Secure Group Management Protocol (V1)

DOF Protocol Specification

Technical Steering Committee

Secure Group Management Protocol (V1): DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-3
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	Secure Group Management Protocol (V1), OpenDOF TSC, [dof-2008-3] (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 1
        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. Overview 9
3. Specification 11
    3.1. Application Format 11
    3.2. Security Primitives 11
    3.3. Cryptography 12
    3.4. Negotiation 12
    3.5. Invalid Opcodes 12
    3.6. State Requirements 12
        3.6.1. $STATE_{AS}$ 13
        3.6.2. $STATE_{GM}$ 13
        3.6.3. $STATE_{MEM}$ 14
    3.7. Versioning 15
    3.8. Epoch Comparison 15
    3.9. Group Manager Priority 15
        3.9.1. Versions Understood 16
        3.9.2. Node Desire 16
        3.9.3. Key 16
        3.9.4. Randomized Tie Break 17
        3.9.5. Ties 17
    3.10. Timing and Behavior 17
        3.10.1. Obtaining $STATE_{AS}$ 20
        3.10.2. Obtaining $STATE_{GM}$ 21
        3.10.3. Becoming Group Manager (New Group) 22
        3.10.4. Acting as Group Manager 22
        3.10.5. Merging a Group 23
        3.10.6. Handling Updates to $STATE_{AS}$ 24
        3.10.7. Acting as a Normal Group Member 25
        3.10.8. Becoming Group Manager (Existing Group) 26
    3.11. Sessions 27
4. Command Reference 29
    4.1. Epoch Changed 29
        4.1.1. Flows 29
        4.1.2. Details 29
        4.1.3. Routing Rules 30
    4.2. Heartbeat 30
        4.2.1. Flows 31
        4.2.2. Details 31
        4.2.3. Routing Rules 32

4.2.4. Multiple Versions 32
4.3. Rekey 32
4.3.1. Flows 34
4.3.2. Details 34
4.4. Request Group 35
4.4.1. Flows 37
4.4.2. Details 38
4.4.3. Cryptography 40
4.4.4. Routing Rules 40
4.4.5. Multiple Versions 40
5. External PDU Reference 41
5.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 41
5.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 43
5.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018) 44

List of Figures

3.1. SGMP Join Multicast Group Process 18
3.2. SGMP Multicast Group Manager Process 19
3.3. SGMP Group Member Process 20

List of Tables

4.1. Request Group Cryptography Parameters 40

Specifications and PDUs

SGMP Application Format `dof-2008-3-pdu-1` 11
SGMP Command/Response Format `dof-2008-3-pdu-2` 11
Reject SGMP in DSP Negotiation. `dof-2008-3-spec-1` 12
Ignore datagrams with invalid opcodes. `dof-2008-3-spec-2` 12
Different SGMP versions must use increasing APPIDs. `dof-2008-3-spec-3` 15
Backward compatibility of SGMP. `dof-2008-3-spec-4` 15
Secure multicast nodes have a desire of zero for the first $2*T_{HB}$ seconds. `dof-2008-3-spec-5` 16
Act as group manager if there are no group managers found. `dof-2008-3-spec-6` 21
Follow the correct process before becoming group manager when none is found. `dof-2008-3-spec-7` 22
Correctly handle group manager responsibilities in secure unicast groups. `dof-2008-3-spec-8` 23
Correctly handle group manager responsibilities in secure multicast groups. `dof-2008-3-spec-9` 23
Merge partitioned groups as a group manager when a higher priority manager is found. `dof-2008-3-spec-10` 23
Check for changes to $STATE_{AS}$ at the correct frequency. `dof-2008-3-spec-11` 24
Send Epoch Changed commands as appropriate. `dof-2008-3-spec-12` 24
As group manager, manage the rekey process when $STATE_{AS}$ changes. `dof-2008-3-spec-13` 24
As group manager, allow nodes to join with the most recent $STATE_{AS}$. `dof-2008-3-spec-14` 25
As a new group manager, continue or establish required epoch rekey operations. `dof-2008-3-spec-15` 25
Correctly update all node state when new epochs become effective. `dof-2008-3-spec-16` 25
As a secure multicast group member, listen for and respond to missing heartbeats. `dof-2008-3-spec-17` 26
As a secure multicast group member, listen for and respond to rekey commands. `dof-2008-3-spec-18` 26
Nodes with higher priority must become group manager. `dof-2008-3-spec-19` 26
Take over as group manager if required. `dof-2008-3-spec-20` 27
Epoch Changed `dof-2008-3-pdu-3` 29
Heartbeat `dof-2008-3-pdu-4` 30
Rekey `dof-2008-3-pdu-5` 33
Rekey Epoch `dof-2008-3-pdu-6` 33
Rekey Merge `dof-2008-3-pdu-7` 34
Request Group (command) `dof-2008-3-pdu-8` 36
Request Group Request Block `dof-2008-3-pdu-9` 36
Request Group (response) `dof-2008-3-pdu-10` 37
Request Group Initialization Block `dof-2008-3-pdu-11` 37
SGMP Key Request `dof-2008-3-pdu-12` 38
SGMP Identity `dof-2008-3-pdu-13` 38
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 41
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 41
Object Identifier (OID) `dof-2009-1-pdu-4` 42
Object Identifier Attribute `dof-2009-1-pdu-5` 42
General Application Header `dof-2008-1-pdu-13` 43
Attribute/Value Pair `dof-2008-1-pdu-16` 43
Security Node Identifier `dof-2008-16-pdu-1` 44
Base Credential Format `dof-2008-16-pdu-2` 44
Security Domain `dof-2008-16-pdu-5` 44
Security Scope `dof-2008-16-pdu-6` 45

