to manage possibly scarce storage or other resources at intermediate nodes in the path the bundle takes from source to destination EID.
An encrypt-then-sign scheme doesn't further expose identity in most cases since the BP mandates that the source EID (which is commonly expected to be the security-source) is already exposed in the primary block of the bundle. Should either exposure of the source EID or signerInfo be considered an interesting vulnerability, then some form of bundle-in-bundle encapsulation would be required as a mitigation.
If a BAB ciphersuite uses digital signatures but doesn't include the security destination (which for a BAB is the next host), then this allows the bundle to be sent to some node other than the intended adjacent node. Because the BAB will still authenticate, the receiving node might erroneously accept and forward the bundle. When asymmetric BAB ciphersuites are used, the security destination field SHOULD therefore be included in the BAB.
If a bundle's PIB-dest is not the same as its destination, then some node other than the destination (the node identified as the PIB-dest) is expected to validate the PIB security result while the bundle is en route. However, if for some reason the PIB is not validated, there is no way for the destination to become aware of this. Typically, a PIB-dest will remove the PIB from the bundle after verifying the PIB and before forwarding it. However, if there is a possibility that the PIB will also be verified at a downstream node, the PIB-dest will leave the PIB in the bundle. Therefore, if a destination receives a bundle with a PIB that has a PIB-dest (which isn't the destination), this might, but does not necessarily, indicate a possible problem.
If a bundle is fragmented after being forwarded by its PIB-source but before being received by its PIB-dest, the payload in the bundle MUST be reassembled before validating the PIB security result in order for

the security result to validate correctly. Therefore, if the PIB-dest is not capable of performing payload reassembly, its utility as a PIB-dest will be limited to validating only those bundles that have not been fragmented since being forwarded from the PIB-source. Similarly, if a bundle is fragmented after being forwarded by its PIB-source but before being received by its PIB-dest, all fragments MUST be received at that PIB-dest in order for the bundle payload to be able to be reassembled. If not all fragments are received at the PIB-dest node, the bundle will not be able to be authenticated, and will therefore never be forwarded by this PIB-dest node.

Specification of a security-destination other than the bundle destination creates a routing requirement that the bundle somehow be directed to the security-destination node on its way to the final destination. This requirement is presently private to the ciphersuite, since routing nodes are not required to implement security processing. If a security target were to generate reports in the event that some security validation step fails, then that might leak information about the internal structure or policies of the DTN containing the security target. This is sometimes considered bad security practice so SHOULD only be done with care.

8. Conformance

As indicated above, this document describes both BSP and ciphersuites. A conformant implementation MUST implement both BSP support and the four ciphersuites described in Section 4. It MAY also support other ciphersuites.

Implementations that support BSP but not all four mandatory ciphersuites MUST claim only "restricted compliance" with this specification, even if they provide other ciphersuites.

All implementations are strongly RECOMMENDED to provide at least a BAB ciphersuite. A relay node, for example, might not deal with end-to-end confidentiality and data integrity but it SHOULD exclude unauthorized traffic and perform hop-by-hop bundle verification.

9. IANA Considerations

This protocol has fields requiring registries managed by IANA.

9.1. Bundle Block Types

This specification allocates four codepoints from the existing Bundle Block Type Codes registry defined in [I-D.irtf-dtnrg-iana-bp-registries].

Additional Entries for the Bundle Block Type Codes Registry:

Value	Description	Reference
2	Bundle Authentication Block	This document
3	Payload Integrity Block	This document
4	Payload Confidentiality Block	This document
9	Extension Security Block	This document

9.2. Ciphersuite Numbers

This Protocol has a ciphersuite number field and certain ciphersuites are defined. An IANA registry shall be set up as follows.
The registration policy for this registry is: Specification Required
The Value range is: Variable Length

Ciphersuite Numbers Registry:

Value	Description	Reference
0	unassigned	This document
1	BAB-HMAC	This document
2	PIB-RSA-SHA256	This document
3	PCB-RSA-AES128-PAYLOAD-PIB-PCB	This document
4	ESB-RSA-AES128-EXT	This document
>4	Reserved	This document

9.3. Ciphersuite Flags

This Protocol has a ciphersuite flags field and certain flags are defined. An IANA registry shall be set up as follows.
The registration policy for this registry is: Specification Required
The Value range is: Variable Length

Ciphersuite Flags Registry:

Bit Position (right to left)	Description	Reference
0	Block contains result	This document
1	Block contains correlator	This document
2	Block contains parameters	This document
3	Destination EIDref present	This document
4	Source EIDref present	This document
all others	Reserved	This document

9.4. Parameters and Results

This Protocol has fields for ciphersuite parameters and results. The field is a type-length-value triple and a registry is required for the "type" sub-field. The values for "type" apply to both the ciphersuite parameters and the ciphersuite results fields. Certain values are defined. An IANA registry shall be set up as follows.
The registration policy for this registry is: Specification Required
The Value range is: 8-bit unsigned integer

Ciphersuite Parameters and Results Type Registry:

Value	Description	Reference
0	reserved	This document
1	initialization vector (IV)	This document
2	reserved	This document
3	key-information	This document
4	fragment range (pair of SDNVs)	This document
5	integrity signature	This document
6	unassigned	This document
7	salt	This document
8	PCB integrity check value (ICV)	This document
9	reserved	This document
10	encapsulated block	This document
11	block type of encapsulated block	This document
12-191	reserved	This document
192-250	private use	This document
251-255	reserved	This document

10. References

10.1. Normative References

[RFC2119]	Bradner, S. and J. Reynolds, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, October 1997.
[DTNBP]	Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.
[DTNMD]	Symington, S.F., "Delay-Tolerant Networking Metadata Extension Block", draft-irtf-dtnrg-bundle-metadata-block-00.txt , June 2007.
[RFC2104]	Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
[RFC5280]	Cooper, D., Santesson, S., Farrell, S., Polk, W. and W. Ford, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
[RFC5652]	Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, July 2004.
[RFC4106]	Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, June 2005.
[I-D.irtf-dtnrg-iana-bp-registries]	Blanchet, M., "Delay-Tolerant Networks (DTN) Bundle Protocol IANA Registries", draft-irtf-dtnrg-iana-bp-registries-00.txt, work-in-progress, April 2010.

10.2. Informative References

[DTNarch]	Cerf, V., Burleigh, S., Durst, R., Fall, K., Hooke, A., Scott, K., Torgerson, L. and H. Weiss, "Delay-Tolerant Network Architecture", RFC 4838, April 2007.
[PHIB]	Symington, S., "Delay-Tolerant Networking Previous Hop Insertion Block", draft-irtf-dtnrg-bundle-previous-hop-block-11.txt, work-in-progress, February 2010.
[RFC5084]	Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007.
[RFC5751]	Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.
[RFC3986]	Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

Authors' Addresses

Susan Flynn Symington Symington The MITRE Corporation 7515 Colshire Drive McLean, VA 22102 US Phone: +1 (703) 983-7209 EMail: susan@mitre.org URI: http://mitre.org/

Stephen Farrell Farrell Trinity College Dublin Distributed Systems Group Department of Computer Science Trinity College Dublin, 2 Ireland Phone: +353-1-608-1539 EMail: stephen.farrell@cs.tcd.ie

Howard Weiss Weiss SPARTA, Inc. 7110 Samuel Morse Drive Columbia, MD 21046 US Phone: +1-443-430-8089 EMail: howard.weiss@sparta.com

Peter Lovell Lovell SPARTA, Inc. 7110 Samuel Morse Drive Columbia, MD 21046 US Phone: +1-443-430-8052 EMail: dtnbsp@gmail.com