Internet Research Task Force (IRTF)
Request for Comments: 8609
Category: Experimental
ISSN: 2070-1721

M. Mosko
PARC, Inc.
I. Solis
LinkedIn
C. Wood
University of California Irvine

July 2019

Content-Centric Networking (CCNx) Messages in TLV Format

Abstract

Content-Centric Networking (CCNx) is a network protocol that uses a hierarchical name to forward requests and to match responses to requests. This document specifies the encoding of CCNx messages in a TLV packet format, including the TLV types used by each message element and the encoding of each value. The semantics of CCNx messages follow the encoding-independent CCNx Semantics specification.

This document is a product of the Information Centric Networking research group (ICNRG). The document received wide review among ICNRG participants and has two full implementations currently in active use, which have informed the technical maturity of the protocol specification.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Information-Centric Networking Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8609.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

This document specifies a Type-Length-Value (TLV) packet format and the TLV type and value encodings for CCNx messages. A full description of the CCNx network protocol, providing an encoding-free description of CCNx messages and message elements, may be found in [RFC8569]. CCNx is a network protocol that uses a hierarchical name to forward requests and to match responses to requests. It does not use endpoint addresses; the Internet Protocol does. Restrictions in a request can limit the response by the public key of the response's signer or the cryptographic hash of the response. Every CCNx forwarder along the path does the name matching and restriction checking. The CCNx protocol fits within the broader framework of Information-Centric Networking (ICN) protocols [RFC7927].

This document describes a TLV scheme using a fixed 2-byte T and a fixed 2-byte L field. The rational for this choice is described in Section 5. Briefly, this choice avoids multiple encodings of the same value (aliases) and reduces the work of a validator to ensure compliance. Unlike some uses of TLV in networking, each network hop must evaluate the encoding, so even small validation latencies at each hop could add up to a large overall forwarding delay. For very small packets or low-throughput links, where the extra bytes may become a concern, one may use a TLV compression protocol, for example, [compress] and [CCNxz].

This document uses the terms CCNx Packet, CCNx Message, and CCNx Message TLV. A CCNx Packet refers to the entire Layer 3 datagram as specified in Section 3.1. A CCNx Message is the ABNF token defined in the CCNx Semantics document [RFC8569]. A CCNx Message TLV refers to the encoding of a CCNx Message as specified in Section 3.6.

This document specifies:

- the CCNx Packet format,
- the CCNx Message TLV format,
- the TLV types used by CCNx messages,
- the encoding of values for each type,
- top-level types that exist at the outermost containment,
- Interest TLVs that exist within Interest containment, and
- Content Object TLVs that exist within Content Object containment.

This document is supplemented by these documents:

- [RFC8569], which covers message semantics and the protocol operation regarding Interest and Content Object, including the Interest Return protocol.
- [CCNxURI], which covers the CCNx URI notation.

The type values in Section 4 conform to the IANA-assigned numbers for the CCNx protocol. This document uses the symbolic names defined in that section. All TLV type values are relative to their parent containers. For example, each level of a nested TLV structure might define a "type = 1" with a completely different meaning.

Packets are represented as 32-bit wide words using ASCII art. Due to the nested levels of TLV encoding and the presence of optional fields and variable sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bit widths, which we typically pad out to word alignment for picture readability.

The document represents the consensus of the ICN RG. It is the first ICN protocol from the RG, created from the early CCNx protocol [nnc] with significant revision and input from the ICN community and RG members. The document has received critical reading by several members of the ICN community and the RG. The authors and RG chairs approve of the contents. The document is sponsored under the IRTF and is not issued by the IETF and is not an IETF standard. This is an experimental protocol and may not be suitable for any specific application and the specification may change in the future.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Definitions

These definitions summarize items defined in [RFC8569]. This document defines their encodings.

- Name: A hierarchically structured variable-length identifier. It is an ordered list of path segments, which are variable-length octet strings. In human-readable form, it is represented in URI format as "ccnx:/path/part". There is no host or query string. See [CCNxURI] for complete details.

- Interest: A message requesting a Content Object with a matching Name and other optional selectors to choose from multiple objects with the same Name. Any Content Object with a Name and attributes that matches the Name and optional selectors of the Interest is said to satisfy the Interest.

- Content Object: A data object sent in response to an Interest request. It has an optional Name and a content payload that are bound together via cryptographic means.

3. Type-Length-Value (TLV) Packets

We use 16-bit Type and 16-bit Length fields to encode TLV-based packets. This provides 65,536 different possible types and value field lengths of up to 64 KiB. With 65,536 possible types at each level of TLV encoding, there should be sufficient space for basic protocol types, while also allowing ample room for experimentation, application use, vendor extensions, and growth. This encoding does not allow for jumbo packets beyond 64 KiB total length. If used on a media that allows for jumbo frames, we suggest defining a media adaptation envelope that allows for multiple smaller frames.

Abbrev	Name	Description
T_ORG	Vendor Specific Information	Information specific to a vendor implementation (Section 3.3.2).
T_PAD	Padding	Adds padding to a field (Section 3.3.1).
n/a	Experimental	Experimental use.

Table 1: Reserved TLV Types

There are several global TLV definitions that we reserve at all hierarchical contexts. The TLV types in the range 0x1000 - 0x1FFF are Reserved for Experimental Use. The TLV type T_ORG is also Reserved for Vendor Extensions (see Section 3.3.2). The TLV type T_PAD is used to optionally pad a field out to some desired alignment.

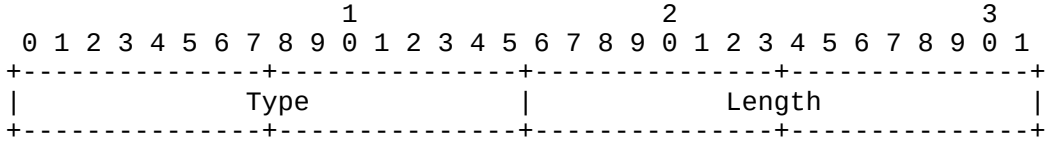

Figure 1: Type and Length encoding

The Length field contains the length of the Value field in octets. It does not include the length of the Type and Length fields. The Length MAY be zero.

TLV structures are nestable, allowing the Value field of one TLV structure to contain additional TLV structures. The enclosing TLV structure is called the container of the enclosed TLV.

Type values are context dependent. Within a TLV container, one may reuse previous type values for new context-dependent purposes.

3.1. Overall Packet Format

Each CCNx Packet includes the 8-byte fixed header, described below, followed by a set of TLV fields. These fields are optional hop-by-hop headers and the Packet Payload.

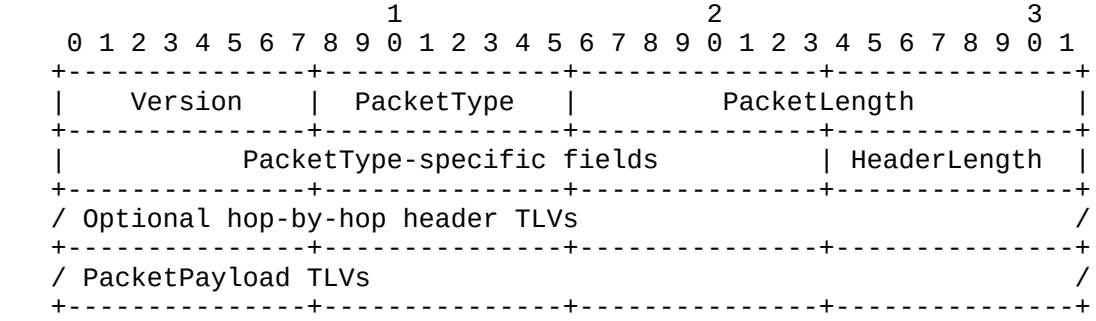

Figure 2: Overall Packet Format

The PacketPayload of a CCNx Packet is the protocol message itself. The Content Object Hash is computed over the PacketPayload only, excluding the fixed and hop-by-hop headers, as those might change from hop to hop. Signed information or similarity hashes should not include any of the fixed or hop-by-hop headers. The PacketPayload should be self-sufficient in the event that the fixed and hop-by-hop headers are removed. See Message Hash (Section 3.4.3).

Following the CCNx Message TLV, the PacketPayload may include optional Validation TLVs.

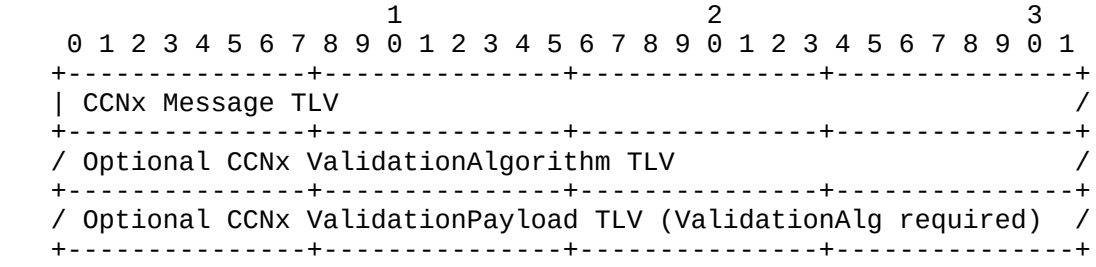

Figure 3: PacketPayload TLVs

After discarding the fixed and hop-by-hop headers, the remaining PacketPayload should be a valid protocol message. Therefore, the PacketPayload always begins with 4 bytes of type-length that specifies the protocol message (whether it is an Interest, Content Object, or other message type) and its total length. The embedding

of a self-sufficient protocol data unit inside the fixed and hop-by-hop headers allows a network stack to discard the headers and operate only on the embedded message. It also decouples the PacketType field -- which specifies how to forward the packet -- from the PacketPayload.