Permission `dof-2008-16-pdu-7` 45
Permission Request `dof-2008-16-pdu-8` 46
Security Mode of Operation Initialization `dof-2008-16-pdu-12` 46
Key Request `dof-2008-16-pdu-13` 46
Key Grant `dof-2008-16-pdu-14` 47

1. About This Document

This document contains information Secure Group Management Protocol, used with the DOF Protocol Stack. This is critical information for anyone who needs to work with the protocols themselves, and contains useful background information for those interested in the architecture and inner workings of DOF implementations.

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.

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.

Unsecured / Encryption / Message Authentication	Description
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.

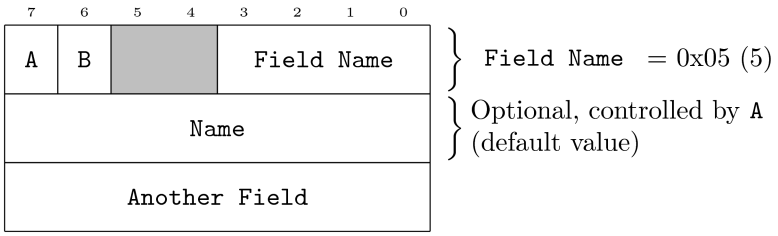
1.3.4. Fields

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. Overview

This section describes the Secure Group Management Protocol (SGMP). SGMP is used by all nodes that can participate in secure multicast groups and also by nodes that can participate in secure unicast groups. SGMP uses the key distribution methods described in the *DOF Security Specification*. It is fundamentally a key distribution protocol, although it also includes group maintenance functions. It does not specifically deal with secure modes of operation, leaving that to the presentation layer of the stack.

SGMP is special as a protocol in that it is never used over a lossless session. It is always used lossy because its sole purpose is managing groups of nodes that are speaking either multicast or unicast, but always using lossy transports.

SGMP is built on the foundation of the DOF Protocol Stack, described in the *DOF Protocol Specification*.

The SGMP is used both to obtain cryptographic tickets from a secure group manager and also to communicate secure group information between group members. Throughout this section the term "secure group" will refer to both secure multicast groups and secure unicast groups. If a feature or function applies to only one type of group or the other, then the more specific term will be used.

The process of joining and maintaining membership in a secure group is almost completely managed by this protocol, with the exception that an authentication protocol like TRP must be used to communicate with the AS. There are also other aspects of secure communication (like the negotiation and use of secure modes of operation) that are not addressed by SGMP. Finally, the methods used to encrypt the data on the wire are not directly discussed here, although the session key used to do the encryption is managed by SGMP.

In the case of secure multicast it is assumed that each node has access to an authentication server and that all nodes can communicate with each other directly using a multicast transport. The opposite is true in the case of secure unicast, where it is assumed that only a single node can access the authentication server directly and that the rest communicate only with that node. The main benefit of the secure unicast group is that very little communication is required to maintain the group and that secure communication can be maintained with a large number of nodes without requiring a unique session key for each node.

