

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: November 12, 2011

W. Eddy
MTI Systems
L. Wood
University of Surrey
W. Ivancic
NASA
May 11, 2011

Reliability-only Ciphersuites for the Bundle Protocol
draft-irtf-dtnrg-bundle-checksum-09

Abstract

The Delay-Tolerant Networking Bundle Protocol includes a custody transfer mechanism to provide acknowledgements of receipt for particular bundles. No checksum is included in the basic DTN Bundle Protocol, however, so at intermediate hops, it is not possible to verify that bundles have been either forwarded or passed through convergence layers without error. Without assurance that a bundle has been received without errors, the custody transfer receipt cannot guarantee that a correct copy of the bundle has been transferred, and errored bundles are forwarded when the destination cannot use the errored content, and discarding the errored bundle early would have been better for performance and throughput reasons. This document addresses that situation by defining new ciphersuites for use within the existing Bundle Security Protocol's Payload Integrity Block (formerly called the Payload Security Block [ED: remove old name before RFC]) to provide error-detection functions that do not require support for other, more complex, security-providing ciphersuites that protect integrity against deliberate modifications. This creates the checksum service needed for error-free reliability, and does so by separating security concerns from the few new reliability-only ciphersuite definitions that are introduced here. The reliability-only ciphersuites given here are intended to protect only against errors and accidental modification; not against deliberate integrity violations. This document discusses the advantages and disadvantages of this approach and the existing constraints that combined to drive this design.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-

Internet-Draft Reliability-only Checksum Ciphersuites

May 2011

Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 12, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | | |
|-----------------------------|--|--------------------|
| 1. | Motivations | 4 |
| 1.1. | Overview of Header and Payload Integrity | 5 |
| 2. | Use of the Payload Integrity Block | 8 |
| 2.1. | Differences from Intended Use of the Payload Integrity Block | 9 |
| 3. | INSECURE Ciphersuites | 12 |
| 3.1. | Generation and Processing Rules | 14 |
| 4. | Performance Considerations | 15 |
| 5. | Security Considerations | 16 |
| 6. | IANA Considerations | 18 |
| 7. | Acknowledgements | 18 |
| 8. | References | 18 |
| 8.1. | Normative References | 18 |
| 8.2. | Informative References | 19 |
| Appendix A. | Mandatory BSP Elements Needed to Implement Error-Detection | 20 |
| A.1. | Discussion of Mutable Canonicalization | 20 |
| A.2. | Discussion of the PIB Format | 22 |
| | Authors' Addresses | 23 |

1. Motivations

Reliable transmission of information is a well-known problem for all protocol layers. Error-detection and correction capabilities are frequently found in lower layers, but are also present in many higher-layer protocols in order to detect residual bit errors and bugs that introduce errors. For example, IPv4 verifies a simple header checksum when processing packets, even when running over a data link, such as Ethernet, that already performs a stronger CRC. The TCP and UDP transport protocols further include a checksum covering their payloads as well as some IP header fields. This checksum is verified before the data is passed to the application. What may seem like paranoia is actually not unfounded, as errors in received data or packet corruption are known to creep into networking systems from many causes other than channel noise [[SP00](#)]. Although coding of data on the channel can reduce the impact of channel noise, and Cyclic Redundancy Codes (CRCs) across each link in a network path can catch many channel-induced errors, end-to-end checksums across the entire network path are understood to be necessary for applications requiring certainty that the data received is error-free [[SGHP98](#)].

The Delay/Disruption-Tolerant Networking (DTN) architecture [[RFC4838](#)] is founded on an overlay of Bundle Agents (BAs). These Bundle Agents forward data units called bundles via the Bundle Protocol [[RFC5050](#)]. Bundles may be lost or errored both during transmission between BAs, or within a BA itself. Bundles belonging to applications that are not tolerant of lost data have a "custody transfer" flag that requests reliable transmission between bundle agents. The notion of

reliability used in the basic custody transfer mechanism means that the Bundle Protocol itself not take the integrity of bundles into account, but acknowledges a bundle's receipt and transfers its custody without verifying its internal data integrity. In this way, the bundle protocol provides what has been called a "better than best-effort" service, which attempts through persistence to provide delivery of content without making claims about the end-to-end integrity of that content. Although [[RFC4838](#)] discusses 'reliable delivery', this is expected to be provided to the bundle layer, and is not a property of the bundle layer. The "convergence layer adapters" that connect BAs to each other may or may not detect and correct errors before presenting bundle data to the BAs themselves. Convergence layer error detection and correction may be adequate in many cases, but is not always adequate, and the lack of adequacy is recognised in the well-known end-to-end principle [[SRC84](#)]. It is possible (and even statistically likely) that either transmission errors will go unnoticed, or unchecked errors will be introduced within a BA's memory, storage, or forwarding systems. Here, each convergence-layer check is analogous to a data-link check, covering a

single hop between Bundle Agents, but not the entire network path between source and destination for the bundle.

Within the context of DTN, even stronger convergence-layer adapter error detection is not sufficient. Errors within a BA's device drivers, errors due to memory issues within the BA's host, e.g. radiation-induced soft errors, and errors introduced from file-system corruption cannot be detected by convergence layer adapters, as these errors occur in gaps between successive phases of forwarding and convergence-layer processing. In order to ensure integrity of DTN bundles forwarded across a system composed of BAs and convergence layer adapters, end-to-end computation and verification of checksums is required [[SRC84](#)]. To detect errors introduced in storage, a checksum across the bundle could be verified before the bundle is sent to the next hop. Detecting errors as early as possible leads to performance increases and better network utilization due to the nature of control loop in DTNs, as described later.