The range of bytes protected by the Validation includes the CCNx Message TLV and the ValidationAlgorithm TLV.

The ContentObjectHash begins with the CCNx Message TLV and ends at the tail of the CCNx Packet.

3.2. Fixed Headers

In Figure 2, the fixed header fields are:

- Version: defines the version of the packet, which MUST be 1.
- HeaderLength: The length of the fixed header (8 bytes) and hop-by-hop headers. The minimum value MUST be 8.
- PacketType: describes forwarder actions to take on the packet.
- PacketLength: Total octets of packet including all headers (fixed header plus hop-by-hop headers) and protocol message.
- PacketType-specific Fields: specific PacketTypes define the use of these bits.

The PacketType field indicates how the forwarder should process the packet. A Request Packet (Interest) has PacketType PT_INTEREST, a Response (Content Object) has PacketType PT_CONTENT, and an Interest Return has PacketType PT_RETURN.

HeaderLength is the number of octets from the start of the CCNx Packet (Version) to the end of the hop-by-hop headers. PacketLength is the number of octets from the start of the packet to the end of the packet. Both lengths have a minimum value of 8 (the fixed header itself).

The PacketType-specific fields are reserved bits whose use depends on the PacketType. They are used for network-level signaling.

3.2.1. Interest Fixed Header

If the PacketType is PT_INTEREST, it indicates that the packet should be forwarded following the Interest pipeline in Section 2.4.4 of [RFC8569]. For this type of packet, the Fixed Header includes a field for a HopLimit as well as Reserved and Flags fields. The Reserved field MUST be set to 0 in an Interest. There are currently no flags defined, so the Flags field MUST be set to 0.

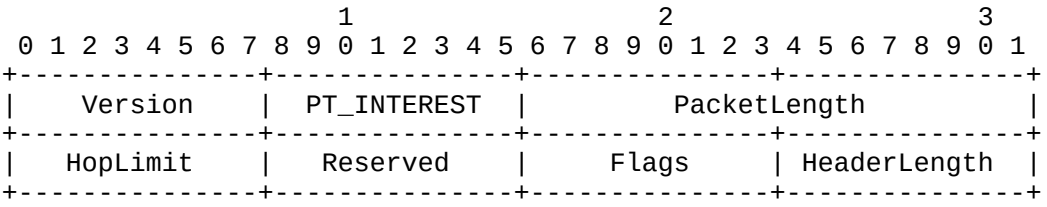

Figure 4: Interest Header

3.2.1.1. Interest HopLimit

For an Interest message, the HopLimit is a counter that is decremented with each hop. It limits the distance an Interest may travel on the network. The node originating the Interest MAY put in any value up to the maximum of 255. Each node that receives an Interest with a HopLimit decrements the value upon reception. If the value is 0 after the decrement, the Interest MUST NOT be forwarded off the node.

It is an error to receive an Interest from a remote node with the HopLimit field set to 0.

3.2.2. Content Object Fixed Header

If the PacketType is PT_CONTENT, it indicates that the packet should be forwarded following the Content Object pipeline in Section 2.4.4 of [RFC8569]. A Content Object defines a Flags field; however, there are currently no flags defined, so the Flags field must be set to 0.

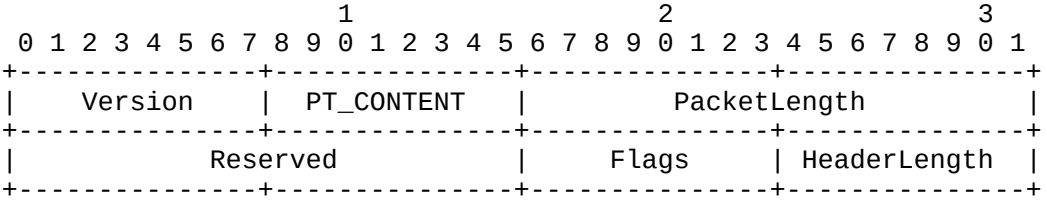

Figure 5: Content Object Header

3.2.3. Interest Return Fixed Header

If the PacketType is PT_RETURN, it indicates that the packet should be processed following the Interest Return rules in Section 10 of [RFC8569]. The only difference between this Interest Return message and the original Interest is that the PacketType is changed to PT_RETURN and a ReturnCode is put into the ReturnCode field. All other fields are unchanged from the Interest packet. The purpose of this encoding is to prevent packet length changes so no additional bytes are needed to return an Interest to the previous hop.

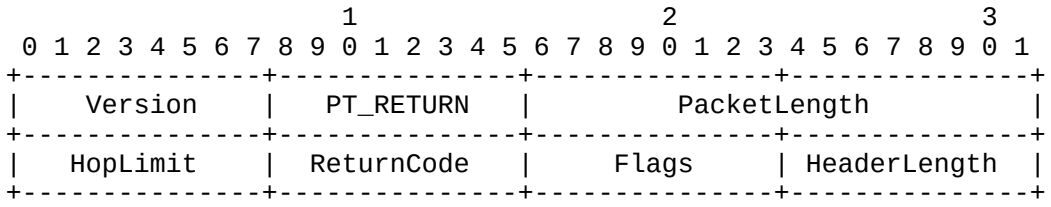

Figure 6: Interest Return Header

3.2.3.1. Interest Return HopLimit

This is the original Interest's HopLimit, as received before decrement at the node sending the Interest Return.

3.2.3.2. Interest Return Flags

These are the original Flags as set in the Interest.

3.2.3.3. Return Code

This section maps the Return Code name [RFC8569] to the TLV symbolic name. Section 4.2 maps the symbolic names to numeric values. This field is set by the node creating the Interest Return.

A return code of "0" MUST NOT be used, as it indicates that the returning system did not modify the Return Code field.

Return Type	Name in RFC 8569
T_RETURN_NO_ROUTE	No Route
T_RETURN_LIMIT_EXCEEDED	Hop Limit Exceeded
T_RETURN_NO_RESOURCES	No Resources
T_RETURN_PATH_ERROR	Path Error
T_RETURN_PROHIBITED	Prohibited
T_RETURN_CONGESTED	Congested
T_RETURN_MTU_TOO_LARGE	MTU too large
T_RETURN_UNSUPPORTED_HASH_RESTRICTION	Unsupported ContentObjectHashRestriction
T_RETURN_MALFORMED_INTEREST	Malformed Interest

Table 2: Return Codes

3.3. Global Formats

This section defines global formats that may be nested within other TLVs.

3.3.1. Pad

The pad type may be used by sources that prefer word-aligned data. Padding 4-byte words, for example, would use a 1-byte, 2-byte, and 3-byte Length. Padding 8-byte words would use a (0, 1, 2, 3, 5, 6, 7)-byte Length.

One MUST NOT pad inside a Name. Apart from that, a pad MAY be inserted after any other TLV in the CCNx Message TLV or in the ValidationAlgorithm TLV. In the remainder of this document, we will not show optional Pad TLVs.

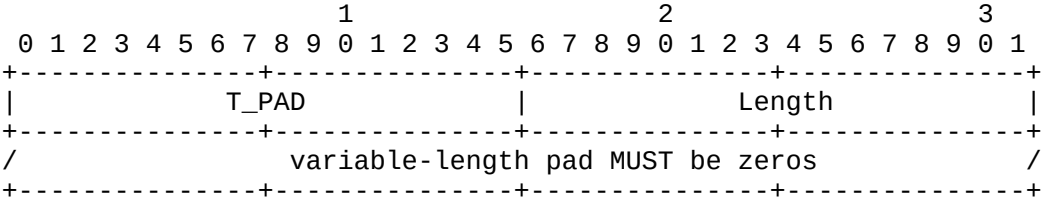

Figure 7: Pad Encoding

3.3.2. Organization-Specific TLVs

Organization-specific TLVs (also known as Vendor TLVs) MUST use the T_ORG type. The Length field is the length of the organization-specific information plus 3. The Value begins with the 3 byte organization number derived from the network byte order encoding of the IANA "Private Enterprise Numbers" registry [IANA-PEN], followed by the organization-specific information.

A T_ORG MAY be used as a path segment in a Name. It is treated like any other path segment.

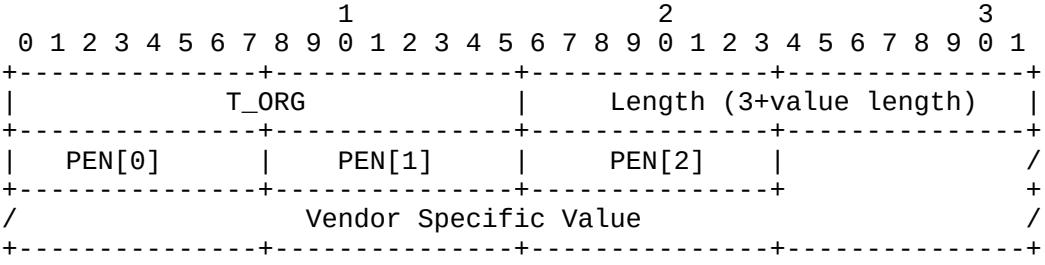

Figure 8: Organization-Specific TLVs

3.3.3. Hash Format

Hash values are used in several fields throughout a packet. This TLV encoding is commonly embedded inside those fields to specify the specific hash function used and its value. Note that the reserved TLV types are also reserved here for user-defined experimental functions.

