CCM Security Mode Specification

DOF Protocol Specification

Technical Steering Committee

CCM Security Mode Specification: DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-12
Document Version	7.0.1, 12 January 2018
Build Version	7.0.1.8
Source Information	2018-01-12, "." at commit c4a21c8bf741a32d4fe20776861d03baab74639a on branch "master" of repository core-specifications
External Reference	CCM Security Mode Specification, OpenDOF TSC, [dof-2008-12] (7.0.1, 12 January 2018)

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc.. Contact the committee using the following information.

Contact Information

Post Mail	OpenDOF Project, Inc. Technical Steering Committee 3855 SW 153rd Drive Beaverton, OR 97003
Telephone	1-503-619-4114
Fax	1-503-644-6708
Email	tsc@member.opendof.org [mailto:tsc@member.opendof.org]

Copyright

Copyright © 2008-2018 Panasonic Corporation, Bryant Eastham

Patents

Contributors to the OpenDOF Project make a non-enforcement pledge related to any patent required to implement the protocols described in this document. For more information see https://opendof.org/pledge.

License

See https://opendof.org/license-specifications.

Table of Contents

1. About This Document 1
    1.1. Audience 1
    1.2. Authority 1
    1.3. PDU Definitions 1
        1.3.1. Context 2
        1.3.2. Generalization 2
        1.3.3. Qualifications 2
        1.3.4. Fields 4
    1.4. Specifications, Notes and Warnings 5
    1.5. Printing This Document 6
    1.6. Comparing Documents 6
        1.6.1. Text Comparison 6
        1.6.2. Graphical Comparison 6
2. Specification 9
    2.1. Application Format 9
    2.2. Security Primitives 9
    2.3. Negotiation 9
    2.4. Initialization and Synchronization 11
    2.5. Security Header 14
    2.6. Node Identifier Compression 17
    2.7. Node Identifier Caching 17
    2.8. CCM Header Fields 18
    2.9. Nonce Formatting 18
    2.10. Node Identifier Assignment and Query 19
        2.10.1. Node Identifier Range 20
        2.10.2. New Manager Logic 21
        2.10.3. Takeover Logic 21
        2.10.4. Modifying NNI 21
        2.10.5. Query 21
        2.10.6. Request 22
    2.11. Security State 23
        2.11.1. Mode Flags and Mode of Operation 24
        2.11.2. Session Key 24
    2.12. Nonce 25
    2.13. Datagram Number 28
3. External PDU Reference 31
    3.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 31
    3.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 32
    3.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018) 33

Specifications and PDUs

CCM Application PDU Format `dof-2008-12-pdu-1` 9
CCM Mode of Operation Attribute `dof-2008-12-pdu-2` 10
The Tmin and Tmax fields must not be zero. `dof-2008-12-spec-1` 11
Encrypt information if requested, otherwise terminate session. `dof-2008-12-spec-2` 11
Mode Initialization (Lossless Transport) `dof-2008-12-pdu-3` 11
Mode Initialization (Lossy Transport) `dof-2008-12-pdu-4` 12
Mode initialization on lossy transports must contain required fields. `dof-2008-12-spec-3` 12
CCM Initialization Flags Format `dof-2008-12-pdu-5` 13
The NODE IDENTIFIER and TARGET NODE IDENTIFIER fields must be different.
`dof-2008-12-spec-4` 14
The TARGET NODE IDENTIFIER must not be specified in transport multicast packets.
`dof-2008-12-spec-5` 14
CCM Flags Format `dof-2008-12-pdu-6` 14
The NODE IDENTIFIER field must only be used in multipoint sessions, where it is required.
`dof-2008-12-spec-6` 16
The **Slot** field must only be used in multipoint sessions, where it is required. `dof-2008-12-spec-7` 16
The **Datagram Number** must be used lossy and may be used lossless. `dof-2008-12-spec-8` 16
The **Target Node Identifier** must only be used in multipoint sessions in order for the datagram to be treated as unicast. `dof-2008-12-spec-9` 17
The **Next Node Identifier** must only be used by the manager in transport multicast sessions.
`dof-2008-12-spec-10` 17
Correctly cache and clear the cache of target node identifiers. `dof-2008-12-spec-11` 18
CCM Mode Nonce Format `dof-2008-12-pdu-7` 19
Monitor inbound PDUs for node identifier collisions. `dof-2008-12-spec-12` 20
Query (command) `dof-2008-12-pdu-8` 21
Request (command) `dof-2008-12-pdu-9` 22
Request (response) `dof-2008-12-pdu-10` 23
Manage security state as described. `dof-2008-12-spec-13` 24
CCM nonces must only be accepted once, and must have increasing values of slot and datagram number. `dof-2008-12-spec-14` 25
Manage CCM slots as required, including delays for newly distributed keys. `dof-2008-12-spec-15` 26
For any old period, CCM slot state must be invalidated within two state synchronization periods.
`dof-2008-12-spec-16` 27
CCM period state must be invalidated within two synchronization periods of the establishment of a new period. `dof-2008-12-spec-17` 28
Drop all datagrams that include a target node identifier not equal to the receivers. `dof-2008-12-spec-18` 28
CCM datagram number must only be accepted once, and must have increasing values.
`dof-2008-12-spec-19` 28
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 31
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 31
Security Mode of Operation Header `dof-2008-1-pdu-9` 32
General Application Header `dof-2008-1-pdu-13` 32
Attribute/Value Pair `dof-2008-1-pdu-16` 32
Security Mode of Operation Initialization `dof-2008-16-pdu-12` 33

1. About This Document

This document presents technical specifications for using Counter with CBC-MAC (CCM) with DOF implementations.

The *DOF Security Specification* presents general requirements that apply to all security modes. Common PDUs related to security modes are covered in the *DOF Protocol Specification.*

This information is highly technical. This document serves as a specification and reference against which someone could implement networked software that interoperated with DOF products without using existing libraries.

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc. (ODP-TSC).

1.1. Audience

The primary audience of this document is people implementing libraries that use DOF protocols. Others that are concerned about how DOF protocols work on a network at a very low level can also benefit from this document.

Readers should be familiar with technical protocol documentation. This document is similar to an 'RFC' for DOF protocols, and familiarity with the language used in these types of documents is helpful.

This document is *not* required reading for those who need to use existing DOF libraries, although system designers may benefit from an understanding of this information. In particular, the information related to security and connections can help system designers to better understand how DOF systems work.

1.2. Authority

This document, in its English form, is the authoritative document for DOF protocols. Due to the detailed nature of protocol design, it may be difficult for a reader to implement these protocols correctly based on a translated document. When questions arise, the English version is authoritative.

All implementations that claim to be DOF compliant must satisfy the requirements set forth in this document.

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc., referred to as 'ODP-TSC'. The inside cover of this document contains contact information for the Technical Steering Committee.

The web site for the Technical Steering Committee is https://opendof.org/tsc.

1.3. PDU Definitions

This document deals with the transmission of information on a network. In order to describe this, diagrams show different fields, their network order, and positioning.

All DOF protocol documentation uses a similar diagramming style, discussed in this section. The word 'PDU', which stands for 'Protocol Data Unit', indicates these diagrams.

1.3.1. Context

Each PDU represents information sent on the network in some context. This usually means that there is information sent 'surrounding' the PDU itself. This information provides the 'context' for the PDU.

For example, the context for the DOF Protocol Stack itself is a network transport. This could be UDP/IP, TCP/IP, or any other transport. The transport will require certain headers and trailers, but they are not shown in each DOF Protocol Stack PDU. In this example the transport provide the context of the DOF Protocol Stack PDU.

In a similar way, the DOF Protocol Stack defines the context for the other DOF protocols. The Object Access Protocol, for example, can use the DOF Protocol Stack for its context. Just as the transport headers are not shown for the DOF Protocol Stack, the DOF Protocol Stack headers are not shown for the Object Access Protocol.

Understanding the context requires understanding the relative position of the different layers in the protocol stack. Each PDU defines the ordering of all of its parts, and may indicate that it is itself an instance of some other general PDU format. This means that it is always possible to start at a high-level PDU definition and understand it completely, but not always possible to identify everywhere that a particular PDU may be used (as a part of some other higher-level PDU).

Each protocol specification defines its specific contexts. The context definition includes a description of the underlying stack layer (or the containing layer), and defines the information that must be available from that layer (for reception) or passed to that layer (for transmission).