Within this document, we describe a use of the Bundle Security Protocol (BSP) [[I-D.irtf-dtnrg-bundle-security](#)] in order to provide the desired error-detection service by defining suitable BSP ciphersuites. The design decisions for doing this are explained in

[Section 2](#). It should be clearly understood by readers, implementers, and users that we are not using the BSP in a way that provides any level of security, which we explain fully in [Section 2.1](#). The guarantee that we attempt to provide is that specific blocks within a received bundle are highly likely to have been propagated across the overlay without errors, under the assumptions of no malicious activity within or between Bundle Agents and no capability to inject forged bundles. The actual format and use of this error-detection mechanism based on the BSP and requirements for support are described in [Section 3](#).

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 [RFC 2119](#). [[RFC2119](#)]

[1.1](#). Overview of Header and Payload Integrity

It is worthwhile to review checksum use in the Internet that ensures error-free delivery, as this should provide a familiar starting point for the reader. This can then be used as a basis for thinking about error-free delivery of bundles. We must distinguish between the need to verify the integrity and reliability of ("protect") carried payloads, and the need to verify the reliability of the protocol carrying the payload - i.e. the need to also protect the header metadata surrounding the payload that the protocol tells itself.

Within the topic of payload reliability, applications have differing requirements. Some applications are capable of dealing with errored content, and desire the delivery of any sent content blocks, even if they are not entirely the same content blocks that were originally transmitted. For instance, some voice and video codecs degrade perceptibly under loss of content, but cope well with low levels of error within delivered content. To date, however, most Internet applications have not been tolerant of errored content - and carrying errored content from a noisy physical channel to the application requires that all underlying layers in use at the time pass through and accept errors in their payload data, while simultaneously protecting against and rejecting errors in their own header data.

In IPv4, header and payload reliability are implemented separately, with the IPv4 header checksum covering the header information, while

the TCP or UDP checksum covers the payload. The TCP/UDP checksum also covers certain important IP header fields - this is the 'pseudo-header' check. In IPv6, the header checksum was removed, to speed up router processing as header fields were changed. The 'pseudo-header' checksum of important IPv6 header fields is still there as part of the overall payload checksum implemented by TCP, UDP, and other protocols. This upper-layer pseudo-header checksum combines the header and payload checks efficiently at the receiving IPv6 endhost to prevent misdelivery of packets and protect against corrupted payloads. This checksum is a final end-to-end check across the entire IP delivery path - for both header and payload. It ensures that the payload has been delivered to the right place, without corruption being introduced.

The TCP and UDP checksum mechanism is frequently criticised for being weak in that there are classes of errors it does not detect. This checksum is computed by summing 16-bit values within a packet. If two strings are swapped in position within the packet, the checksum can remain unchanged even though the datagram is now different from the original, and clearly corrupted. The UDP convergence-layer adapter that has been popularly implemented in DTN stacks relies on UDP's usual 16-bit one's-complement checksum (the same algorithm used for IPv4 headers and TCP segments) to validate incoming datagrams. The proposed TCP-based convergence layer [[I-D.irtf-dtnrg-tcp-clayer](#)] relies on the same checksum algorithm. This checksum algorithm is remarkably useful in its position covering an entire network path and ability to detect all errors introduced, even though its overall strength against errors is considered weak, and in more recent transport protocols has been reconsidered; for instance, SCTP uses the CRC-32c algorithm instead [[RFC3309](#)]. It is the ability to provide a check that the packet at the destination is the same at the source that is most useful for Internet traffic; the overall check mechanism strength is secondary to this. Any check is better than

none. The UDP- and TCP-convergence layers and their checksums are only useful for one 'bundle hop' between peer agents across the Internet, and will not cover multiple bundle hops as an end-to-end checksum.

In absence of IP packet fragmentation, it would be possible for all intermediate routers in the terrestrial Internet to verify the upper-layer pseudo-header checksum at each hop rather than just checking

data-link CRCs, but this is not normally done, as it would be burdensome to busy routers as well as a philosophical "layer violation". (Network Address Translation, where the change in pseudo-header addresses affects the checksum value, is an exception to this.) Not checking the payload at every hop takes advantage of the closed-loop nature of fast Internet communication; the cost of packet delivery is cheap, so necessary protection against corrupted packets or headers need only be done at the last possible moment with a check at the endhost, as the costs incurred in delivering the packet to that node, and any resent packets to that node, are minuscule. It is the cheap cost of resending and the tight end-to-end control loop which can permit reliability checks to be pushed up to or only done at the application layer, in accordance with the end-to-end principle, without adversely affecting overall communication efficiency.