The LENGTH field of the hash value MUST be less than or equal to the hash function length. If the LENGTH is less than the full length, it is taken as the left LENGTH bytes of the hash function output. Only specified truncations are allowed, not arbitrary truncations.

This nested format is used because it allows binary comparison of hash values for certain fields without a router needing to understand a new hash function. For example, the KeyIdRestriction is bit-wise compared between an Interest's KeyIdRestriction field and a ContentObject's KeyId field. This format means the outer field values do not change with differing hash functions so a router can still identify those fields and do a binary comparison of the hash TLV without need to understand the specific hash used. An alternative approach, such as using T_KEYID_SHA512-256, would require each router keeps an up-to-date parser and supporting user-defined hash functions here would explode the parsing state-space.

A CCNx entity MUST support the hash type T_SHA-256. An entity MAY support the remaining hash types.

Abbrev	Lengths (octets)
T_SHA-256	32
T_SHA-512	64, 32
n/a	Experimental TLV types

Table 3: CCNx Hash Functions

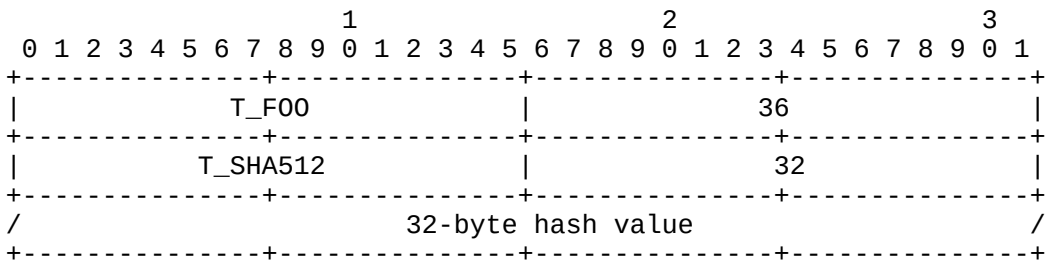

Figure 9: Example nesting inside type T_FOO

3.3.4. Link

A Link is the tuple: {Name, [KeyIdRestr], [ContentObjectHashRestr]}. It is a general encoding that is used in both the payload of a Content Object with PayloadType = "Link" and in a Content Object's KeyLink field. A Link is essentially the body of an Interest.

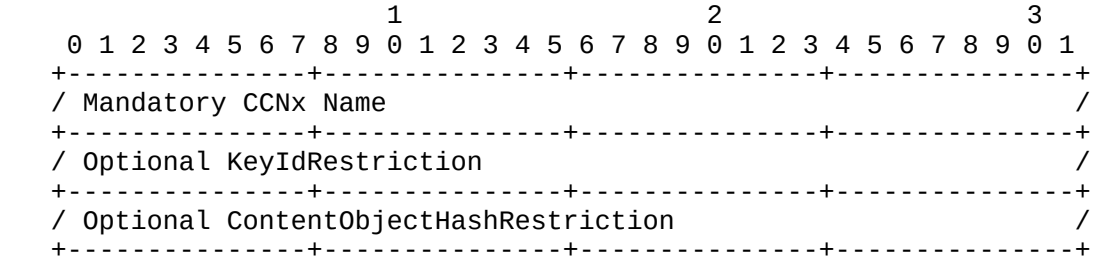

Figure 10: Link Encoding

3.4. Hop-by-Hop TLV Headers

Hop-by-hop TLV headers are unordered and meaning MUST NOT be attached to their ordering. Three hop-by-hop headers are described in this document:

Abbrev	Name	Description
T_INTLIFE	Interest Lifetime (Section 3.4.1)	The time an Interest should stay pending at an intermediate node.
T_CACHETIME	Recommended Cache Time (Section 3.4.2)	The Recommended Cache Time for Content Objects.
T_MSGHASH	Message Hash (Section 3.4.3)	A cryptographic hash (Section 3.3.3).

Table 4: Hop-by-Hop Header Types

Additional hop-by-hop headers are defined in higher level specifications such as the fragmentation specification.

3.4.1. Interest Lifetime

The Interest Lifetime is the time that an Interest should stay pending at an intermediate node. It is expressed in milliseconds as an unsigned integer in network byte order.

A value of 0 (encoded as 1 byte 0x00) indicates the Interest does not elicit a Content Object response. It should still be forwarded, but no reply is expected and a forwarder could skip creating a PIT entry.

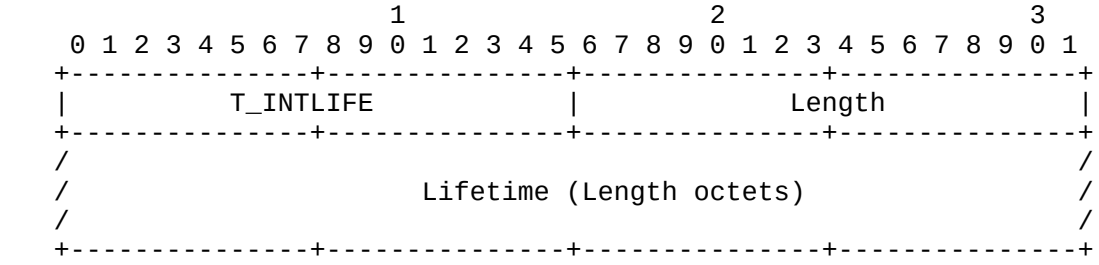

Figure 11: Interest Lifetime Encoding

3.4.2. Recommended Cache Time

The Recommended Cache Time (RCT) is a measure of the useful lifetime of a Content Object as assigned by a content producer or upstream node. It serves as a guideline to the Content Store cache in determining how long to keep the Content Object. It is a recommendation only and may be ignored by the cache. This is in contrast to the ExpiryTime (described in Section 3.6.2.2.2) which takes precedence over the RCT and must be obeyed.

Because the Recommended Cache Time is an optional hop-by-hop header and not a part of the signed message, a content producer may re-issue a previously signed Content Object with an updated RCT without needing to re-sign the message. There is little ill effect from an attacker changing the RCT as the RCT serves as a guideline only.

The Recommended Cache Time (a millisecond timestamp) is an unsigned integer in network byte order that indicates the time when the payload expires (as the number of milliseconds since the epoch in UTC). It is a 64-bit field.

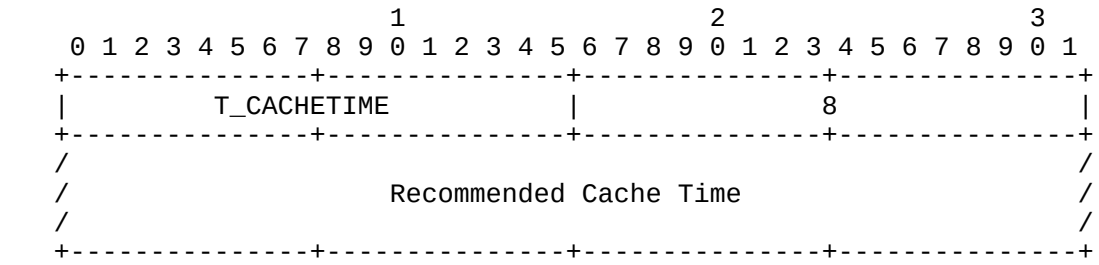

Figure 12: Recommended Cache Time Encoding

3.4.3. Message Hash

Within a trusted domain, an operator may calculate the message hash at a border device and insert that value into the hop-by-hop headers of a message. An egress device should remove the value. This permits intermediate devices within that trusted domain to match against a ContentObjectHashRestriction without calculating it at every hop.

The message hash is a cryptographic hash from the start of the CCNx Message TLV to the end of the packet. It is used to match against the ContentObjectHashRestriction (Section 3.6.2.1.2). The Message Hash may be of longer length than an Interest's restriction, in which case the device should use the left bytes of the Message Hash to check against the Interest's value.

The Message Hash may only carry one hash type and there may only be one Message Hash header.

The Message Hash header is unprotected, so this header is only of practical use within a trusted domain, such as an operator's autonomous system.

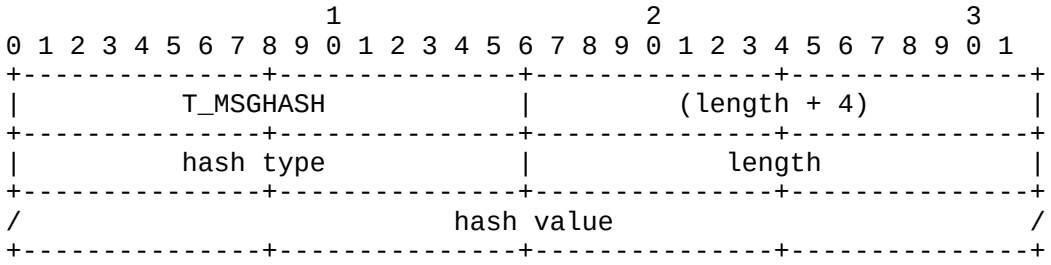

Figure 13: Message Hash Header

3.5. Top-Level Types

The top-level TLV types listed below exist at the outermost level of a CCNx Message TLV.

Abbrev	Name	Description
T_INTEREST	Interest (Section 3.6)	An Interest MessageType.
T_OBJECT	Content Object (Section 3.6)	A Content Object MessageType
T_VALIDATION_ALG	Validation Algorithm (Section 3.6.4.1)	The method of message verification such as a Message Integrity Check (MIC), Message Authentication Code (MAC), or cryptographic signature.
T_VALIDATION_PAYLOAD	Validation Payload (Section 3.6.4.2)	The validation output, such as the CRC32C code or the RSA signature.