As an example, the context of the DOF Network Protocol is a transport. The transport has a requirement to provide the addresses of nodes, and so part of the context for the transport is the sender address (for reception) or the target address (for transmission). Any transport that will work with the DOF Protocol Stack must meet the context requirements.

1.3.2. Generalization

Many times a particular PDU will be a specific type of a more general PDU. In these cases, the PDU will indicate that it is an 'instance' of the more general type.

1.3.3. Qualifications

Each PDU will have qualifications placed on its use. The general qualifications include security and transport requirements. The following terms identify these qualifications. Others may be present as well.

1.3.3.1. Transport Qualifications

There are three major transport categories for DOF protocols (see the *DOF Protocol Specification*, Transport Requirements). These are None, Lossy, and Lossless. There are also three addressing types: Unicast, Multicast, and Broadcast. PDUs indicate the ability to use each of these categories as follows:

```
Session: Combinations of None, Lossy (2-node, n-node), or Lossless.
Addressing: Combinations of Unicast, Multicast, or Broadcast.
```

1.3.3.2. Security Qualifications

Each PDU categorizes security qualifications according to the major aspects of security (see the *DOF Security Specification*, Overview). These are Encryption, Data Integrity, Authentication, and Access

Control. Encryption prevents viewing of the PDU, Data Integrity ensures that others do not modify the PDU, Authentication indicates knowledge of the nodes involved in the communication, and Access Control restricts those able to use the PDU.

Data Integrity is required according to the DOF Protocol Stack specification for all secure packets. Authentication is required whenever permissions are involved, or when it is important to know something about the relationship between the sender and receiver. Encryption is required if the contents of the packet contain information that must not be visible to potential attackers.

In addition, there is the case of unsecured communication. PDUs that do not require security will list 'Unsecured' as a possible choice. Finally, certain PDUs are required to be Unsecured, and are identified as such.

Each application PDU must specify its security requirements. If Access Control is required, the PDU will indicate the permissions required for the command and for any response. However, it is typical for all of the PDUs of a given protocol to share the same requirements, and so the protocol specification may define its security qualifications in a single section. In this case, the individual PDUs should refer back to this section to avoid confusion.

It is also typical for applications to require the security of the session. A PDU indicates this as 'session security', and is its own security qualification. If the PDU allows the 'None' session type then this implies that Unsecured is allowed, as session type 'None' cannot be secure.

For PDUs that refer to session security or a common security section, one of the following formats is used:

```
Security: See the section 'Security Qualifications'.
```

or

```
Security: Session.
```

The PDU uses the following identifications to specify the security qualifications for a PDU:

```
Unsecured: (description).
Encryption: (description).
Message authentication: (description).

Permission (command): (description).
Permission (response): (description).
```

For each of these, the (description) varies. There are many different combinations of values, as described here.

Unsecured can have the values 'Required', 'Allowed', 'Not Allowed'.

Encryption can have the values 'Required', 'Allowed', 'Not Allowed'.

Message authentication can have the values 'Required', 'Allowed', 'Not Allowed'.

The specific PDU determines the permissions required (for both command and response).

There are combinations that are not valid. The following table defines all allowed combinations if a combination does not appear in the table then it is invalid.

In general, security theory links Encryption and Data Integrity. If one is 'Required' then the other must at least be 'Allowed'. Encryption and Data Integrity are the opposite of Unsecured. If either is 'Required' then

unsecured must be 'Not Allowed'. Again, note that since data integrity is a common requirement for all secure packets the PDUs do not list it separately.

Unsecured / Encryption / Message Authentication	**Description**
Required / Not Allowed / Not Allowed	Indicates the PDU must be unsecured.
Allowed / Allowed / Allowed	Indicates there are no security requirements.
Not Allowed / Allowed / Allowed	Indicates the PDU must be either encrypted or authenticated. This is the lowest level of security possible for a PDU.
Not Allowed / Required / Allowed	Indicates that encryption is required, and data integrity is optional.
Not Allowed / Allowed / Required	Indicates that encrypted is optional, but authentication is required.
Not Allowed / Required / Required	Indicates both encryption and authentication are required.

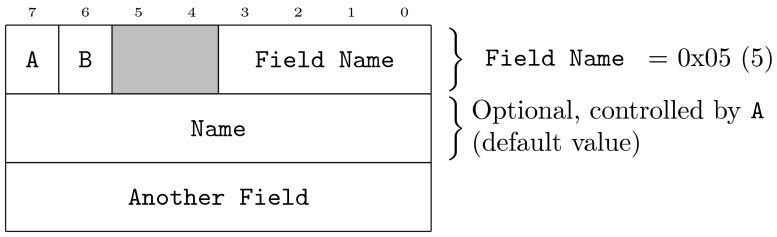
1.3.4. Fields

Each PDU shows a container, represented as a sequence of fields. The table fully describes each field.

The following is an example of a PDU description.

Example PDU`example-pdu-1`

PDU Context



`A`	**One bit.** Controls the presence of `Name`.
`B`	**One bit.** Field description.
`Field Name`	**Type and size description.** Field description.
`Name`	**Type and size description. Optional, controlled by** `A`. Field description.
`Another Field`	**Type and size description.** Field description.

The container represents the PDU. The diagram shows Fields first to last, left to right and top to bottom. This means that the first fields appear on the wire before the last fields. This is a general principle: things on the top and toward the left in the diagrams appear on the wire before things on the right and bottom. Note that the transport is free to use whatever ordering it must, and memory layout may different from the PDU definition. However, the transfer of a PDU from a program, over an arbitrary transport and into another program must maintain the perceived ordering between the two programs. PDU diagrams

are always shown MSB first (most significant byte) and msb first (most significant bit) unless otherwise indicated.

Each field identifies its contents and provides information necessary to understand its use. In the example above, the diagram shows the first six fields of the container. Each field is named, with the exception of reserved fields which are indicated as being grayed out. Field names use `monospace text` similar to that used in program listing. The third field shown is indicated as reserved. Reserved bits have special requirements described in the *DOF Protocol Specification*.

Several different field cases are typical, described in the following sections.

1.3.4.1. Typical Field

A typical field defines its name, its type, and its description. As shown, this field is not optional, and so it must appear in the position indicated by the diagram. The PDU uses `Field Name` throughout the related documentation to refer to the field. The type of the field is a reference to some other defined PDU. Otherwise, the PDU indicates the size of the field and any defined structure.

1.3.4.2. Optional Field

An optional field is similar to the typical case, but is indicated as optional in a note to the right. Each optional field will indicate what controls whether or not it is included as well as the default value that the field takes on when it is absent.

1.3.4.3. Fixed Values

Fields may have required values based on the specific PDU. This is common in the case of a PDU being an instance of another PDU, where the specific instance requires certain field values. In this case, it is not valid for the PDU to contain values other than that specified, if so the PDU is invalid.

The required value is shown to the right of the field as a comment.

1.3.4.4. Bit Fields

It is common for PDUs to leverage a single byte to store multiple fields (bit fields). The packing of multiple fields into a single byte is indicated as shown in the earlier example. This packing of bits may also leave space that can be indicated as reserved.

The PDU indicates reserved bits visually although the PDU does not individual name these fields.

Note that bit fields may specify default values just as other fields. However, single bit fields do not show their hexadecimal representation. In addition, bits without names that have required values may indicate their values in the place of the name.

1.4. Specifications, Notes and Warnings

This is a technical specification document. It is critical to understand how the document indicates specification items. There are also annotations for notes (implementation notes or other information that is worthy of extra notice) and warnings (pitfalls or extremely critical information). In addition, there are implementation notes, which point out optimizations that are not obvious.

Each type of callout ends either when another callout begins or when the text goes back to the normal indentation.



This is a note. The text is inset and there is a header to indicate the beginning of the text.



This is an implementation note. The text is inset and there is a header to indicate the beginning of the text.



This is a warning. The text is bold inset and there is a different header to indicate the beginning of the text.



This is an example`example-spec-1`

This is the text.

Following the summary is descriptive information about the specification item. Each specification item has a unique number assigned.

1.5. Printing This Document

The format of this document is for two-sided printing on US Letter paper.