In challenged delay-tolerant networks, the network capacity, forwarding and storage costs for bundles are far higher, as are the costs of getting a resent bundle. This produces a longer, slower, control loop and alters the cost/benefit tradeoff. It becomes worthwhile for intermediate bundle nodes to check non-error-tolerant content, which includes important header fields and metadata, for error-free correctness before forwarding it, to prevent incurring unnecessary transmission costs and to also increase the possibility of getting the bundle from the source or from over the previous hop again if it needs to be resent, shortening the overall resend time. Due to the more open-loop nature of communication across delay-tolerant and disrupted networks, applications do not communicate as well or as easily as they do in the traditional Internet, meaning that the efficiency tradeoff for application-only checks is such that non-error-tolerant DTN application performance can be expected to increase with earlier detection of errors and earlier resends within the network. Network costs are decreased and utilization is increased by early discard of known 'bad' traffic. This is a concern that has previously led to e.g. ATM Early Packet Discard (EPD).

This shift in costs caused by the difficulty of resending information, when compared to the traditional Internet, makes verifying the correctness of the header and non-error-tolerant payload content at each bundle node considerably more important, to

The Bundle Protocol, as specified in [[RFC5050](#)], does neither of these things. Nor does the Bundle Protocol protect itself end-to-end. There is no Bundle Protocol check on header or payload at destination bundle nodes; there is no end-to-end equivalent to the two checks that the TCP/UDP checksum so neatly combines and provides for Internet traffic. There is a need for a check at intermediate nodes in challenged networks for suitable traffic, given the cost tradeoffs outlined above, yet those nodes cannot take advantage of a precomputed end-to-end checksum unless one is first provided for them.

This document lays out a way to redress this lack of robustness in the existing Bundle Protocol in [Section 3](#), and carefully describes the drawbacks to this method in [Section 4](#) and [Section 5](#).

[2](#). Use of the Payload Integrity Block

The BSP defines three types of blocks:

- o the Bundle Authentication Block (BAB),
- o the Payload Integrity Block (PIB),
- o the Payload Confidentiality Block (PCB).

Algorithms applied within each of these blocks could be reused to detect errors introduced in bundle contents. However, based on the different design goals motivating these three block types, the PIB is the only candidate that is truly suitable for supporting the type of checksum fields needed to yield end-to-end reliability of received bundles.

The BAB is intended to operate along a single hop within a DTN overlay network, and due to the issues discussed in the previous section, even an end-to-end chain of hops using the BAB over each hop is not sufficient for ensuring reliability.

The PCB is primarily concerned with the operation of privacy transforms over a bundle's contents, which are intended to significantly alter and disguise the protected data while in transit, rather than simply performing consistency checks over untransformed data.

The PIB is intended to be used end-to-end; that is, by a set of endpoints, rather than hop-by-hop at each intermediate point. The PIB is intended to be used with sets of cryptographic algorithms

(ciphersuites) that provide Message Authentication Codes (MACs) or signatures over bundle or block contents. MAC and signature algorithms are security constructions that may allow verification of a legitimate sender (authentication), detection of in-transit tampering (integrity), and proof of a particular sender (non-repudiation with proof of origin). As a consequence of the integrity goal, which is based on the assumption of an adversary that can alter messages in-flight, MACs and signatures can also be effective at detecting errors that occur without the presence of an attacker and in the absence of any malicious intent, e.g. due to bit errors within transmission media, file system corruption, etc. Since the PIB uses the BSP's mutable canonicalization and covers the Primary Bundle Block, the EIDs and other data influencing forwarding and delivery of payloads are also protected by the MACs or signatures in addition to the payload data.

The error-detecting and rejecting capabilities of a MAC or signature are similar to those of more-simple checksum algorithms that are intended only for error-detection. In fact, several popular MAC and signature constructions use checksums as primary components. For instance, the MD5 digest (hash) algorithm [[RFC1321](#)] is used within the HMAC-MD5 keyed-hash MAC construction [[RFC2104](#)]. Computationally, for large messages, the efficiency of a security construction providing integrity is similar to that of a simple checksum, although for short messages, it may be much worse. For instance, HMAC requires multiple applications of the underlying hash function, with the final one being over a very short input, but if the message itself fits within a single block, this results in twice the overhead compared to a simple checksum. Thus, assuming large bundles in relation to the block size of typical hash functions, the PIB can provide end-to-end error-detection capability for bundles from the standpoints of both reasonable effectiveness and reasonable computational cost.

[2.1.](#) Differences from Intended Use of the Payload Integrity Block

The main difference between any simple error-detecting checksum and a security construction designed for integrity is that the security construction requires keying material. Key management is recognized as an outstanding unsolved problem within the DTNRG [[I-D.irtf-dtnrg-sec-overview](#)] [[WEH09](#)] and is thought to be quite difficult. Key management in well-connected systems, such as the Internet, is difficult itself, without the additional complications of a DTN networking environment. However, if using a keyed security construction for simple error-detection, the secrecy of the key is unimportant, and a feasible approach is to specify a hard-coded key

that all nodes use in the error-detection mechanism. The NULL ciphersuite used in the Licklider Transmission Protocol (LTP) for its

authentication extension is one example of this [[RFC5327](#)]) that led directly to this approach. Using this approach, existing keyed ciphersuites defined for the PIB could be used with well-known keys to provide an error-detection mechanism, without requiring a key management mechanism. However, this key-based method reuses a security mechanism for error detection, for which it is not intended, and requires implementing a security algorithm more complex and processor-intensive than the minimal CRC32c approach discussed here. As the Bundle Protocol has no separate outer error detection covering this security payload, if a secret key is used, then third-party intermediate nodes that do not know that secret key cannot determine the reliability of the content, and would be unable to prevent unnecessary forwarding of errored bundles belonging to non-error-tolerant applications. This may lead to decreased performance of the network due to utilization of storage, bandwidth, and power for unusable bundles, and poor end-to-end performance due to the open nature of its control loop as discussed above.