Table 5: CCNx Top Level Types

3.6. CCNx Message TLV

This is the format for the CCNx Message itself. The CCNx Message TLV is the portion of the CCNx Packet between the hop-by-hop headers and the Validation TLVs. The figure below is an expansion of the "CCNx Message TLV" depicted in the beginning of Section 3. The CCNx Message TLV begins with MessageType and runs through the optional Payload. The same general format is used for both Interest and Content Object messages which are differentiated by the MessageType field. The first enclosed TLV of a CCNx Message TLV is always the Name TLV, if present. This is followed by an optional Message TLVs and an optional Payload TLV.

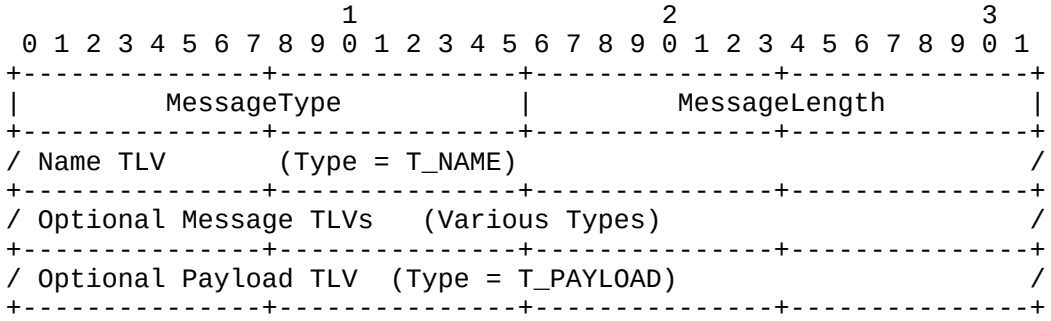

Figure 14: CCNx Message TLV Encoding

Abbrev	Name	Description
T_NAME	Name (Section 3.6.1)	The CCNx Name requested in an Interest or published in a Content Object.
T_PAYLOAD	Payload (Section 3.6.3)	The message payload.

Table 6: CCNx Message TLV Types

3.6.1. Name

A Name is a TLV encoded sequence of segments. The table below lists the type values appropriate for these name segments. A Name MUST NOT include Pad TLVs.

As described in CCNx Semantics [RFC8569], using the CCNx URI [CCNxURI] notation, a T_NAME with zero length corresponds to "ccnx:/" (the default route). The message grammar does not allow the first name segment to have zero length in a CCNx Message TLV Name. In the TLV encoding, "ccnx:/" corresponds to T_NAME with zero length.

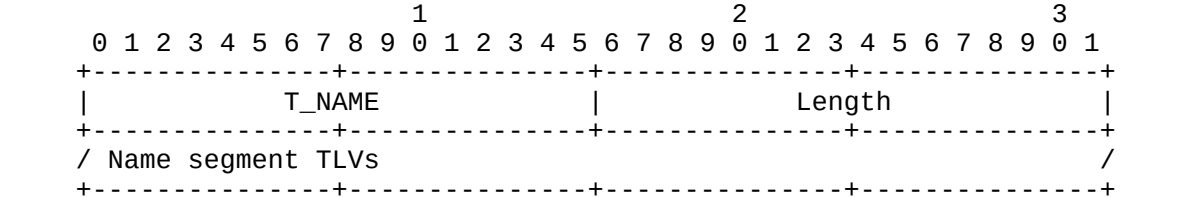

Figure 15: Name Encoding

Symbolic Name	Name	Description
T_NAMESEGMENT	Name segment (Section 3.6.1.1)	A generic name segment.
T_IPID	Interest Payload ID (Section 3.6.1.2)	An identifier that represents the Interest Payload field. As an example, the Payload ID might be a hash of the Interest Payload. This provides a way to differentiate between Interests based on their payloads without having to parse all the bytes of the payload itself, and instead using only this Payload ID name segment.
T_APP:00 - T_APP:4096	Application Components (Section 3.6.1.1)	Application-specific payload in a name segment. An application may apply its own semantics to the 4096 reserved types.

Table 7: CCNx Name Types

3.6.1.1. Name Segments

4096 special application payload name segments are allocated. These have application semantics applied to them. A good convention is to put the application's identity in the name prior to using these name segments.

For example, a name like "ccnx:/foo/bar/hi" would be encoded as:

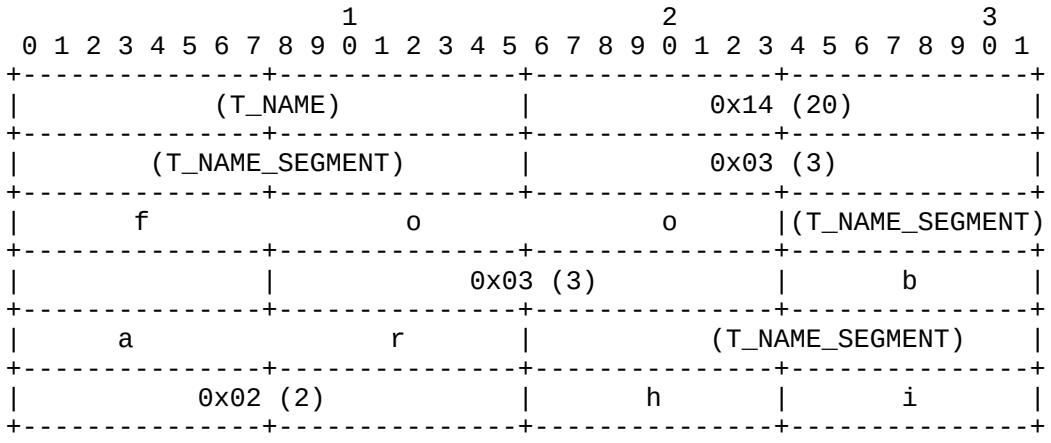

Figure 16: Name Encoding Example

3.6.1.2. Interest Payload ID

The InterestPayloadID is a name segment created by the origin of an Interest to represent the Interest Payload. This allows the proper multiplexing of Interests based on their name if they have different payloads. A common representation is to use a hash of the Interest Payload as the InterestPayloadID.

As part of the Value of the TLV, the InterestPayloadID contains a one-octet identifier of the method used to create the InterestPayloadID followed by a variable-length octet string. An implementation is not required to implement any of the methods to receive an Interest; the InterestPayloadID may be treated only as an opaque octet string for the purposes of multiplexing Interests with different payloads. Only a device creating an InterestPayloadID name segment or a device verifying such a segment needs to implement the algorithms.

It uses the encoding of hash values specified in Section 3.3.3.

In normal operations, we recommend displaying the InterestPayloadID as an opaque octet string in a CCNx URI, as this is the common denominator for implementation parsing.

The InterestPayloadID, even if it is a hash, should not convey any security context. If a system requires confirmation that a specific entity created the InterestPayload, it should use a cryptographic signature on the Interest via the ValidationAlgorithm and ValidationPayload or use its own methods inside the Interest Payload.

3.6.2. Message TLVs

Each message type (Interest or Content Object) is associated with a set of optional Message TLVs. Additional specification documents may extend the types associated with each.

3.6.2.1. Interest Message TLVs

There are two Message TLVs currently associated with an Interest message: the KeyIdRestriction selector and the ContentObjectHashRestr selector are used to narrow the universe of acceptable Content Objects that would satisfy the Interest.

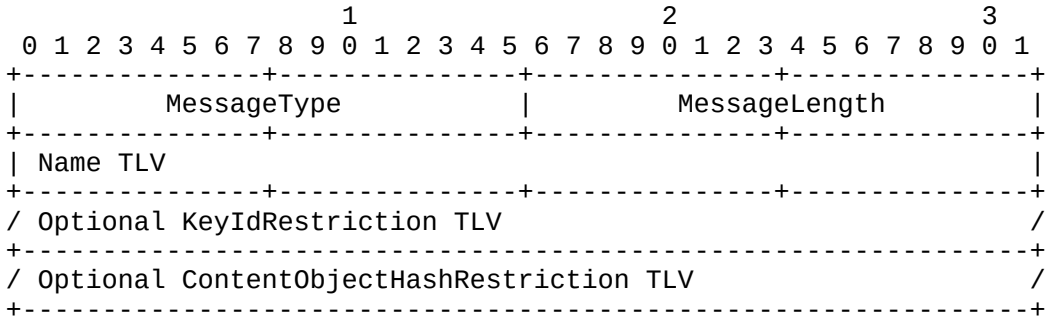

Figure 17: Interest Message TLVs

Abbrev	Name	Description
T_KEYIDRESTR	KeyIdRestriction (Section 3.6.2.1.1)	A representation (as per Section 3.3.3) of the KeyId
T_OBJHASHRESTR	ContentObjectHashRestriction (Section 3.6.2.1.2)	A representation (as per Section 3.3.3) of the hash of the specific Content Object that would satisfy the Interest.

Table 8: CCNx Interest Message TLV Types

3.6.2.1.1. KeyIdRestriction

An Interest MAY include a KeyIdRestriction selector. This ensures that only Content Objects with matching KeyIds will satisfy the Interest. See Section 3.6.4.1.4.1 for the format of a KeyId.

3.6.2.1.2. ContentObjectHashRestriction

An Interest MAY contain a ContentObjectHashRestriction selector. This is the hash of the Content Object -- the self-certifying name restriction that must be verified in the network, if an Interest carried this restriction (see Message Hash (Section 3.4.3)). The LENGTH MUST be from one of the allowed values for that hash (see Section 3.3.3).