1.6. Comparing Documents

The easiest way to compare versions of this document is to compare the corresponding source. The source format is AsciiDoctor, which is a markup-style text format. This format can be compared using any text comparison tool, including those that work with source control documents. To facilitate determining the source for each document there is information about the commit, branch, and repository along with the document version information. In addition the following guidelines are followed to make things easy to compare:

1. The input files put each sentence on a separate line, with no hard line-wrapping.

2. To the extent possible, macros and templates are used to determine formatting and structure.

3. Graphics and diagrams also use text formats.

If all you have available is a compiled document (PDF or HTML) then there are tools available to compare them. This section summarizes some of these methods with their pros and cons.

1.6.1. Text Comparison

There are several free tools that will do comparisons of the text of a PDF. One of these is the xdocdiff plugin for WinMerge. The plugin extracts the text from the PDF (including some information like page numbers), and WinMerge then compares the text.

WinMerge is available from http://winmerge.org. The xdocdiff plugin is available from http://freemind.s57.xrea.com/xdocdiffPlugin/en/index.html.

1.6.2. Graphical Comparison

In order to compare formatting, or to see the changes in the context of the document, a graphical compare is required. There are several free tools, but each has limitations.

1. diff-pdf, available from https://github.com/vslavik/diff-pdf, shows an overlay of the matching pages in different colors.

2. DiffPDF, available fromhttp://www.qtrac.eu/diffpdf.html, shows side-by-side red-lined differences of matching pages.

Both of these tools suffer from their page-by-page comparison methods. This means that when context shifts because of additions or removals that they start sensing changes at each page boundary. DiffPDF has the ability to select comparison ranges, allowing the program to synchronize at the beginning of each section, but this requires manual configuration and can be difficult for large documents.

There are also commercial tools that do an excellent job at comparing PDF documents. The first is Adobe Acrobat (http://www.adobe.com/products/acrobat.html), which in later releases has a 'Compare Document' feature. The result of the comparison is an annotated PDF, and hovering the mouse over the change will show the details.

Another commercial tool is PDF Content Comparer (http://www.inetsoftware.de/products/pdf-content-comparer) from i-net Software. This tool does side-by-side comparison, but has an intuitive auto-synchronize that keeps similar content lined up even when additions or deletions have occurred.

2. Specification

This mode of operation uses CCM (as defined in NIST Special Publication 800-38C [http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf]). The specifics of mode implementation are in this section, although it relies heavily on an understanding of the CCM mode as defined in the NIST reference.

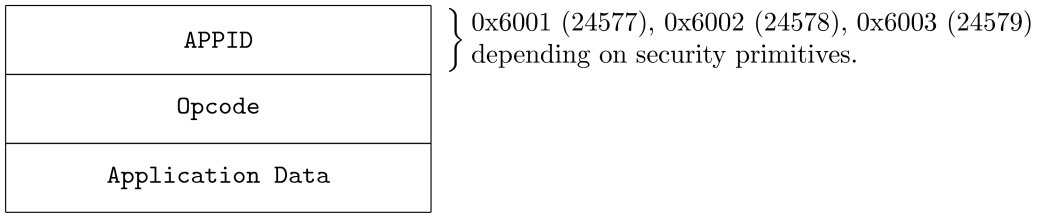
CCM is an DOF security mode of operation, and appears in many different contexts. Most of these references are part of the security specification found in the *DOF Security Specification*.

2.1. Application Format

The ODP-TSC has reserved APPIDs 0x6001 (24577), 0x6002 (24578), and 0x6003 (24579) for CCM. When used to send application PDUs, the following format is used.

⇄ **CCM Application PDU Format** `dof-2008-12-pdu-1`

Instance of General Application Header`dof-2008-1-pdu-13`



`APPID` **Two bytes, MSB.** The indicated value identifies the request as CCM.

`Opcode` **One byte.** The value will indicate the particular operation.

`Application Data` **Variable length. Optional, controlled by** `Opcode`. Data for the command depending on the `Opcode`.

2.2. Security Primitives

This protocol uses security primitives defined by the particular protocol number, as defined here.

APPID	ENC	MODE
0x6001 (24577)	AES	CCM
0x6002 (24578)	Twofish	CCM
0x6003 (24579)	SMS4	CCM

The encryption strength is negotiated, as is the MAC length.

2.3. Negotiation

CCM is a peer-to-peer protocol, meaning that only one node in a session needs to request it and all other nodes may use it. Specifically, in DSP negotiation either the transport client or server may negotiate CCM.

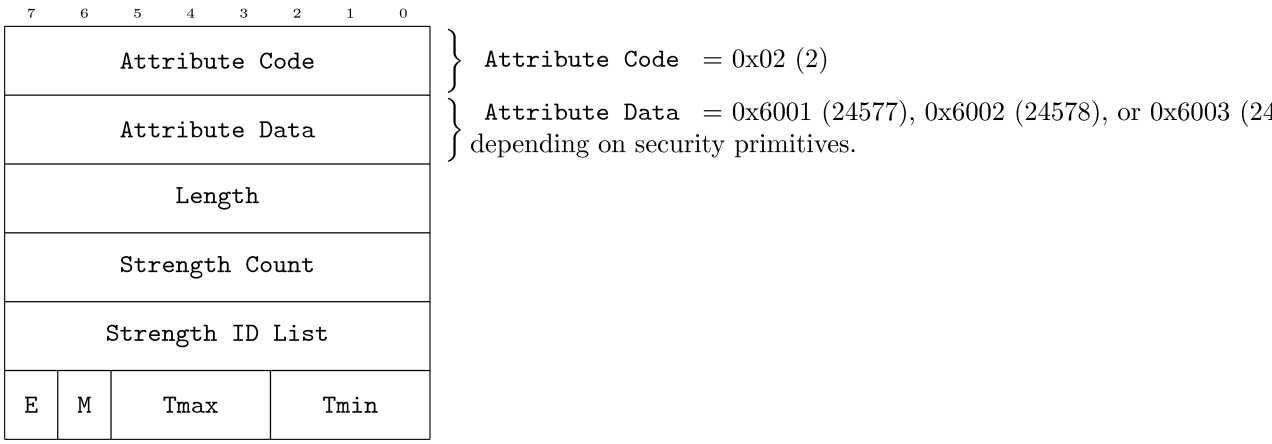
When requested using DSP, the request identifies the security strength, the use of encryption or only of authentication, and the MAC length. There are choices allowed for each of these. This allows a negative acknowledge to pass back a range of acceptable values, which helps to prevent multiple negotiation exchanges. However, the use of the ranges is not required, and a node may respond with multiple DSP attribute blocks to represent different choices.

The client may negotiate the encryption mode of operation (including CCM), but it is not required. It may be beneficial to negotiate, because attempts to use a non-acceptable mode of operation later may result in session termination. If the client does not negotiate a mode of operation then the server may negotiate one.

If an attribute with ranges is exchanged, then the nodes will use the highest priority strength (the first in the list), and the minimum encryption and MAC size that is allowed.

CCM Mode of Operation Attribute `dof-2008-12-pdu-2`

Instance of Attribute/Value Pair`dof-2008-1-pdu-16`



`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

`Attribute Data` **Two bytes, MSB.** This field completes the definition of an attribute in combination with the code.

`Length` **One byte.** This will be two plus `Strength Count`.

`Strength Count` **One byte.** Controls the `Strength ID List`. This value must be greater than zero.

`Strength ID List` **One byte.** This field indicates the allowed security strengths, in decreasing priority. See the *DOF Security Specification* for valid strengths.

`E` **One bit.** This field represents the minimum encryption requirement for the mode. When set it indicates that the mode must encrypt the data in addition to authenticating it. When clear is indicates that the mode may only authenticate the data and not encrypt it.

`M`	**One bit.** This field represents the maximum encryption requirement for the mode. When set it indicates that the mode may encrypt the data in addition to authenticating it. When clear is indicates that the mode must only authenticate the data and not encrypt it.
`Tmax`	**Three bits.** This field controls the maximum MAC length. This is defined in the CCM standard of CTR_0 and is passed as described there (as (t-2)/2). For example, a 16-byte MAC would use the value seven (7) ((16-2)/2).
`Tmin`	**Three bits.** This field controls the minimum MAC length. This is defined in the CCM standard of CTR_0 and is passed as described there (as (t-2)/2). For example, a 16-byte MAC would use the value seven (7) ((16-2)/2).

The Tmin and Tmax fields must not be zero. `dof-2008-12-spec-1`

Based on the CCM standard a T value of zero has no meaning.