If early detection and discard of unusable bundles is done to prevent errors, this gives unencrypted bundle delivery a network performance advantage over secure bundle delivery using secret keys.

Applications wanting both security and network performance would gain both by implementing their own end-to-end security within unencrypted bundles using the insecure ciphersuites defined in this document, or by applying a reliability ciphersuite after applying a security ciphersuite.

The only PIB ciphersuite included in the BSP to date is PIB-RSA-SHA256, which creates and verifies signatures of bundles using RSA asymmetric-key operations over a SHA-256 hash of the bundle. The length of the SHA-256 output (32 octets) can be considered too large an overhead for providing simple error-detection on all but extremely large bundles. The processing overhead of SHA-256 and RSA calculations also discourages adopting these on computationally-constrained embedded systems. But the biggest problem with PIB-RSA-SHA256 is the bulk of code needed to support the RSA operations, which include math on numbers larger than that supported by common processors' native instruction sets and modular arithmetic libraries. Since error-detection and rejection is a vital and absolutely

essential component of reliable networking protocols, and much of the purpose of the DTN architecture is to enable internetworking of devices with limited resources, e.g. motes, it would be burdensome on limited low-end embedded systems to require all Bundle Protocol implementations to include RSA code.

The BAB-HMAC ciphersuite that uses SHA1 [[RFC3174](#)] within the HMAC construction (HMAC-SHA1) has been specified as mandatory for BSP support. Even though the BAB is not appropriate for end-to-end

error-detection, it is certain that BSP implementations will include HMAC-SHA1 routines, and that creating another ciphersuite for PIB-HMAC (which does not exist in the base BSP specification) would impose very little additional code. Partial support for the BSP's elements (at least the PIB's format and mutable canonicalization) could be made mandatory in a future revision of the Bundle Protocol along with support for the PIB-HMAC ciphersuite with NULL keys while retaining as optional all of the other components of the BSP (BAB, PCB, and other ciphersuites). Such a path seems to be desirable in that it allows re-use of existing code along with re-use of existing specifications, but does not significantly burden lightweight implementations or deployments unconcerned with overlay-layer security. This approach is followed in [Section 3](#) within this document.

There are disadvantages to reusing the existing Bundle Security Protocol for reliability-only purposes. Some deployments on limited hardware within closed networks will not desire to run heavyweight security protocols, nor include the full BSP and its mandatory ciphersuites within their code footprints, decreasing interoperability and adoption of the BSP. There is a desire to avoid creating a false sense of security by using a mechanism labelled the Bundle _Security_ Protocol with either a ciphersuite or NULL key that provides absolutely no security services. For instance, if an implementation allowed this to be configured using the same mechanisms or policy directive configuration files, formats, etc. that are normally used to configure BSP mechanisms providing real security, then a misconfiguration or misunderstanding could have a negative security impact to an operational system.

In order to allay these concerns, it was decided to define simple error-detection ciphersuites with the string "INSECURE" in their

mnemonics and draw a line as to which portions of the BSP security framework become mandatory and which remain optional. This allows implementation of error-detection capabilities either with or without the majority of the BSP, and with reduced potential for misleading users with regards to security.

Implementations that provide both the full BSP and simple error-detection ciphersuites SHOULD ensure that their configuration by users is sufficiently dissimilar from the normal BSP configuration. Implementations MUST NOT implement the "INSECURE" ciphersuites in such a way that leads to their being construed as security mechanisms, e.g. in logging output or configuration directives.

[3.](#) INSECURE Ciphersuites

Any PIB ciphersuite providing only integrity checking for error-detection and using published (or "null") keys MUST contain the string "INSECURE" in its mnemonic. PIBs that use these ciphersuites are otherwise indistinguishable from PIBs used to implement security services. PIB-HMAC is keyed, and so does not use "INSECURE" in its name. When used with a secret key, PIB-HMAC is useful for security, although this is not the case when it is used with a NULL key, we assume that the presence of a NULL key in the configuration significantly alerts users to the fact that it is not providing security.

To provide the desired functionality, three new ciphersuites are defined in this document (PIB-HMAC used with either real or NULL keys, PIB-INSECURE-MD5, and PIB-INSECURE-CRC32). The motivations behind defining all three of these ciphersuites are outlined below in the more detailed description of each ciphersuite. All of the ciphersuites defined here use the mutable canonicalization algorithm that is defined in the BSP and compute their checksums over the canonical forms of bundles. If error-detection support is considered essential for use of the bundle protocol, this means that the minimal BSP elements that all Bundle Protocol implementations MUST support include:

1. Mutable Canonicalization - discussed in [Appendix A.1](#) of this document.
2. PIB wire-format - discussed in [Appendix A.2](#) of this document.
3. PIB-HMAC ciphersuite with NULL key definition, PIB-INSECURE-MD5 and PIB-INSECURE-CRC32 ciphersuite

The new ciphersuites are identified by the following ciphersuite IDs within the abstract security block:

- o 0x04 - PIB-HMAC
- o 0x05 - PIB-INSECURE-MD5
- o 0x06 - PIB-INSECURE-CRC32

PIB-HMAC is defined to use the same HMAC-SHA1 construction as the BSP's BAB-HMAC ciphersuite, and can thus leverage existing code. Three ciphersuite parameters are needed, all of which are SDNVs. The first SDNV is a key identifier. A zero value in the key identifier field of a PIB using the PIB-HMAC ciphersuite indicates that the algorithm is keyed with the special NULL key. The NULL key used here

is defined to be 0xc37b 7e64 9258 4340 bed1 2207 8089 4115 5068 f738, the same fixed NULL key used with LTP's NULL ciphersuite. The later two SDNVs are offsets describing the protected bits of the bundle, identical to the offset and length parameters describe in the BSP's PIB-RSA-SHA256 ciphersuite. The first identifies the first covered octet and the second identifies the last covered octet.