The ContentObjectHashRestriction SHOULD be of type T_SHA-256 and of length 32 bytes.

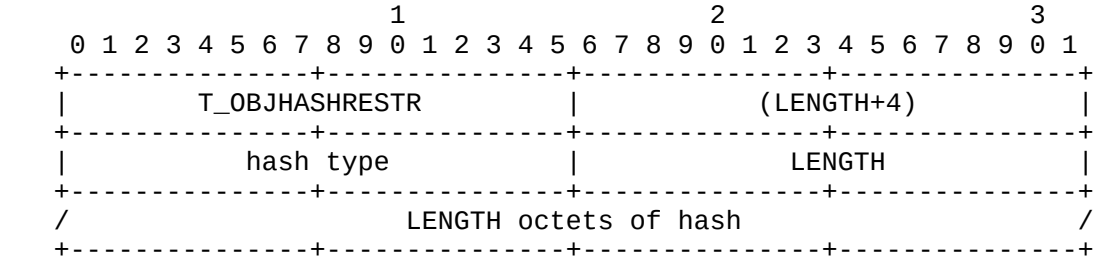

Figure 18: ContentObjectHashRestriction Encoding

3.6.2.2. Content Object Message TLVs

The following message TLVs are currently defined for Content Objects: PayloadType (optional) and ExpiryTime (optional).

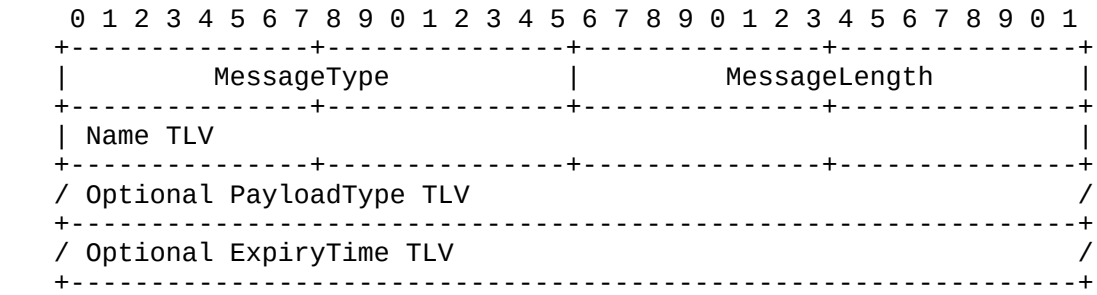

Figure 19: Content Object Message TLVs

Abbrev	Name	Description
T_PAYLDTYPE	PayloadType (Section 3.6.2.2.1)	Indicates the type of Payload contents.
T_EXPIRY	ExpiryTime (Section 3.6.2.2.2)	The time at which the Payload expires, as expressed in the number of milliseconds since the epoch in UTC. If missing, Content Object may be used as long as desired.

Table 9: CCNx Content Object Message TLV Types

3.6.2.2.1. PayloadType

The PayloadType is an octet representing the general type of the Payload TLV.

- T_PAYLOADTYPE_DATA: Data (possibly encrypted)
- T_PAYLOADTYPE_KEY: Key
- T_PAYLOADTYPE_LINK: Link

The Data type indicates that the Payload of the ContentObject is opaque application bytes. The Key type indicates that the Payload is a DER-encoded public key. The Link type indicates that the Payload is one or more Links (Section 3.3.4). If this field is missing, a Data type is assumed.

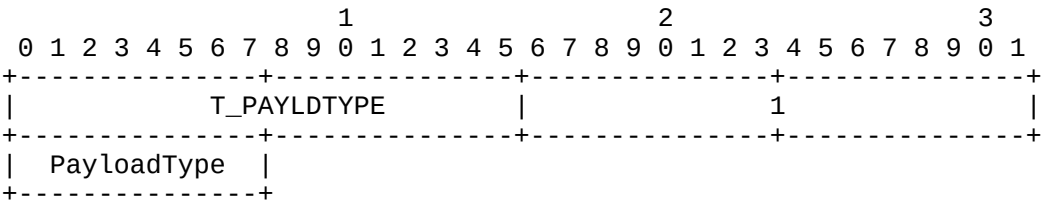

Figure 20: PayloadType Encoding

3.6.2.2.2. ExpiryTime

The ExpiryTime is the time at which the Payload expires, as expressed by a timestamp containing the number of milliseconds since the epoch in UTC. It is a network byte order unsigned integer in a 64-bit field. A cache or end system should not respond with a Content Object past its ExpiryTime. Routers forwarding a Content Object do not need to check the ExpiryTime. If the ExpiryTime field is missing, the Content Object has no expressed expiration, and a cache or end system may use the Content Object for as long as desired.

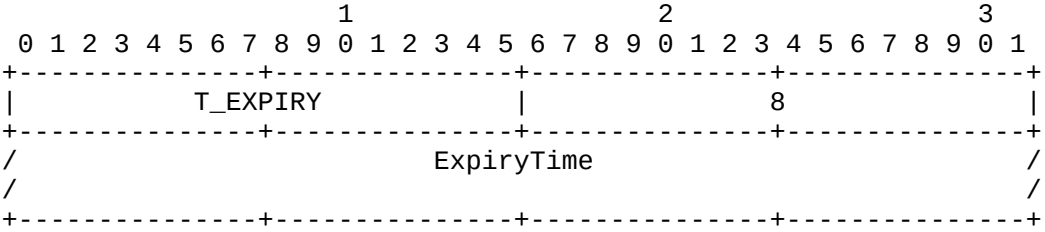

Figure 21: ExpiryTime encoding

3.6.3. Payload

The Payload TLV contains the content of the packet. It MAY be of zero length. If a packet does not have any payload, this field SHOULD be omitted, rather than being of zero length.

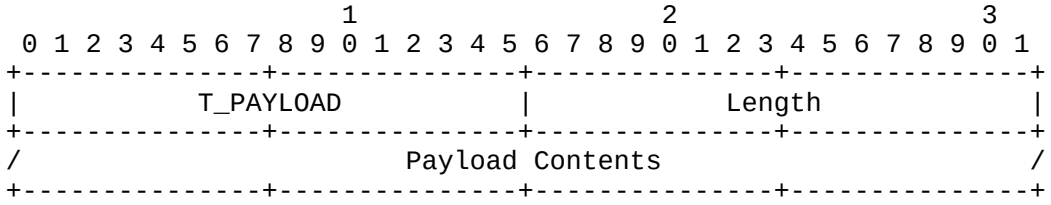

Figure 22: Payload Encoding

3.6.4. Validation

Both Interests and Content Objects have the option to include information about how to validate the CCNx Message. This information is contained in two TLVs: the ValidationAlgorithm TLV and the ValidationPayload TLV. The ValidationAlgorithm TLV specifies the mechanism to be used to verify the CCNx Message. Examples include verification with a Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature. The ValidationPayload TLV contains the validation output, such as the CRC32C code or the RSA signature.

An Interest would most likely only use a MIC type of validation -- a CRC, checksum, or digest.

3.6.4.1. Validation Algorithm

The ValidationAlgorithm is a set of nested TLVs containing all of the information needed to verify the message. The outermost container has type = T_VALIDATION_ALG. The first nested TLV defines the specific type of validation to be performed on the message. The type is identified with the "ValidationType" as shown in the figure below and elaborated in the table below. Nested within that container are the TLVs for any ValidationType-dependent data -- for example, a Key Id, Key Locator, etc.

Complete examples of several types may be found in Section 3.6.4.1.5.

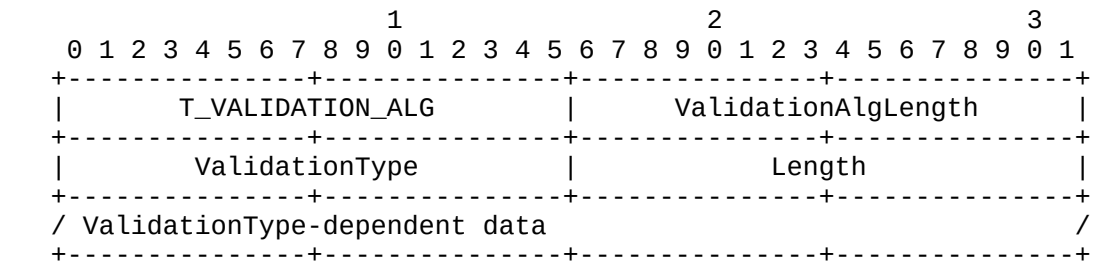

Figure 23: Validation Algorithm Encoding

Abbrev	Name	Description
T_CRC32C	CRC32C (Section 3.6.4.1.1)	Castagnoli CRC32 (iSCSI, ext4, etc.) with normal form polynomial 0x1EDC6F41.
T_HMAC-SHA256	HMAC-SHA256 (Section 3.6.4.1.2)	HMAC (RFC 2104) using SHA256 hash.
T_RSA-SHA256	RSA-SHA256 (Section 3.6.4.1.3)	RSA public-key signature using SHA256 digest.
T_EC-SECP-256K1	SECP-256K1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-256K1 parameters (see [ECC]).
T_EC-SECP-384R1	SECP-384R1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-384R1 parameters (see [ECC]).

Table 10: CCNx Validation Types

3.6.4.1.1. Message Integrity Checks

MICs do not require additional data in order to perform the verification. An example is CRC32C that has a zero-length value.