DOF security allows for authenticated and unencrypted data. CCM supports authenticated-only data by using “A Data” as described in the CCM specification. Encrypted data is “P Data.” The DOF Presentation Protocol describes which data must be authenticated and which data must be encrypted (if requested).

Encrypt information if requested, otherwise terminate session. `dof-2008-12-spec-2`

Requests for encryption must be honored. If encryption is not possible then the session must be terminated.

2.4. Initialization and Synchronization

CCM requires that the mode be negotiated and that the initial state be determined before encrypted or authenticated datagrams can be exchanged. Once the state is initialized then encrypted datagrams can be accepted and validated. Further, it may be necessary to keep nodes synchronized with the state of the manager (discussed later), and so a method of distributing the current state information is required.

When used during state initialization on lossless sessions the following format is used. The lossless nature of the session means that synchronization of state can assume synchronized session state, and so less information needs to be passed with each datagram.

Mode Initialization (Lossless Transport) `dof-2008-12-pdu-3`

Instance of Security Mode of Operation Initialization`dof-2008-16-pdu-12`

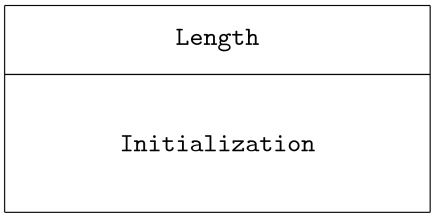


`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.There is no initialization on lossless transports, and the indicated value specifies that no data is present. This means that no initialization data is passed on the wire, and all state is initialized to zero, except for the **Node Identifier**. For lossless transport sessions the **Node Identifier** is initialized based on the role of the node in the security session: one (1) for server nodes and zero (0) for client nodes. Note that in this case the **Manager** bit is assumed based on the transport usage and is ignored in the header.

When used during state initialization on lossy transports the following format is used.

Mode Initialization (Lossy Transport) `dof-2008-12-pdu-4`

Instance of Security Mode of Operation Initialization`dof-2008-16-pdu-12`



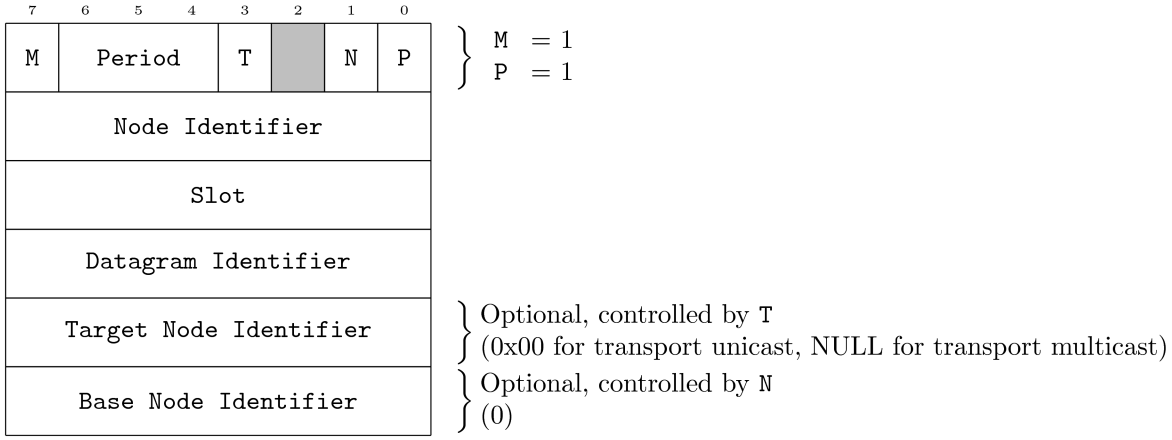
`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The number of bytes in `Initialization`.

`Initialization` **Instance of** CCM Initialization Flags Format`dof-2008-12-pdu-5`.The initialization fields, described below.

Mode initialization on lossy transports must contain required fields. `dof-2008-12-spec-3`

The specification of the referenced PDU specifies the requirements.

CCM Initialization Flags Format `dof-2008-12-pdu-5`



`M` **One bit.** Manager flag.

`Period` **Three bits.** This field indicates which period the manager is using (the period encompasses the key and the slot value). This value wraps, and so a total of 8 unique periods out of an infinite total number can be identified. All distributed state is associated with the specified period.

`T` **One bit.** Controls presence of `Target Node Identifier`.

`N` **One bit.** Controls presence of `Base Node Identifier`.

`P` **One bit.** Controls presence of `Datagram Identifier`.

`Node Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.This field contains the node identifier of the manager.

`Slot` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This field contains the current slot of the period.

`Datagram Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `P`. This field contains the *next* datagram that will be sent by the manager. No datagram numbers less than the specified number should be accepted from the manager. Note that if the value is zero then any datagram number may be accepted and no state needs to be kept.

`Target Node Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `T`. Present if `T` is one. The identifier must be used as the node identifier of the receiver if it is non-null. Null must be used to force the receiver to explicitly request a node identifier or to continue using an existing node identifier; it is invalid to specify a value using transport multicast. In point-to-point sessions the default

	value is different and the option of not assigning this value does not exist (either the value or the default must be used).
`Base Node Identifier`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `N`. Present if `N` is one. This field represents the node identifier offset associated with the period. All node identifiers used to communicate in the associated period must be adjusted as described below in the section on Node Identifier Compression. However, the node identifiers in this PDU do *not* use compression.

The NODE IDENTIFIER and TARGET NODE IDENTIFIER fields must be different. `dof-2008-12-spec-4`

Configuring a node to use the node identifier of the sender is always fatal and must be rejected.

The TARGET NODE IDENTIFIER must not be specified in transport multicast packets. `dof-2008-12-spec-5`

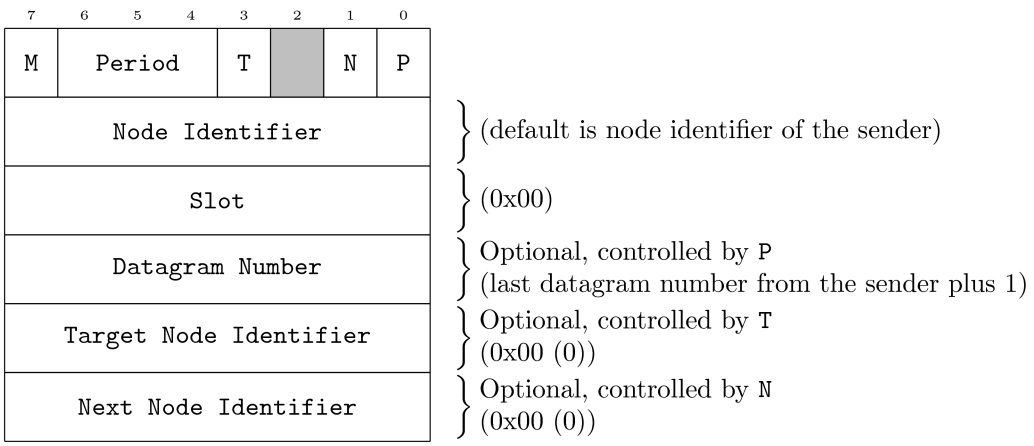
Doing so would force many nodes to use the same node identifier, which is invalid. Note that this requirement does not prevent its use in multipoint sessions. Since this field may arrive in an encrypted packet that does not include a target node identifier, it will be treated as a multipoint session. This would conflict with the requirement of assigning the target node identifier, and so the requirement is based on transport multicast rather than multipoint session.

2.5. Security Header

The following PDU is used in the DPP security header for both lossless and lossy transports. It is substantially the same as used during initialization, but there are differences. The encryption boundary is immediately after this PDU.

CCM Flags Format `dof-2008-12-pdu-6`

Instance of Security Mode of Operation Header`dof-2008-1-pdu-9`



`M` **One bit.** If set, indicates that the sender is the manager of the state and the key.

`Period` **Three bits.** This field indicates which period the sender is using (the period encompasses the key and the slot value). This value wraps, and so a total of 8 unique periods out of an infinite total number can be identified. It is always treated as a positive number, and increases with each new key.

`T` **One bit.** Controls presence of `Target Node Identifier`.

`N` **One bit.** Controls presence of `Next Node Identifier`.

`P` **One bit.** Controls presence of `Datagram Number`.

`Node Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by session type - present in multipoint sessions.** This field contains the unique identifier of the sender in the context of all users of the same key. The value is passed as an offset from the BNI as described below.