PIB-HMAC creates a ten-octet security result and should provide adequate error-detection capabilities for large bundles of at least several gigabytes in size. Its advantage lies in that the NULL-keyed version can be implemented with minor additions to existing BSP codebases, and support for HMAC-SHA1 is known to only require around two hundred lines of portable C code for implementations that do not already contain BSP support.

The existence of the PIB-INSECURE-MD5 ciphersuite is motivated by the fact that an MD5 hash can be computed on the order of twice as fast as a SHA1 hash over the same data, as demonstrated by benchmarking

activities [[RFC1810](#)], yet still yields robust error-detection over fairly large inputs. This may be desirable in environments that have only limited computational resources to expend on bundle generation or processing. For instance, the authors have implemented generation of bundles of up to several hundred megabytes in size, onboard an imaging satellite solid-state data recorder using only a 200 MHz processor. The PIB-INSECURE-MD5 parameters consist of two SDNVs, an offset and length, that convey the covered portion of the bundle, in an identical way to the corresponding PIB-HMAC and PIB-RSA-SHA256 parameters.

The security result included with the PIB-INSECURE-MD5 ciphersuite is a full 16-octet MD5 output. The longer security result than PIB-HMAC may provide better error-detection for very large bundles, in addition to being faster to compute. For small bundles, the lack of the HMAC construction's second application of the hash function also improves efficiency in PIB-INSECURE-MD5 compared to PIB-HMAC. Implementations of MD5 are known to require only around 200 lines of portable C code and are widely available as open-source and within the MD5 RFC [[RFC1321](#)].

The PIB-INSECURE-CRC32 ciphersuite is intended for small bundles, and SHOULD only be used on bundles whose payload length is less than 65535 bits, because its protection weakens for longer payloads due to the increased risk of collisions [[Koop02](#)]. The parameters included with this ciphersuite are identical to those used with PIB-INSECURE-MD5. The security result is computed using the CRC-32c algorithm, identically to that defined for use with SCTP [[RFC3309](#)]. The security result is always a 4-octet quantity when PIB-INSECURE-CRC32 is used. The advantage of the CRC used as a checksum in this

ciphersuite is that it implies a lower header overhead to in-flight or in-memory bundles with small payloads, in comparison to the length of the PIB-HMAC and PIB-INSECURE-MD5 results. This may be highly desirable in environments where small messaging bundles are normal and only bandwidth-limited links are available. The CRC-32c algorithm is known to be implementable in only a few dozen lines of portable C code.

[3.1](#). Generation and Processing Rules

Since the INSECURE ciphersuites and NULL-keyed PIB-HMAC use the same

block type code and format as the more secure uses of PIB, they inherit the existing generation and processing rules of the PIB. This is good from a security standpoint in two respects:

1. The existing PIB processing rules consider interaction with any BABs that might be added to a bundle and prevent interactions that would cause wrongful failure of MACs, signatures, and checksums.
2. The PIB and PCB processing rules remove the possibility of a MAC, signature, or checksum revealing information about the private contents of the PCB via the ordering of the applied security/error-detection transforms. This is discussed within the PCB-RSA-AES128-PAYLOAD-PIB ciphersuite definition in the BSP specification.

Although the BSP was intended as an optional suite of extensions to the Bundle Protocol, and only needed in cases where certain security services are desired at the bundle layer, a subset of its components is now required to implement the PIB-based error-detection mechanism. As (1) providing error detection at some place in the stack is needed by applications that require reliable delivery of payload content, (2) many conceivable applications require such delivery, and (3) the Bundle Protocol is a proposed new stack "waist in the hourglass", error detection is clearly very desirable in common implementations of the Bundle Protocol. Supporting error-free delivery does not require mandatory implementation of the full BSP and accompanying security ciphersuites, but only requires the PIB block format, and the mutable canonicalization rules. These two portions of the BSP are fully described in the BSP specification

[[I-D.irtf-dtnrg-bundle-security](#)] with some commentary regarding their use for error-detection in [Appendix A](#) of this document.

When non-error-tolerant applications request custody transfer, it is extremely desirable to use an unkeyed or NULL-keyed PIB ciphersuite defined in this document between the source and destination bundle agents, to ensure that the custodian now has a correct copy of the

bundle. Use of ciphersuites with secret keys shared only by end-hosts cannot assist in reliable hop-by-hop delivery, increasing the time to error detection at the end nodes and the time required for resends. For other bundles, not requiring custody transfer, an

unkeyed or NULL-keyed PIB ciphersuite SHOULD be used.