3.6.4.1.2. Message Authentication Codes

MACs are useful for communication between two trusting parties who have already shared secret keys. An example is the HMAC algorithm. A MAC uses the KeyId field to identify which shared secret is in use. The meaning of the KeyId is specific to the two parties involved and could be simply an integer to enumerate keys. If a new MAC requires an additional field, such as an Initialization Vector, that field would need to be defined as part of the updated specification.

3.6.4.1.3. Signature

Signature type Validators specify a digest mechanism and a signing algorithm to verify the message. Examples include an RSA signature on a SHA256 digest, an Elliptic Curve signature with SECP-256K1 parameters, etc. These Validators require a KeyId and a mechanism for locating the publisher's public key (a KeyLocator) -- and optionally a PublicKey or Certificate or KeyLink.

3.6.4.1.4. Validation-Dependent Data

Different Validation Algorithms require access to different pieces of data contained in the ValidationAlgorithm TLV. As described above, Key Ids, Key Locators, Public Keys, Certificates, Links, and Key Names all play a role in different Validation Algorithms. Any number of Validation-Dependent Data containers can be present in a Validation Algorithm TLV.

Below is a table of CCNx ValidationType-dependent data types:

Abbrev	Name	Description
T_KEYID	SignerKeyId (Section 3.6.4.1.4.1)	An identifier of the shared secret or public key associated with a MAC or Signature.
T_PUBLICKEY	Public Key (Section 3.6.4.1.4.2)	DER-encoded public key.
T_CERT	Certificate (Section 3.6.4.1.4.3)	DER-encoded X.509 certificate.
T_KEYLINK	KeyLink (Section 3.6.4.1.4.4)	A CCNx Link object.
T_SIGTIME	SignatureTime (Section 3.6.4.1.4.5)	A millisecond timestamp indicating the time when the signature was created.

Table 11: CCNx Validation-Dependent Data Types

3.6.4.1.4.1. KeyId

The KeyId for a signature is the publisher key identifier. It is similar to a Subject Key Identifier from X.509 (see Section 4.2.1.2 of [RFC5280]). It should be derived from the key used to sign, such as from the SHA-256 hash of the key. It applies to both public and private key systems and to symmetric key systems.

The KeyId is represented using the hash format in Section 3.3.3. If an application protocol uses a non-hash identifier, it should use one of the reserved values.

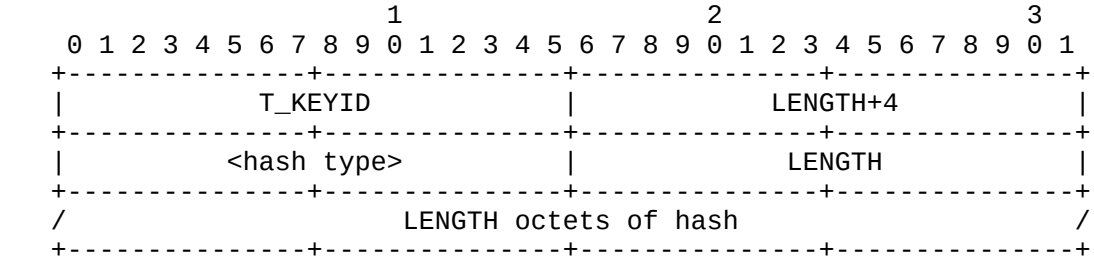

Figure 24: KeyId Encoding

3.6.4.1.4.2. Public Key

A Public Key is a DER-encoded Subject Public Key Info block, as in an X.509 certificate.

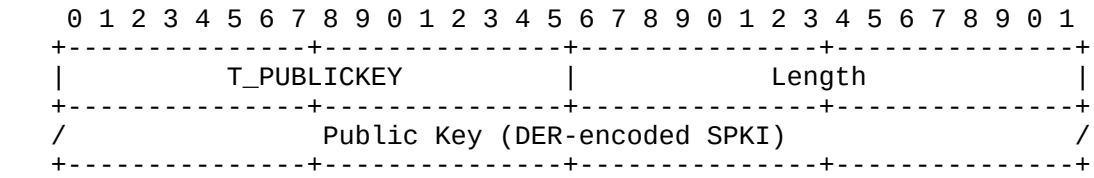

Figure 25: Public Key Encoding

3.6.4.1.4.3. Certificate

A Certificate is a DER-encoded X.509 certificate. The KeyId (Section 3.6.4.1.4.1) is derived from this encoding.

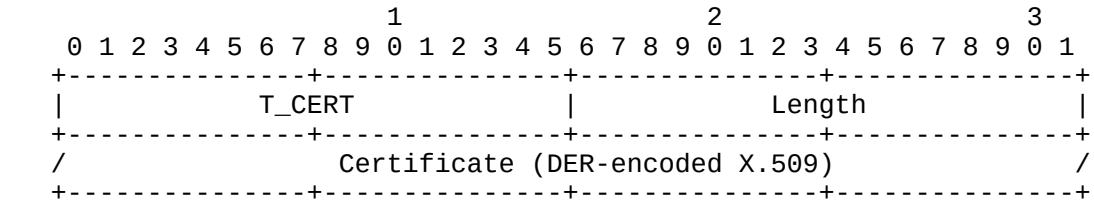

Figure 26: Certificate Encoding

3.6.4.1.4.4. KeyLink

A KeyLink type KeyLocator is a Link.

The KeyLink ContentObjectHashRestr, if included, is the digest of the Content Object identified by KeyLink, not the digest of the public key. Likewise, the KeyIdRestr of the KeyLink is the KeyId of the ContentObject, not necessarily of the wrapped key.

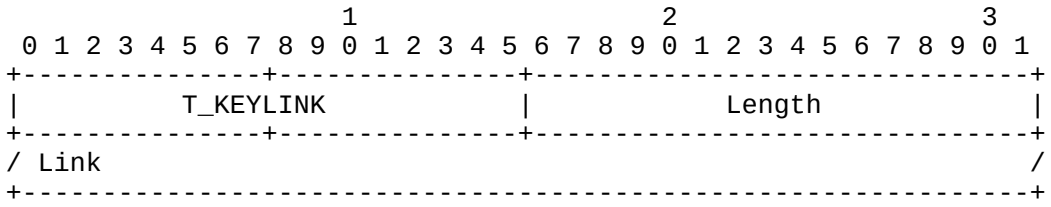

Figure 27: KeyLink Encoding

3.6.4.1.4.5. SignatureTime

The SignatureTime is a millisecond timestamp indicating the time at which a signature was created. The signer sets this field to the current time when creating a signature. A verifier may use this time to determine whether or not the signature was created during the validity period of a key, or if it occurred in a reasonable sequence with other associated signatures. The SignatureTime is unrelated to any time associated with the actual CCNx Message, which could have been created long before the signature. The default behavior is to always include a SignatureTime when creating an authenticated message (e.g., HMAC or RSA).

SignatureTime is an unsigned integer in network byte order that indicates when the signature was created (as the number of milliseconds since the epoch in UTC). It is a fixed 64-bit field.

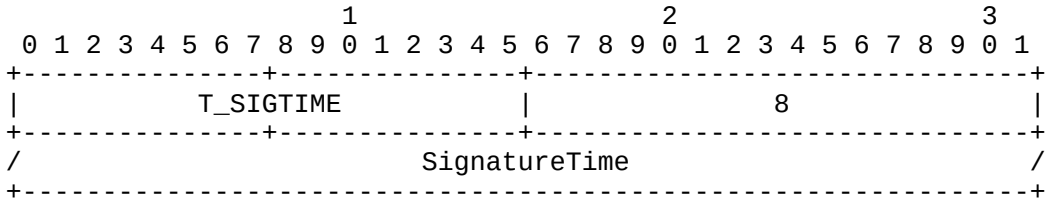

Figure 28: SignatureTime Encoding

3.6.4.1.5. Validation Examples

As an example of a MIC-type validation, the encoding for CRC32C validation would be:

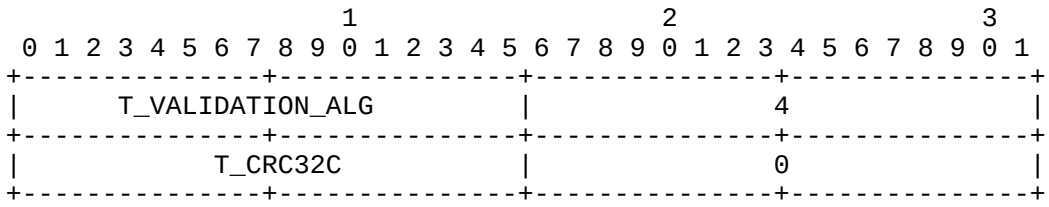

Figure 29: CRC32C Encoding Example

As an example of a MAC-type validation, the encoding for an HMAC using a SHA256 hash would be:

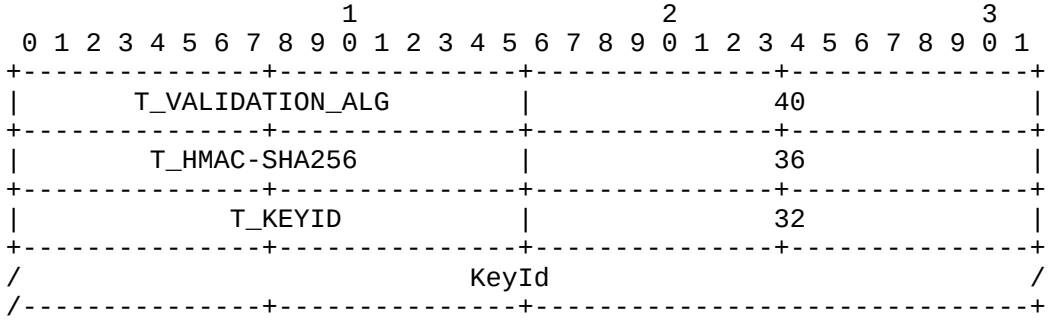

Figure 30: HMAC-SHA256 Encoding Example

As an example of a Signature-type validation, the encoding for an RSA public-key signature using a SHA256 digest and Public Key would be:

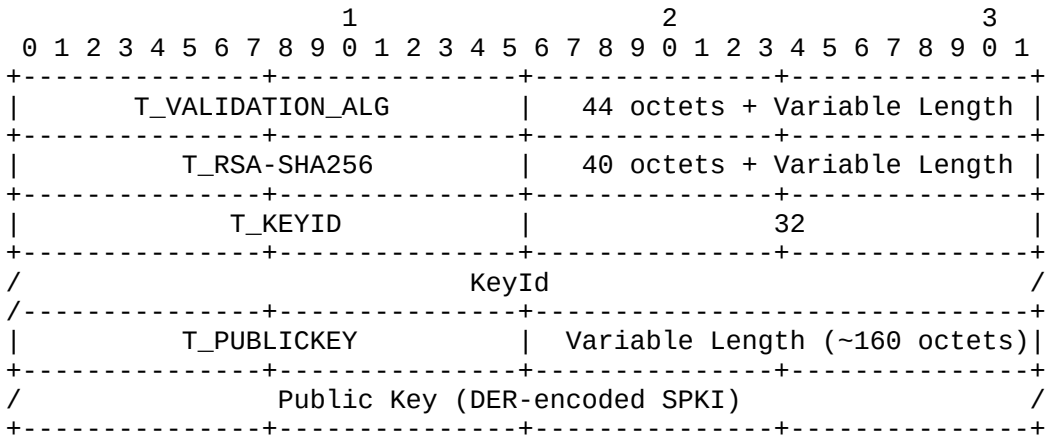

Figure 31: RSA-SHA256 Encoding Example

3.6.4.2. Validation Payload

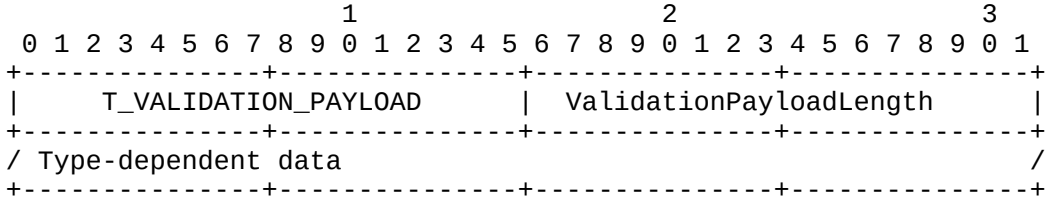

Figure 32: Validation Payload Encoding

The ValidationPayload contains the validation output, such as the CRC32C code or the RSA signature.

4. IANA Considerations

This section details each kind of CCNx protocol value that can be registered. Each type registry can be updated by incrementally expanding the type space, i.e., by allocating and reserving new types. As per [RFC8126], this section details the creation of the "Content-Centric Networking (CCNx)" registry and several subregistries.

4.1. Packet Type Registry

IANA has created the "CCNx Packet Types" registry and allocated the packet types described below. The registration procedure is RFC Required. The Type value is 1 octet. The range is 0x00-0xFF.

Type	Name	Reference
0x00	PT_INTEREST	Fixed Header Types (Section 3.2)
0x01	PT_CONTENT	Fixed Header Types (Section 3.2)
0x02	PT_RETURN	Fixed Header Types (Section 3.2)

Packet Types

4.2. Interest Return Code Registry

IANA has created the "CCNx Interest Return Code Types" registry and allocated the Interest Return code types described below. The registration procedure is Specification Required. The Type value is 1 octet. The range is 0x00-0xFF.

Type	Name	Reference
0x00	Reserved	
0x01	T_RETURN_NO_ROUTE	Fixed Header Types (Section 3.2.3.3)
0x02	T_RETURN_LIMIT_EXCEEDED	Fixed Header Types (Section 3.2.3.3)
0x03	T_RETURN_NO_RESOURCES	Fixed Header Types (Section 3.2.3.3)
0x04	T_RETURN_PATH_ERROR	Fixed Header Types (Section 3.2.3.3)
0x05	T_RETURN_PROHIBITED	Fixed Header Types (Section 3.2.3.3)
0x06	T_RETURN_CONGESTED	Fixed Header Types (Section 3.2.3.3)
0x07	T_RETURN_MTU_TOO_LARGE	Fixed Header Types (Section 3.2.3.3)
0x08	T_RETURN_UNSUPPORTED_HASH_RESTRICTION	Fixed Header Types (Section 3.2.3.3)
0x09	T_RETURN_MALFORMED_INTEREST	Fixed Header Types (Section 3.2.3.3)

CCNx Interest Return Types

4.3. Hop-by-Hop Type Registry

IANA has created the "CCNx Hop-by-Hop Types" registry and allocated the hop-by-hop types described below. The registration procedure is RFC Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0001	T_INTLIFE	Hop-by-hop TLV headers (Section 3.4)
0x0002	T_CACHETIME	Hop-by-hop TLV headers (Section 3.4)
0x0003	T_MSGHASH	Hop-by-hop TLV headers (Section 3.4)
0x0004 - 0x0007	Reserved	
0x0FFE	T_PAD	Pad (Section 3.3.1)
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000-0x1FFF	Reserved	Experimental Use (Section 3)

CCNx Hop-by-Hop Types

4.4. Top-Level Type Registry

IANA has created the "CCNx Top-Level Types" registry and allocated the top-level types described below. The registration procedure is RFC Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0001	T_INTEREST	Top-Level Types (Section 3.5)
0x0002	T_OBJECT	Top-Level Types (Section 3.5)
0x0003	T_VALIDATION_ALG	Top-Level Types (Section 3.5)
0x0004	T_VALIDATION_PAYLOAD	Top-Level Types (Section 3.5)

CCNx Top-Level Types

4.5. Name Segment Type Registry

IANA has created the "CCNx Name Segment Types" registry and allocated the name segment types described below. The registration procedure is Specification Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0001	T_NAMESEGMENT	Name (Section 3.6.1)
0x0002	T_IPID	Name (Section 3.6.1)
0x0010 - 0x0013	Reserved	RFC 8609
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000 - 0x1FFF	T_APP:00 - T_APP:4096	Application Components (Section 3.6.1)

CCNx Name Segment Types

4.6. Message Type Registry

IANA has created the "CCNx Message Types" registry and registered the message segment types described below. The registration procedure is RFC Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	T_NAME	Message Types (Section 3.6)
0x0001	T_PAYLOAD	Message Types (Section 3.6)
0x0002	T_KEYIDRESTR	Message Types (Section 3.6)
0x0003	T_OBJHASHRESTR	Message Types (Section 3.6)
0x0005	T_PAYLDTYPE	Content Object Message Types (Section 3.6.2.2)
0x0006	T_EXPIRY	Content Object Message Types (Section 3.6.2.2)
0x0007 - 0x000C	Reserved	RFC 8609
0x0FFE	T_PAD	Pad (Section 3.3.1)
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000-0x1FFF	Reserved	Experimental Use (Section 3)

CCNx Message Types

4.7. Payload Type Registry

IANA has created the "CCNx Payload Types" registry and allocated the payload types described below. The registration procedure is Specification Required. The Type value is 1 octet. The range is 0x00-0xFF.

Type	Name	Reference
0x00	T_PAYLOADTYPE_DATA	Payload Types (Section 3.6.2.2.1)
0x01	T_PAYLOADTYPE_KEY	Payload Types (Section 3.6.2.2.1)
0x02	T_PAYLOADTYPE_LINK	Payload Types (Section 3.6.2.2.1)

CCNx Payload Types

4.8. Validation Algorithm Type Registry

IANA has created the "CCNx Validation Algorithm Types" registry and allocated the validation algorithm types described below. The registration procedure is Specification Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0002	T_CRC32C	Validation Algorithm (Section 3.6.4.1)
0x0004	T_HMAC-SHA256	Validation Algorithm (Section 3.6.4.1)
0x0005	T_RSA-SHA256	Validation Algorithm (Section 3.6.4.1)
0x0006	T_EC-SECP-256K1	Validation Algorithm (Section 3.6.4.1)
0x0007	T_EC-SECP-384R1	Validation Algorithm (Section 3.6.4.1)
0x0FFE	T_PAD	Pad (Section 3.3.1)
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000-0x1FFF	Reserved	Experimental Use (Section 3)

CCNx Validation Algorithm Types

4.9. Validation-Dependent Data Type Registry

IANA has created the "CCNx Validation-Dependent Data Types" registry and allocated the validation-dependent data types described below. The registration procedure is RFC Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0009	T_KEYID	Validation-Dependent Data (Section 3.6.4.1.4)
0x000A	T_PUBLICKEYLOC	Validation-Dependent Data (Section 3.6.4.1.4)
0x000B	T_PUBLICKEY	Validation-Dependent Data (Section 3.6.4.1.4)
0x000C	T_CERT	Validation-Dependent Data (Section 3.6.4.1.4)
0x000D	T_LINK	Validation-Dependent Data (Section 3.6.4.1.4)
0x000E	T_KEYLINK	Validation-Dependent Data (Section 3.6.4.1.4)
0x000F	T_SIGTIME	Validation-Dependent Data (Section 3.6.4.1.4)
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000-0x1FFF	Reserved	Experimental Use (Section 3)

CCNx Validation-Dependent Data Types

4.10. Hash Function Type Registry

IANA has created the "CCNx Hash Function Types" registry and allocated the hash function types described below. The registration procedure is Specification Required. The Type value is 2 octets. The range is 0x0000-0xFFFF.