`Slot` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by session type - prsent in multipoint sessions.** The field contains the sender's idea of the current slot. This is discussed further later on. The base slot value is obtained during state initialization as described above.

`Datagram Number` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `P`. This field is the sequence of the incoming datagram from the sender. If the datagram number is not present in a datagram then the value in the current state *plus one* is used as the nonce for the datagram. If the datagram number is present then it specifies the nonce value of the datagram. If the datagram is accepted then the

	nonce value of the accepted datagram is written back into the state.
`Target Node Identifier`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `T`. Present if `T` is one. The `Target Node Identifier` can be used to restrict the target in the case of multipoint sessions. It is never used in point-to-point sessions. It is optional in multipoint sessions. When specified it indicates the node identifier of the node that the datagram is being sent to. The use of the target node identifier allows for shared state to be correctly used in multipoint sessions. Without specifying the field it is possible that a datagram sent with transport unicast could be replayed against another node in the group. In this case the datagram might be accepted, but incorrectly interpreted because of different state of the receiver. Treating the datagram as session broadcast prevents incorrect shared state from being used. When the node identifier of the recipient is known and the datagram is meant solely for that node then the target node identifier should be used. The **Target Node Identifier** is passed as an offset from the BNI as described below.
`Next Node Identifier`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `N`. This field synchronizes state between the manager and nodes that may become the manager. It is only required periodically, and always on transport multicast sessions.

The NODE IDENTIFIER field must only be used in multipoint sessions, where it is required. `dof-2008-12-spec-6`

The use of the NODE IDENTIFIER field is reserved for situations where there is ambiguity about the sender. This includes all datagrams in multipoint sessions. Multipoint datagrams must include the field. Point-to-point datagrams must not include the field.

The Slot field must only be used in multipoint sessions, where it is required. `dof-2008-12-spec-7`

Multipoint datagrams must include the field. Point-to-point datagrams must not include the field.

The Datagram Number must be used lossy and may be used lossless. `dof-2008-12-spec-8`

The lack of consistent state forces this requirement, although lossless nodes are encouraged not to pass the datagram number and use the default behavior.

The Target Node Identifier must only be used in multipoint sessions in order for the datagram to be treated as unicast. `dof-2008-12-spec-9`

Any datagram which is not intended for all nodes that share the same key should include this field. Multipoint datagrams that do not specify the field must be treated as transport broadcast within the session, independent of actual transport usage. Transport broadcast/multicast datagrams that specify this field must be treated by the receiver as if they had been sent transport unicast. Attempts to use the field in non-multipoint situations must result in the secure session being terminated.

The Next Node Identifier must only be used by the manager in transport multicast sessions. `dof-2008-12-spec-10`

This field must only be used on transport multicast sessions, and must only be used by the manager. This means that the **M** field must be set. Attempts to use the field in non-transport-multicast situations must result in the secure session being terminated.

2.6. Node Identifier Compression

The use of node identifiers (both as source and target) requires some explanation. The range of values for node identifiers is from 0 to 2^{29}-1. In many cases the total number of nodes is small. However, as described below, the node identifiers are assigned sequentially. This means that over time the node identifiers will become large, increasing datagram size. The manager may determine to optimize wire size by passing an offset associated with the period, called the **Base Node Identifier** (BNI). This offset is used to normalize all node identifier references around the offset, meaning that node identifiers can be passed on the wire using smaller relative offsets.

The **Target Node Identifier** and **Node Identifier** of CCM.4: CCM Flags Format are both passed as an offset from the BNI. The format on the wire is stored as the absolute value of the difference, with the sign bit of the difference stored in the *low bit* of the number.

For example, a node identifier of 10 with a BNI of 5 would be passed on the wire as 10 (10 – 5, shifted up with a sign bit of 0). A node identifier of 2 with the same BNI would be passed on the wire as 7 (2 – 5 = (-)3, shifted up with a sign bit of 1). Placing the sign bit in the low bit allows for compression of negative numbers.

2.7. Node Identifier Caching

The requirement to include the target node identifier in some instances can cause performance problems. In the case of a response (the most common case), the target node identifier for the response was present in the corresponding command and should be easily remembered.

However, in the case of a unicast command the node identifier of the target may not be known. In this case the CCM mode of operation provides a method to determine the appropriate target node identifier. This is only an issue in multipoint sessions.

The problem with determining the node identifier of a given target is that it would appear to require a unicast command, resulting in a logical problem (you need to send a unicast command in order to send a unicast command)! There is no secure solution to this problem, and so the only way that you can know the node identifier is through either a response to multicast/broadcast command (cached as described) or though a unicast datagram that is multicast/broadcast safe.

This solution involves a cache, populated on incoming datagrams, which include the transport address information and the sender's node identifier. Since a typical case is to issue a multicast/broadcast command, receive a response, and then potentially send commands directed at the responder this kind of caching can be very effective. The second part of the solution is being able to resolve cache misses. This is done through the Query command, described later.

Whenever a cache is involved there is the issue of staleness. This makes how long to maintain a cached value important. In general it is safe to remember the node identifier until:

- The transport address it is associated with is no longer valid.
- The node identifiers of the group are invalidated (usually because of group merging or period state synchronization).
- An authenticated datagram from the same address is received using a different node identifier.

It is required, however, that cached information be cleared when the group is notified that its node identifiers are invalid, usually associated with a group merge.



Correctly cache and clear the cache of target node identifiers. `dof-2008-12-spec-11`

Specifically, it is required that the cache be cleared when the group is notified that its node identifiers are invalid and when state synchronization invalidates the node identifier.

2.8. CCM Header Fields

The following definitions come from section 4.2 of NIST Special Publication 800-38C [http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf].

A, Alen, a	For encrypted packets these values are set to authenticate the entire DPP header. For non-encrypted packets these values are set to authenticate the entire payload starting at the DPP header.
P, Plen	For encrypted packets these values are set to encrypt the payload starting at the beginning after the Security Header defined above. Otherwise Plen is zero (0).
K, Klen	The block cipher key is based on the . Note that it is possible that multiple keys may exist in the Security State, and care should be taken to use the appropriate key. The Klen value is determined from the negotiated CCM mode.
N, n	The nonce length 'n' is eleven (11). The nonce 'N' is defined by the and the fields present in the Security Header as defined below.
Q, q	The datagram length 'q' is four (4). The 'Q' parameter is based on this number as indicated in the CCM specification.
t	The MAC length is determined by the and initialized during mode setup.

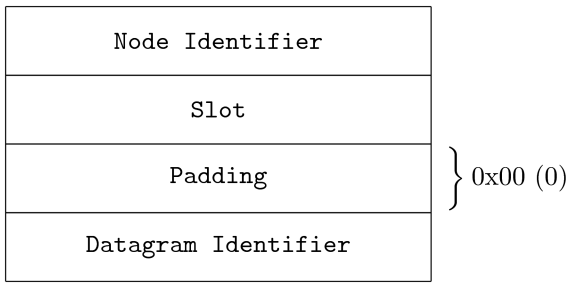
2.9. Nonce Formatting

The CCM mode of operation requires a unique nonce for each datagram. The primary purpose of the security header is to pass the information that makes up this nonce, although the specific fields that are required depend on the session type as described above.

The actual nonce used in the CCM algorithm is a function of the current security state for the session and the fields present in the security header. Once a datagram is accepted the security mode may modify the security state of the session.

As specified above, 'n' is eleven (11). This indicates that an eleven byte nonce is required. However, such a large nonce is usually not passed on the wire because the fields are compressed and possibly not present. The following format is used for the nonce. The nonce is determined from current state information in addition to values that may be present in the PDU. Note that the fields that make up the nonce may be stored differently than the nonce format.

CCM Mode Nonce Format `dof-2008-12-pdu-7`



`Node Identifier`	**Four bytes, MSB.** The sender's node identifier.
`Slot`	**Two bytes, MSB.**
`Padding`	**One byte.**
`Datagram Identifier`	**Four bytes, MSB.**

On receipt of a datagram and its acceptance (meaning that it decrypts correctly) then the nonce used for the datagram is created in this format and used as described below.

2.10. Node Identifier Assignment and Query

In order to securely communicate using CCM each node using the same encryption key requires a unique node identifier, ID_{NODE}. This identifier is used as part of the encryption and authentication of datagrams sent by the node (it is part of the nonce, described above). It is **extremely** important that these identifiers do not collide with *any* other node which could use the same encryption key. This includes nodes that are not in direct communication, because an attacker is assumed to have power to partition a network and block communication (although not indefinitely).