Checking of unkeyed or NULL-keyed PIBs at intermediate bundle agents SHOULD be performed, when possible, and an agent which fails to match the PIB security result within a bundle SHOULD immediately discard the bundle. This limits the wasted resources involved in propagating data now known to be errored. It is desirable that PIB hashes or checksums are recalculated whenever fragmented bundles are reassembled, to ensure that no damage was introduced by the fragmentation process. A future version of the Bundle Protocol might include a bundle processing flag that signals that errored-delivery is acceptable to a receiving application. However, the current version does not define such a flag. A future version of the Bundle Protocol specification might also define an administrative record that signals when a bundle has been dropped due to a corruption event detected via an unkeyed or NULL-keyed PIB check; that has not been defined in the current Bundle Protocol. The IRTF Delay-Tolerant Networking Research Group (DTNRG) has not achieved consensus on either of these possibilities at time of writing.

[4.](#) Performance Considerations

The normal method for handling error detection with security is to cover the encrypted payload with an outer error-detecting checksum wrapper. Use of an end-to-end secret key without a separate end-to-end outer error-detecting checksum prevents determination of the bundle's fidelity by any in-path forwarding nodes lacking that secret key. This discourages interoperability between parties that do not share keys, and consumes more network resources in relaying an errored bundle to the receiving destination end node, and in resending the bundle across its entire path once the error is finally detected at the destination on decryption. Custody transfer guaranteeing error-free receipt at intermediate nodes is not possible with secret keys in the mechanism outlined above. When secret keys are in use, errors introduced into bundles can only be detected at the decoding endpoint, rather than at intermediate nodes, meaning that resends across the entire network are requested far later when security is in use. This means that applications using unencrypted traffic can be expected to outperform applications using secret keys in DTN networks, thanks to their ability to detect errors earlier, their smaller resend control loops to get replacement bundles, and adding the capability for content verification to the use of custody transfer.

Again, an outer error-detecting checksum around the encrypted or unencrypted bundle prevents these problems, and allows custody transfer to be meaningful and indicate truly reliable receipt even for encrypted traffic where the encryption keys are not known by the custody agent. Having an error-detecting checksum cover a previously encrypted block allows the reliability of that block to be checked at intermediate nodes without requiring decryption key, leading to earlier detection of errors and earlier resends. This enables the intermediate network to help with increasing the throughput and performance of the applications at the end nodes. A worked example of this is given in [\[WEH09\]](#).

When secret keys are used for security, a second reliability-only insecure ciphersuite SHOULD be used across the encrypted payloads, to allow verification of correct delivery at intermediate nodes, to allow custody transfer to indicate reliable receipt of the encrypted content, and to increase the forwarding performance and efficiency of the overall network.

[5.](#) Security Considerations

This document has attempted to assuage any security concerns that would result from applying non-security-providing algorithms within a mechanism intended for security. This is accomplished through semantic overloading of the PIB, reusing its structure to hold a simple checksum when it is not intended to provide security services.

The potential leakage of information if checksums are not covered by some BSP confidentiality transform that is applied later in the transmission path is eliminated by the fact that the existing PIB block type code is used, and the BSP itself already contains rules for ensuring that confidentiality transforms applied by the PCB protect the security result fields within PIB instances.

This design decision to reuse a security block for error-detection may seem bizarre to both security and networking experts. However, this decision was necessitated by the late addition of checksum support to the Bundle Protocol. By the time interest in this subject arose within the DTNRG, the Bundle Protocol itself was in final review phases and had been implemented multiple times. When we began this work, the group did not have consensus that block validation and payload error detection even belonged in the Bundle Protocol itself. The Bundle Security Protocol was also no longer malleable enough to ensure compatibility with checksum support, as it had obtained a level of relative stability in its specification and there were existing implementation efforts based on these which could have

required modification in order to not pass checksums carried in a

non-BSP block as unprotected after performing a confidentiality transform of the payload.

In the authors' opinion, ideally, the error-detection functions would be implemented within the basic networking portions of the Bundle Protocol, and not as a subset of the security framework. However, the existing Bundle Protocol design was too well-established for the current definition of the Bundle Protocol to be affected. At this time, the Bundle Protocol has only been proposed as Experimental with some disclaimers. It is felt by some that future revisions of the Bundle Protocol need to provide mechanisms ensuring error detection for reliable delivery. In order to limit overhead, shorter checksums, e.g. CRC-16, could be used for small blocks, with longer checksums, e.g. MD5, reserved for large payload blocks. This would allow checksums to cover and provide confident processing of even blocks with mutable fields, and retain efficient updating in-network as a mutable field changes, without the recomputation also covering large unchanging payload blocks. This might also constrain the damage caused by errors to the functions provided by an individual block, rather than affecting the whole bundle and causing the whole bundle to be discarded, although the overall value of this is currently unknown.

The need to conserve limited network resources by detecting and avoiding further propagating errored bundles in-transit means that Bundle Agents SHOULD always validate checksums of in-flight bundles, even if the Agents are not the ultimate destination. This opens the door for a potential denial-of-service attack on DTN Bundle Agents by forcing them to expend computational cycles on bundles with large payloads. In this case, the attacker would also have to send these bundles over some link towards the target Bundle Agent, which will often be more constrained in bandwidth or availability than the Bundle Agent is in computational cycles, so this threat may be unrealistic, or better combatted through access-control on links. If this threat does turn out to be realistic in some set of circumstances, intermediary validation of PIBs was intentionally left as a SHOULD-level activity rather than a MUST, and could be dynamically disabled at some threshold of CPU use.