Type	Name	Reference
0x0000	Reserved	
0x0001	T_SHA-256	Hash Format (Section 3.3.3)
0x0002	T_SHA-512	Hash Format (Section 3.3.3)
0x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
0x1000-0x1FFF	Reserved	Experimental Use (Section 3)

CCNx Hash Function Types

5. Security Considerations

The CCNx protocol is a Layer 3 network protocol, which may also operate as an overlay using other transports such as UDP or other tunnels. It includes intrinsic support for message authentication via a signature (e.g., RSA or elliptic curve) or Message Authentication Code (e.g., HMAC). In lieu of an authenticator, it may instead use a Message Integrity Check (e.g., SHA or CRC). CCNx does not specify an encryption envelope; that function is left to a high-layer protocol (e.g., Encrypted Sessions in CCNx [esic]).

The CCNx Packet format includes the ability to attach MICs (e.g., SHA-256 or CRC), MACs (e.g., HMAC), and Signatures (e.g., RSA or ECDSA) to all packet types. Because Interest packets can be sent at will, an application should carefully select when to use a given ValidationAlgorithm in an Interest to avoid DoS attacks. MICs, for example, are inexpensive and could be used as desired, whereas MACs and Signatures are more expensive and their inappropriate use could open a computational DoS attack surface. Applications should use an explicit protocol to guide their use of packet signatures. As a general guideline, an application might use a MIC on an Interest to detect unintentionally corrupted packets. If one wishes to secure an Interest, one should consider using an encrypted wrapper and a protocol that prevents replay attacks, especially if the Interest is being used as an actuator. Simply using an authentication code or signature does not make an Interest secure. There are several examples in the literature on how to secure ICN-style messaging [mobile] [ace].

As a Layer 3 protocol, this document does not describe how one arrives at keys or how one trusts keys. The CCNx content object may include a public key embedded in the object or may use the PublicKeyLocator field to point to a public key (or public-key certificate) that authenticates the message. One key exchange specification is CCNxKE [ccnxke] [mobile], which is similar to the TLS 1.3 key exchange except it is over the CCNx Layer 3 messages. Trust is beyond the scope of a Layer 3 protocol and is left to applications or application frameworks.

The combination of an ephemeral key exchange (e.g., CCNxKE [ccnxke]) and an encapsulating encryption (e.g., [esic]) provides the equivalent of a TLS tunnel. Intermediate nodes may forward the Interests and Content Objects but have no visibility inside. It also completely hides the internal names in those used by the encryption layer. This type of tunneling encryption is useful for content that has little or no cacheability, as it can only be used by someone with the ephemeral key. Short-term caching may help with lossy links or mobility, but long-term caching is usually not of interest.

Broadcast encryption or proxy re-encryption may be useful for content with multiple uses over time or many consumers. There is currently no recommendation for this form of encryption.

The specific encoding of messages will have security implications. This document uses a Type-Length-Value (TLV) encoding. We chose to compromise between extensibility and unambiguous encodings of types and lengths. Some TLV encodings use variable-length T and variable-length L fields to accommodate a wide gamut of values while trying to be byte efficient. Our TLV encoding uses a fixed length 2-byte T and 2-byte L. Using fixed-length T and L fields solves two problems. The first is aliases. If one is able to encode the same value, such as 0x02 and 0x0002, in different byte lengths, then one must decide if they mean the same thing, if they are different, or if one is illegal. If they are different, then one must always compare on the buffers not the integer equivalents. If one is illegal, then one must validate the TLV encoding -- every field of every packet at every hop. If they are the same, then one has the second problem: how to specify packet filters. For example, if a name has 6 name components, then there are 7 T fields and 7 L fields, each of which might have up to 4 representations of the same value. That would be 14 fields with 4 encodings each, or 1001 combinations. It also means that one cannot compare, for example, a name via a memory function, as one needs to consider that any embedded T or L might have a different format.

The Interest Return message has no authenticator from the previous hop. Therefore, the payload of the Interest Return should only be used locally to match an Interest. A node should never forward that Interest payload as an Interest. It should also verify that it sent the Interest in the Interest Return to that node and not allow anyone to negate Interest messages.

Caching nodes must take caution when processing content objects. It is essential that the Content Store obey the rules outlined in [RFC8569] to avoid certain types of attacks. CCNx 1.0 has no mechanism to work around an undesired result from the network (there are no "excludes"), so if a cache becomes poisoned with bad content it might cause problems retrieving content. There are three types of access to content from a Content Store: unrestricted, signature restricted, and hash restricted. If an Interest has no restrictions, then the requester is not particular about what they get back, so any matching cached object is OK. In the hash restricted case, the requester is very specific about what they want, and the Content Store (and every forward hop) can easily verify that the content matches the request. In the signature restricted case (which is often used for initial manifest discovery), the requester only knows the KeyId that signed the content. This case requires the closest attention in the Content Store to avoid amplifying bad data. The Content Store must only respond with a content object if it can verify the signature -- this means either the content object carries the public key inside it or the Interest carries the public key in addition to the KeyId. If that is not the case, then the Content Store should treat the Interest as a cache miss and let an endpoint respond.

A user-level cache could perform full signature verification by fetching a public key according to the PublicKeyLocator. However, that is not a burden we wish to impose on the forwarder. A user-level cache could also rely on out-of-band attestation, such as the cache operator only inserting content that it knows has the correct signature.

The CCNx grammar allows for hash algorithm agility via the HashType. It specifies a short list of acceptable hash algorithms that should be implemented at each forwarder. Some hash values only apply to end systems, so updating the hash algorithm does not affect forwarders -- they would simply match the buffer that includes the type-length-hash buffer. Some fields, such as the ConObjHash, must be verified at each hop, so a forwarder (or related system) must know the hash algorithm, and it could cause backward compatibility problems if the hash type is updated.

A CCNx name uses binary matching, whereas a URI uses a case-insensitive hostname. Some systems may also use case-insensitive matching of the URI path to a resource. An implication of this is that human-entered CCNx names will likely have case or non-ASCII symbol mismatches unless one uses a consistent URI normalization for the CCNx name. It also means that an entity that registers a CCNx-routable prefix -- say, "ccnx:/example.com" -- would need separate registrations for simple variations like "ccnx:/Example.com". Unless this is addressed in URI normalization and routing protocol conventions, there could be phishing attacks.

For a more general introduction to ICN-related security concerns and approaches, see [RFC7927] and [RFC7945].

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

6.2. Informative References

[ace] Shang, W., Yu, Y., Liang, T., Zhang, B., and L. Zhang, "NDN-ACE: Access control for constrained environments over named data networking", NDN Technical Report NDN-0036, 2015, <http://new.named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf>.

[ccnxke] Mosko, M., Uzun, E., and C. Wood, "CCNx Key Exchange Protocol Version 1.0", Work in Progress, draft-wood-icnrg-ccnxkeyexchange-02, March 2017.

[CCNxURI] Mosko, M. and C. Wood, "The CCNx URI Scheme", Work in Progress, draft-mosko-icnrg-ccnxurischeme-01, April 2016.

[CCNxz] Mosko, M., "CCNxz TLV Header Compression Experimental Code", commit f1093a2, March 2018, <https://github.com/PARC/CCNxz>.

[compress] Mosko, M., "Header Compression for TLV-based Packets", ICNRG Interim Meeting, 2016, <https://datatracker.ietf.org/meeting/interim-2016-icnrg-02/materials/slides-interim-2016-icnrg-2-7>.

[ECC] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", 2010, <http://www.secg.org/sec2-v2.pdf>.

[esic] Mosko, M. and C. Wood, "Encrypted Sessions In CCNx (ESIC)", Work in Progress, draft-wood-icnrg-esic-01, September 2017.

[IANA-PEN] IANA, "Private Enterprise Numbers", <http://www.iana.org/assignments/enterprise-numbers>.

[mobile] Mosko, M., Uzun, E., and C. Wood, "Mobile Sessions in Content-Centric Networks", IFIP Networking, 2017, <http://dl.ifip.org/db/conf/networking/networking2017/1570334964.pdf>.

[nnc] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT '09), 2009, <http://dx.doi.org/10.1145/1658939.1658941>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7927] Kutscher, D., Ed., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", RFC 7927, DOI 10.17487/RFC7927, July 2016, <https://www.rfc-editor.org/info/rfc7927>.

[RFC7945] Pentikousis, K., Ed., Ohlman, B., Davies, E., Spirou, S., and G. Boggia, "Information-Centric Networking: Evaluation and Security Considerations", RFC 7945, DOI 10.17487/RFC7945, September 2016, <https://www.rfc-editor.org/info/rfc7945>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8569] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Semantics", RFC 8569, DOI 10.17487/RFC8569, July 2019, <https://www.rfc-editor.org/info/rfc8569>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
United States of America

Phone: +01 650-812-4405
Email: mmosko@parc.com

Ignacio Solis
LinkedIn
Mountain View, California 94043
United States of America

Email: nsolis@linkedin.com

Christopher A. Wood
University of California, Irvine
Irvine, California 92697
United States of America

Phone: +01 315-806-5939
Email: woodc1@uci.edu