It is the responsibility of the CCM mode of operation to choose and to validate ID_{NODE} for each session in which a different key is used. It operates in conjunction with the appropriate key distribution protocol in some cases to know when the key is changed or when its validation is possibly no longer correct.

In the case of a lossless session that involves only two point-to-point nodes, the assignment of node identifiers is simple and based on the role of the node (client or server). For lossless point-to-point sessions the default values of the initialization may be used for the client. This method of assignment does not work in a multipoint situation and is not used in the lossy point-to-point situation. However, in every active multipoint group there is a manager who is in charge of the CCM state, and so in the multipoint situation

the manager is responsible for assigning node identifiers. Point-to-point lossy initialization is managed by the server.

The requirements of this section apply only to multipoint sessions where node identifiers are assigned by the manager.

The creation of a multipoint session always begins with the establishment of a manager. It is the manager that maintains the group security state, and it is also the manager that shares this state with new members. However, it is not required that the manager remain the same for the lifetime of the multipoint session, and so provision for transferring the responsibility of assigning node identifiers must be made. This problem is complicated because of the requirement that no two nodes in a multipoint session share the same node identifier.

In order to manage this unique assignment a sequential ordering is used. The manager knows the next node identifier (which is incremented each time one is assigned). This information is periodically shared with the members of the multipoint session. If any member needs to become the manager it will take the last shared state and increment it by a fixed number (representing skipping assignments that had been possibly made by the previous manager without synchronizing state). Duplication due to wrap-around is prevented by invalidating current node identifiers based on the period state synchronization.

Note that there is a special case whenever a node is beginning to use a new key that it knows is unique (randomly generated). In this case it knows there are no other nodes using the key and so it can pick ID_{NODE} using any method it wants and immediately begin sending encrypted datagrams if it desires. This is true because it is the manager and owns the security state of the entire group.

Once the node state is determined, all incoming traffic must be monitored for collisions. If a collision of ID_{NODE} is detected by the receipt of a *valid, authenticated* datagram using the same identifier then the receiving node must invalidate its node identifier, and obtain a new one just as if the node identifier had been invalidated by the manager.



Monitor inbound PDUs for node identifier collisions. `dof-2008-12-spec-12`

As described, the receipt of a valid, authenticated colliding datagram must invalidate the node identifier of the receiver or remove it from its list of possibilities if it is negotiating.

2.10.1. Node Identifier Range

The node identifier has a total range of 29 bits due to its compressed format. There are several different values that are tracked by the implementation.

Conceptually there are divisions as follows:

- NID_{NEXT}: Known only to the manager, this is the actual “next” node identifier that will be assigned. It is incremented (with wrapping) each time a node identifier is assigned. NID_{NEXT} must never be greater than NNI.

- NNI: Distributed by the manager as part of its state, the NNI represents the range of node identifiers that the manager is reserving immediately for future assignments. Note that the NNI is the *advertised* version of the reserved node identifier range, and does not relate directly to the NID_{NEXT} that will be assigned by the manager.

- $NID_{INVALID}$: A range that extends from NNI to NNI + $2^{29}/8$ (exclusive, wrapped). Nodes using a node identifier in this range are immediately invalidated.

- $NID_{WARNING}$: A range that extends from the end of $NID_{INVALID}$ for the next $2^{29}/8$ (exclusive, wrapped). Nodes using node identifiers in this range should use the Request command to determine a new node identifier, based on their position in the range. The closer to $NID_{INVALID}$ that a node gets, the more quickly it should obtain a new node identifier.

2.10.2. New Manager Logic

As mentioned above, when a new manager is created knowing that there is no previous multipoint session state, it is free to determine node identifiers without fear of collision. In this case it:

- Sets NID_{NEXT} to zero (0).
- Assigns its own node identifier (which will be zero). This assignment increments NID_{NEXT} to one (1).
- Determines NNI. It is free to determine this as any value, although the choice may limit its ability to assign node identifiers in the future based on the restriction that NID_{NEXT} must not be greater than NNI.

2.10.3. Takeover Logic

When a new manger takes over for a previous manager it does so knowing NNI. In this case it:

- Sets NID_{NEXT} to NNI.
- Determines a new NNI value.
- Synchronizes the session state, including the new NNI.

2.10.4. Modifying NNI

As the manager assigns new node identifiers NID_{NEXT} will approach NNI. Based on its need for more node identifiers, the manager may modify NNI. In doing so, it:

- Determines the new NNI value. This must be done by incrementing the current NNI value by less than $2^{29}/32$.
- Synchronizes the session state, including the new NNI.

2.10.5. Query

Query a target node for its node identifier.
Session: Lossy (2-node), Lossy (n-node)
Addressing: Unicast
Security: Session (since CCM is a security mode of operation, the session must be secure).
Permission (command): None
Permission (response): N/A



Query (command) `dof-2008-12-pdu-8`

SESSION: The multipoint session.
TADDR: The node to query.
COMMAND/DIRECTED
OPID

Instance of CCM Application PDU Format`dof-2008-12-pdu-1`

```
Opcode  = 0x01 (1)
```

`Opcode` **One byte.**

2.10.5.1. Flows

Query() $\rightarrow$ (none) : Failure.

Query() $\rightarrow$ (any valid secure datagram) : Success, ID_{NODE} extracted from the nonce.

2.10.5.2. Details

When a node needs to determine the node identifier of a target node it has two options. The first is to send a multicast/broadcast command and cache the information from the response. The second is to use this command, which operates in the same way but can be sent transport unicast. Note that the receiver will see no target node identifier (as the sender does not know it) and therefore treat the command as if it was received session broadcast, but that is allowed for this datagram.

The correct response is to send any secured datagram back to the sender, using as target node identifier the node identifier obtained from the command. This ensures that the datagram will be treated as unicast, and the cache of node identifiers will be updated based on the transport address of the sender.

2.10.6. Request

Request a new node identifier from the multipoint session manager.
Session: Lossy (2-node), Lossy (n-node)
Addressing: Unicast
Message authentication: Not allowed (command), Allowed (response).
Unsecured: Required (oommand), Not allowed (response).
Permission (command): None
Permission (response): None

⇄ **Request (command)** **`dof-2008-12-pdu-9`**

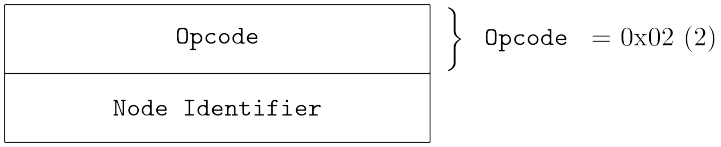
SESSION: The multipoint session.
TADDR: The node to query.
COMMAND/DIRECTED
OPID
Instance of CCM Application PDU Format`dof-2008-12-pdu-1`

```
Opcode  = 0x02 (2)
```

`Opcode` **One byte.**

Request (response) `dof-2008-12-pdu-10`

RESPONSE
Instance of CCM Application PDU Format`dof-2008-12-pdu-1`



`Opcode` **One byte.**

`Node Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.The node identifer assigned to the node by the manager.

2.10.6.1. Flows

The following flows are allowed for this command:

Request() $\rightarrow$ (none) : Failure.

Request() $\rightarrow$ (response) : Success, ID_{NODE} extracted from the response.

2.10.6.2. Details

When a node's node identifier has been invalidated for any reason, it may use this command to obtain a new one. The request is sent unsecured, since the node no longer has a node identifier. The response is sent securely (the node can still receive secure traffic, it cannot send).

Since this command is sent to the secure session, it must identify the session in the presentation layer.

Note that only the presentation layer session identification is used, and that this may not result in a single matching session (specifically, domain differences may cause duplication). In this case, a secure response should be sent in each matching secure session. Any response to the wrong group will be safely ignored because of key differences.

2.11. Security State

The use of security on a session always involves some amount of shared state. How this state is obtained originally, synchronized, and validated is critical to the security of the system.

Each mode of operation requires potentially different state, and so each mode describes the state it requires, where it comes from, and how it changes over time. This section defines the state for all CCM modes of operation.