Using the same protocol mechanism to provide (1) error-detection

without security claims, (2) error-detection using a security protocol insecurely-keyed with a known NULL key, and (3) actual security protection using the same protocol but with secret keys, any of which can defined and used in the same "Payload Integrity Block", is confusing at best, and not a good clean-sheet approach to helping ensure secure configurations, interoperable implementations, or efficient handling of errored bundles.

Use of a NULL key is inferior to separately handling the security concerns of sender-authentication and integrity-protection from that of error-checking, as it opens the door to secret keys that prevent standalone error-detection, and should be discouraged. Also, the NULL key cannot provide error-detection needed for the mutable parts of the bundle. Providing any error detection for the mutable parts of the bundle has not been done here, and reliance on the fidelity of mutable fields and payloads should be avoided for this reason.

[6.](#) IANA Considerations

This document has no considerations for IANA.

[7.](#) Acknowledgements

Some of the work on this document was performed at NASA's Glenn Research Center under funding from the Earth Science Technology Office (ESTO) and the Space Communications Architecture Working Group (SCAWG).

Discussion in the DTNRG and particular suggestions from (alphabetically) Mike Demmer, Kevin Fall, Stephen Farrell, Darren Long, Peter Lovell, and Susan Symington guided the genesis of this document and were crucial to adding limited error-detection capabilities to the Bundle Protocol within the existing pre-established security framework.

[8.](#) References

[8.1.](#) Normative References

[I-D.irtf-dtnrg-bundle-security]

Symington, S., Farrell, S., Weiss, H., and P. Lovell,
"Bundle Security Protocol Specification",
[draft-irtf-dtnrg-bundle-security-19](#) (work in progress),
March 2011.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", [RFC 1321](#),
April 1992.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
Hashing for Message Authentication", [RFC 2104](#),
February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Eddy, et al.

Expires November 12, 2011

[Page 18]

Internet-Draft Reliability-only Checksum Ciphersuites

May 2011

Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1
(SHA1)", [RFC 3174](#), September 2001.

[RFC3309] Stone, J., Stewart, R., and D. Otis, "Stream Control
Transmission Protocol (SCTP) Checksum Change", [RFC 3309](#),
September 2002.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol
Specification", [RFC 5050](#), November 2007.

[8.2](#). Informative References

[I-D.irtf-dtnrg-sec-overview]

Farrell, S., Symington, S., Weiss, H., and P. Lovell,
"Delay-Tolerant Networking Security Overview",
[draft-irtf-dtnrg-sec-overview-06](#) (work in progress),
March 2009.

[I-D.irtf-dtnrg-tcp-clayer]

Demmer, M. and J. Ott, "Delay Tolerant Networking TCP
Convergence Layer Protocol",
[draft-irtf-dtnrg-tcp-clayer-02](#) (work in progress),
November 2008.

- [Koop02] Koopman, P., "32-bit cyclic redundancy codes for Internet applications", Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp. 459-468 , June 2002.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, [RFC 791](#), September 1981.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, [RFC 793](#), September 1981.
- [RFC1662] Simpson, W., "PPP in HDLC-like Framing", STD 51, [RFC 1662](#), July 1994.
- [RFC1810] Touch, J., "Report on MD5 Performance", [RFC 1810](#), June 1995.
- [RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", [RFC 4838](#), April 2007.
- [RFC5327] Farrell, S., Ramadas, M., and S. Burleigh, "Licklider

- Transmission Protocol - Security Extensions", [RFC 5327](#), September 2008.
- [SGHP98] Stone, J., Greenwald, M., Hughes, J., and C. Partridge, "Performance of checksums and CRCs over real data", IEEE Transactions on Networks vol. 6 issue 5, pp. 529-543, October 1998.
- [SP00] Stone, J. and C. Partridge, "When the CRC and TCP Checksum Disagree", Proceedings of ACM SIGCOMM , September 2000.
- [SRC84] Saltzer, J., Reed, D., and D. Clark, "End-to-end Arguments in System Design", ACM Transactions on Computer Systems 2 (4), November 1984.
- [WEH09] Wood, L., Eddy, W., and P. Holliday, "A Bundle of Problems", IEEE Aerospace conference, Big Sky, Montana , March 2009.

[Appendix A](#). Mandatory BSP Elements Needed to Implement Error-Detection

This document makes some BSP components mandatory to Bundle Protocol implementations, in that while their use is optional they must be supported for interoperability reasons. Previously, the BSP was previously entirely optional, meaning that bundle protocol implementations could have no internal integrity mechanisms. This appendix discusses these elements in greater detail, and highlights some further drawbacks of basing error detection and protocol reliability upon these elements.

[A.1](#). Discussion of Mutable Canonicalization

Mutable Canonicalisation is defined by the Bundle Security Protocol.

While impressively named, the mutable canonicalization procedure should actually be quite simple to understand. The requirement for mutable canonicalization stems from the Bundle Protocol's forwarding design that allows several "mutable" fields (e.g. the "dictionary", custodian, and some flags and length fields), to change in-transit at intermediate nodes. In order for the checksum, MAC, or signature computed and placed in the security result of a sent PIB to match the result computed over the received bundle, the sender and receiver need to leave mutable fields out of these computations. The format of a bundle that is input to the PIB algorithms thus differs from its wire-format, and is called its "mutable canonicalization".

Using mutable canonicalization implies either using an incrementally-

updatable checksum algorithm and feeding many small pieces of data to it, or entirely rewriting a bundle block-by-block based on mutable canonicalization rules before feeding it to the checksum function. (The mutable fields still require protection against errors; a hop-by-hop checksum over only the mutable fields could be used to provide this. Hop-by-hop checksum coverage could be provided by a convergence layer or BAB, but this would likely cover the entire bundle or fragment.)

Several problems are known to plague mutable canonicalization:

1. [EDITOR'S NOTE: This concern may be easily fixed in updates to

the BSP document.] The Bundle Protocol specification describes the bundle processing control flags as a single variable-length SDNV whose bits are sub-divided in-order by function into "SRR" (Status Report Request), "COS" (Class of Service), and "General". The BSP's mutable canonicalization description shows three separate fields, with only slightly differing names, but in totally opposite order: "Proc. Flags", "COS Flags", and "SRR Flags", instead of one single SDNV, yet has text that describes operating on these as a single 64-bit value and applying a fixed-length mask to them. This is unclear at best, and feared to be uninteroperable in implementations.

2. Many bits within the bundle processing control flags are masked out (i.e. forced to zero within) the mutable canonicalization format. This includes all of the reserved class of service (COS) bits that are highly likely to be needed to overcome the limitation of having only three defined priority levels in the Bundle Specification (compare to DiffServ, CLNP's priority field, Aeronautical Mobile Radio Service message priorities, or mechanisms in other networking stacks that provide many more bits). This means that these bits, and any other bundle processing control bits, will be unprotected by the end-to-end checksum and may change in-transit, potentially causing mis-treatment or mal-delivery of bundles.
3. The existing "bundle is a fragment" bit is unprotected in mutable canonicalization. Errors in this bit itself can probably be caught through other means, such as careful length and bound checking in processing the rest of the bundle.
4. The entire mutable canonicalization procedure of parsing and re-formatting bundles in order to perform a checksum validation is significantly more complex than is typical in most existing protocols that are designed to be capable of simply computing a validation over a frame either without modifications [[RFC1662](#)], with only a small fixed-length and position field masked

[[RFC0791](#)], or with only a simple fixed-size pseudoheader [[RFC0793](#)]. The significant additional complexity of mutable canonicalization prevents high performance in forwarding nodes that follow the guideline of verifying unkeyed or NULL-keyed PIBs.

[A.2.](#) Discussion of the PIB Format

The PIB is defined by the Bundle Security Protocol.

PIBs follow the format of the abstract security block with a block type code that identifies them as PIBs. Some of the processing rules for PIBs that make the PIB less than ideal for error-detection purposes include:

1. If a PCB is placed into a bundle that already has a PSB, then another PCB is created that hides the PIB. This means that for end-to-end error-detection PIBs, any in-network security proxies that add PCB blocks also prevent the checksum in the PIB from being verifiable before the PCB's security destination recovers the cleartext PIB. If the PCB security destination is never reached, the bundle cannot be checked for errors. Errored bundles will consume resources between these two security gateways, since the errors cannot be detected and the bundles discarded en route. The RSA signature of such an errored bundle will only fail at the security destination, and the bundle will only be discarded at that end point, but there may be significant resources expended in delivering the useless bundle to that point.
2. A previously-generated PIB's security result cannot be retained outside a PCB in the clear, because an observer could correlate the value to some known probable plaintext payload value. It might be better to reverse the order of operations and always generate rewritten PIB ciphersuite checksums after generating PCBs that encrypt the payload, so that the PIB security result covers the PCB's encrypted form of the payload rather than the unencrypted form, and uses the same security destination as the PCB. Upon reaching this security destination, another PIB destined for the receiver, covering the payload revealed at the security destination, could be generated. Requiring this would allow detection of errored bundles between PCB security source and PCB security destination, but would involve adding another instruction to the PCB generation process within the BSP. This assumes no errors are introduced during the decryption process of the PCB, as such errors would go undetected. If bundles pass through nested security domains, this could compound the error rate.

There appears to be both benefits and drawbacks to any approach to PIB and PCB interaction that does not involve layering multiple PIBs that can be pushed and popped off of a bundle at various security sources and destinations. Pushing and popping nested PSBs approximates the outer checksum around inner security payload used successfully elsewhere in networking. By some reasonable metrics, the BSP-prescribed interaction that we have attempted to build on and fix here may be among the least desirable of all known methods for error detection.

Authors' Addresses

Wesley M. Eddy
MTI Systems
MS 500-ASRC
NASA Glenn Research Center
21000 Brookpark Road
Cleveland, Ohio 44135
United States of America

Phone: +1-216-433-6682
Email: wes@mti-systems.com

Lloyd Wood
Centre for Communication Systems Research, University of Surrey
Guildford, Surrey GU2 7XH
United Kingdom

Phone: +44-1483-689123
Email: L.Wood@surrey.ac.uk

Will Ivancic
NASA Glenn Research Center
21000 Brookpark Road
Cleveland, Ohio 44135
United States of America

Phone: +1-216-433-3494
Fax: +1-216-433-8705
Email: William.D.Ivancic@nasa.gov