Note, however, that the distribution of state is integrally tied to the distribution of a new key. This means that it is a joint responsibility of both the key distribution algorithm and the mode of operation to distribute the state. It is interesting that in order to maintain modularity, the key distribution algorithm is asked to authenticate and distribute information that it likely does not understand.

The initial distribution of state always proceeds from the manager to the node receiving the new key. The master is always the *opposite* of the node making the request for the key, which usually means that the server is the master.

Once initial state has been determined and distributed, the mode of operation determines the rules for state modification. State modification can include any modification to the state, as well as the establishment of new state. Again, the establishment of new state is usually integrally tied to key distribution, although this is not always the case.

In the case of CCM the establishment of state proceeds through two stages: first is the mode request, second is the state establishment. The mode request is part of the key distribution process and the format of the mode request is described above. The state establishment includes information not present in the mode request. The format used for state establishment is also described above.



Manage security state as described. `dof-2008-12-spec-13`

The following sections cover specific requirements. This specification includes correctly determining initial state, monitoring state change, and managing state based on incoming PDUs.

2.11.1. Mode Flags and Mode of Operation

The state includes information about the mode flags that were part of the original mode request. In the case of CCM, this includes things like the MAC length, whether the datagram is encrypted, etc. This information is constant for the life of the state, and it is the same for all nodes that are communicating.

Any attempt by a node to use a different mode of operation will result in a malformed datagram.

2.11.2. Session Key

Each session shares a session key. The session key is managed through a key distribution protocol. Determining the session key is critical to the validation of the datagram.

The number of simultaneous keys that may exist in the state is a function of the key exchange methods used. It may be more than one, and it may be that for periods of time different keys are used outbound vs. inbound. Here is an example using TEP:

- A and B are communicating using K_1.
- A begins rekeying using TEP.
- B communicates with the AS and sends an Authorize Session response representing K_2. At this point B can send datagrams using , but must still receive datagrams using K_1.
- A receives the Authorize Session response and sends the Confirm Session command (encrypted with K_1). A begins using .

Node B in this example is required to use two keys, K_1 and K_2, while node A processes the Authorize Session response. Alternatively, node B could queue datagrams until the Confirm Session command is received, switch to K_2 and then send the queued datagrams.

This description applies to point-to-point communications. When multicast/broadcast is involved the situation is compounded by needing to keep the state of each node synchronized. This can be impossible to achieve perfectly, and so some tolerance for nodes being “out of sync” needs to be built into the state. However, nodes desire to limit the number of states (and particularly keys) that they need to manage.

The example given above was simplified because only two nodes were involved, but also because they assumed no datagram loss. In the general case these are not good assumptions. In general, then, the particular state being used must be known.

The most critical issue is to determine the correct key. As mentioned, this is easy in the lossless case. In the lossy case, while there are many possible indicators of the correct key, they are not unique enough to be trusted completely. Instead, the part of the security header controlled by the security mode includes the **Period**. The period indicates which key the sender used. If the receiver is aware of the information for that period then it can decrypt the datagram, otherwise it cannot.

The period increases by one each time a new key is distributed. This is always done by the manager, who flags all datagrams by setting the **Manager** field. The value of this field can be used by the receiver to indicate that its state is out of sync with the manager's state (and the manager's state is the one that matters most since everyone else should be trying to synchronize to it).

Each unique period indicates both the shared key used and also the correct values for the slot fields. Slots increase within a given period, and for each period the slot value should reset to zero.

For example, assume that a node has recently joined a multipoint session. As part of the process of joining the session the node received both a session key K_1 and the initial state, which included a period and slot. Datagrams sent with this key will use the current slot value associated with the key, which changes as described below. When the datagram is received the period can be compared against each known state to determine which to use. If the datagram references an unknown period then it must be dropped (it cannot be decrypted).

2.12. Nonce

The handling of the nonce has been discussed earlier. Note that the state includes a nonce for each sender according to the node identifier *for each period*. If there are multiple periods in use then the nonce state is kept for each period. Since a key is always identified with a period, and the period identifies a security state, this is the same as saying that the nonce storage is part of the security state.

For point-to-point sessions this means storing a nonce for two nodes. For multipoint groups this means *conceptually* storing a nonce for each member of the group.



CCM nonces must only be accepted once, and must have increasing values of slot and datagram number. `dof-2008-12-spec-14`

This requirement prevents replay attacks. Note that the datagram number may reset as long as the slot value increases. For lossless sessions this requires keeping state for both nodes. For lossy sessions certain optimizations to state storage may apply.

There are several optimizations that can be applied to nonce storage that meet the requirement specified above but also allow state to be pruned.



Nonce state storage optimizations *never* apply to lossless transports, *only* for lossy transports. Lossless sessions must keep full nonce state information. The following information only applies to lossy sessions.

The ability to prune information relies on the ability to guarantee that a replayed datagram will not be accepted a second time. Assume that any modification of the datagram will result in it being corrupt and ignored, so modified datagrams cannot be used to corrupt state.

To understand pruning a set of rules that will guarantee security without saving complete nonce state is defined. The rules define when a particular nonce state is no longer required.

The following rules are applied for each node in the group and the manager *for the state associated with a given period and group manager*:

- When the group manager creates a key it associates a slot number (S_G) with it. The initial value is zero (0). The manager also determines the (possibly future) time that the group shall use the key for transmitting, and the slot number (S_G) begins to increment each second after that.
- When a node receives a distributed key, the manager distributes the current value of S_G to the node in the state initialization, and from that point the node maintains its own sense of S_G, according to the rules listed below. If the key will become active in the future then the delay is distributed along with key. In that case the value of S_G does not begin to increment until after that delay.
- In addition to the node requirements, the manager uses a function of input slots to modify S_G as described below.

The value of S_G represents the current slot. It is passed by each node in each datagram in the nonce. The rules for using this value from the nonce are listed below. The rules for modifying the value of S_G are as follows:

- When the initial state is received from the manager the receiving node must wait until the next slot to send any encrypted traffic. This requirement prevents a quickly rebooting node from reusing the same nonce.
- If a valid datagram is received with the **Manager** field set and an S_G value that is greater than the nodes value then the node updates its S_G to that slot. This synchronizes slow nodes with the manager.
- If a valid datagram is received with the **Manager** field set and an S_G value that is less than the node's value then the node stops its periodic updates to the corresponding S_G for a number of seconds equal to the difference in values. This synchronizes fast nodes to the group manager. Note that this may cause the node to use larger Datagram Counts than it normally would.
- Each node updates its own S_G each second by incrementing it by one (1). It uses this value in all nonces sent using that security state.



Manage CCM slots as required, including delays for newly distributed keys.

`dof-2008-12-spec-15`

The specific requirements are discussed earlier in this section.

The effect of these rules is to keep all nodes using roughly the same values for S_G. However, there are cases where drift will occur, primarily when there is little communication on the group and so the group manager's sense of S_G varies from the majority of the nodes.

In order to address this situation, the group manager should apply a filter to the slot numbers that arrive (in valid datagrams) and adjust its value of S_G based on this filter. This means that if the majority of transmitting nodes have S_G values greater than the node manager that it will "speed up". As it does this and sends S_G values to the rest of the group they will also speed up, keeping them in sync with the faster (communicating) nodes.



Care should be taken to ensure that a single (or a few) nodes with radically incorrect clocks do not skew the sense of the current slot in a disproportionate way. For example, a single node with an internal clock running at 10x speed should not have a huge effect on the group slot. This behavior is not required, but not implementing it can cause the node to miss datagrams and if not implemented by a manager can cause the group to behave badly.

This sense of a common (or roughly common) value of S_G can be used to solve the nonce state problem. In general, a node saves nonce state in a way that it can easily look up the most recent value given a particular

node identifier, can easily remove all state with a given slot number, and has a sense of how much state it desires to maintain. It knows the oldest value of S_G that it has state for.

When a datagram arrives, the nonce will contain the node identifier, the slot, and the datagram number. The following algorithm will handle rejecting duplicates:

- Check that the datagram slot number is greater than or equal to the lowest slot number for which the node has state in that period. If it is lower then drop the datagram. This is done because the node has removed the state for that slot and cannot know if the datagram is valid or a replay. Note that this allows multiple slot states to exist.
- Look up the nonce state based on the node identifier. If there is no state then accept the datagram and create state with the given values of slot and datagram number.
- If the state exists then check that the slot from the datagram is greater than or equal to the slot in the state, and if they are equal that the datagram number is greater than that in the state. Update the state with the newly received slot and datagram number.



When a node becomes aware of a new period it must act as though it has no state for prior slots in that period. This may cause datagrams to be ignored during node identifier validation, if the sender has a slow-running clock. Rather than create special cases for this situation the datagram will be ignored. The situation should resolve itself when the validating node sender sends its first encrypted datagram, which will collide with that of the node with the slow-running clock.

If in this process the node determines that it is keeping too much state then it can remove the state associated with older slots, starting with the oldest. In doing this it keeps track of the new lower limit of slot value that it will accept. If it must remove the state associated with the current slot (or even future slots) then it must ensure that it will not accept future datagrams in those slots. This will likely cause datagram loss, but it ensures security.



For any old period, CCM slot state must be invalidated within two state synchronization periods. `dof-2008-12-spec-16`

This requirement prevents any possible attack on old data. Old periods are marked once the manager begins using a new period. Once a period is old, the implementation may choose to invalidate its slot data. This must occur at most after two new periods are defined by the manager.

Other ways can be used to reduce the stored nonce state. Extreme care should be taken that the fundamental requirement of not accepting a replayed datagram are not violated.

- If a node knows that it will never be able to respond to an incoming request (for example, it knows that it is a requestor and will never provide) then it can ignore the state for all those requests. Note that the calculation of S_G should still use the datagram as input, but the actual nonce state does not need to be kept.
- If the node does not have permission to respond then it may do almost the same thing. However, it must ensure that if its permissions change to be more allowing that it must reject all incoming requests that it might have responded to earlier but that it ignored. This usually means rejecting requests until the value S_P (the slot value when the new permission was obtained) is removed from state. This introduces a delay during which requests that utilize the newly obtained permission will still be rejected just as they were when the permission was not granted.

Just as slot data is invalidated, period data (which is a superset of slot data and includes the key associated with the period) is also invalidated.

CCM period state must be invalidated within two synchronization periods of the establishment of a new period. `dof-2008-12-spec-17`

This requirement prevents any possible attack on old data. Old periods are marked once the manager begins using a new period. Once a period is old, the implementation may choose to invalidate its slot data. This must occur at most after two new periods are defined by the manager.

Note that valid unicast datagrams (including responses) must always be added to state.

Drop all datagrams that include a target node identifier not equal to the receivers. `dof-2008-12-spec-18`

The mode flags may include the target node identifier as specified by the sender of the datagram. If the target node identifier is present and does not match the receiver the datagram must be silently ignored. Without this logic, a datagram sent to one node could be echoed to another node. Since responses are sent unicast and may be generic enough that it would match another requestor's command, care must be taken not to accept these "echoes".

The preceding information details the requirements for handling nonces. The next section deals with datagram numbers.

2.13. Datagram Number

Some information on the handling of datagram numbers has already been discussed. As with the nonce (which the datagram number is part of), the general requirements are to accept a datagram a single time, and require increasing values of datagram number.

CCM datagram number must only be accepted once, and must have increasing values. `dof-2008-12-spec-19`

This requirement prevents replay attacks. Note that the datagram number may reset as long as the slot value increases. This only applies to multipoint since the slot is not used point-to-point.

The datagram number is part of the nonce state discussed earlier. The following describes special cases for the use of datagram numbers on lossless sessions.

Note that there are special rules for the handling of lossless nonces and datagram numbers. As discussed above in the description of the nonce, the value for the nonce is based on the current state *plus one* or is the value in the datagram. If the datagram is authenticated (the datagram is valid and decrypts) then the datagram number used in the nonce is written back into the state.

The result of this rule is that if the security state is initialized as described above then no nonce ever needs to be passed over a lossless session. The sender will get its value from its state, increment it, use it to encrypt and store the incremented value back to state. When a datagram arrives it will get the state, increment it, decrypt, and if successful store the value back.

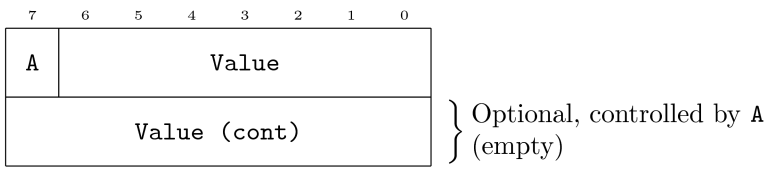
Note, however, that it is still valid for a datagram to include a datagram number. If so the value in the state is not used, but the datagram number must be greater than that stored in the security state (as specified above). If the datagram is valid then the datagram number used is stored back in the state.

This means that a node could alternate between passing and not passing the datagram number.

3. External PDU Reference

3.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018)

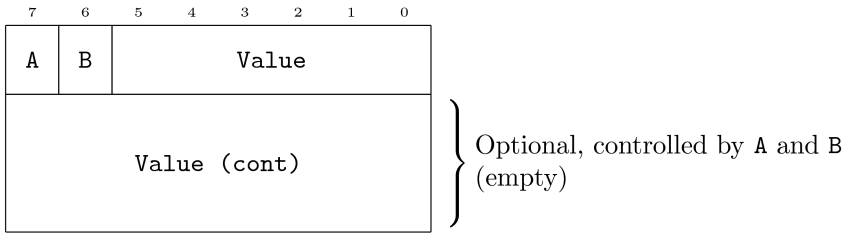
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`A` **One bit.** Controls the size of `Value`.

`Value` **Either seven or fifteen bits, MSB, controlled by A.** If the `A` field is zero then the `Value` is contained in a single byte. If the `A` field is one then the `Value` is passed in fifteen bits with the least significant bits passed in a second byte.

Compressed Unsigned 32-Bit `dof-2009-1-pdu-3`



`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`B` **One bit.** Controls the size of `Value` or becomes part of the value.

`Value` **Either seven, fourteen, or thirty bits, MSB, controlled by A and B.** If `A` is zero the `Value` is seven bits and one byte is used. In this case, the `B` field is part of the `Value`. If `A` is one then the `B` field determines the size of the `Value` rather than being part of it. If `B` is zero then fourteen bits (two bytes) are used, and if `B` is one then thirty bits (four bytes) are used.

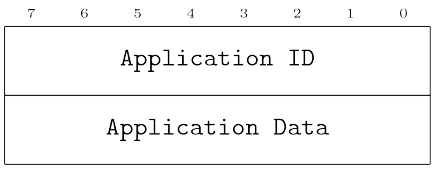
3.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018)

Security Mode of Operation Header `dof-2008-1-pdu-9`



`Fields` **Variable.** The format of the data is controlle dby the mode of operation, which is always known in the context.

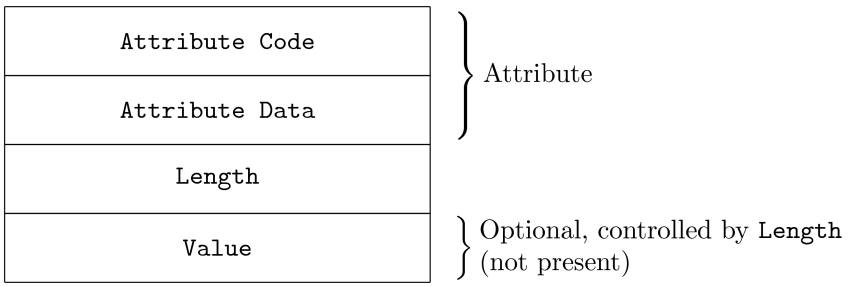
General Application Header `dof-2008-1-pdu-13`



`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Application Data` **Optional, variable length.** Contents are defined by the application.

Attribute/Value Pair `dof-2008-1-pdu-16`



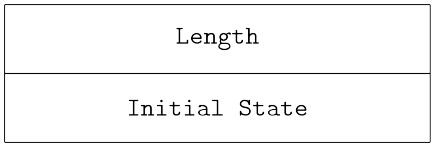
`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

`Attribute Data` **Two bytes, MSB.** This field completes the definition of an attribute in combination with the code.

`Length`	**One byte.** This field determines the length of the `Value`. It may be zero, in which case the `Value` field is not present.
`Value`	**Variable length. Optional, controlled by** `Length`. Contains the value for the associated attribute.

3.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018)

⇄ **Security Mode of Operation Initialization** `dof-2008-16-pdu-12`



`Length`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The length of the `Initial State`.
`Initial State`	**Variable-length, optional.** This data is defined by mode of operation, which is known in all contexts where this PDU can appear.