The steps for secure group communication are similar for both secure multicast and secure unicast groups. First, the node determines the group that it desires to join. This determination will result in a group identifier (an OID). With this OID and its own authentication information, the node communicates with an AS to determine a Key Exchange Key (KEK) associated with the group, along with other parameters. The specifics of this communication are not discussed here, although it will result in obtaining certain information about the group including the KEK and other timing parameters that affect the use of SGMP.

Once in possession of a KEK the node uses SGMP to locate the manager of the group. In the secure unicast group the group manager is usually configured and not discovered, although discovery may be used. The group manager will securely distribute the current group session key, and the node is then able to communicate with the group directly (multicast) or with the group manager (unicast). If, however, there is no established group then the node may elect to form the group itself, become its manager and then allow other nodes to join.

Once formed, a group must maintain communication in order to remain a group. Due to communication failures a single logical group may be partitioned and merge again over time. Handling communication failures is a key aspect of SGMP. It is particularly difficult to maintain security requirements during times of network instability, as this usually causes a loss of state which can lead to security requirement violations if the specifications are not followed precisely.

All nodes that can communicate in secure groups must support this protocol.

3. Specification

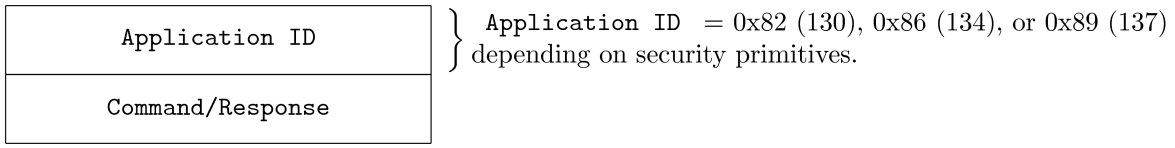
This section contains the specification of the Secure Group Management Protocol.

3.1. Application Format

SGMP uses an application identifier of either 0x82 (130), 0x86 (134), and 0x89 (137) depending on the security primitives desired. When used in the DPS, the following format is used.

SGMP Application Format `dof-2008-3-pdu-1`

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

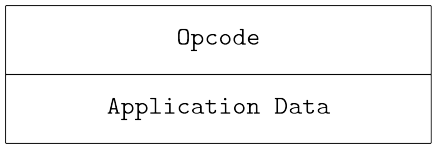


`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Command/Response` **Instance of** SGMP Command/Response Format`dof-2008-3-pdu-2`.Contains the command or response data.

Each application PDU uses the following format.

SGMP Command/Response Format `dof-2008-3-pdu-2`



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

`Application Data` **Variable bytes. Optional, controlled by** `Opcode`. Each **Opcode** will define the required **Application Data**.

3.2. Security Primitives

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

APPID	ENC	HASH	MAC
0x82 (130)	AES	SHA-256	HMAC

APPID	ENC	HASH	MAC
0x86 (134)	Twofish	SHA-256	HMAC
0x89 (137)	SMS4	SHA-256	HMAC

3.3. Cryptography

Cryptography is the science of protecting information using encryption. The cryptography of SGMP is described in detail in the *DOF Security Specification*.

Note that all SGMP cryptography is based on knowledge of the KEK, which is always a 256-bit random (and somewhat stable) key. Although it is possible to obtain the KEK using any credentials understood by the AS, the KEK is always used in SGMP and as such SGMP never deals with any credentials other than a shared 256-bit key. See the Request Group command for more information.

3.4. Negotiation

This protocol must not be used over lossless sessions. Any attempt to negotiate SGMP using DSP must be rejected.

Reject SGMP in DSP Negotiation. `dof-2008-3-spec-1`

SGMP only applies to lossy sessions, never to lossless sessions. Since DSP negotiation only applies to 2-node sessions, any attempted use of SGMP in lossless DSP negotiation must be rejected. SGMP may appear in DSP application discovery.

3.5. Invalid Opcodes

Invalid opcodes must cause the datagram to be ignored.

Ignore datagrams with invalid opcodes. `dof-2008-3-spec-2`

Invalid opcodes signal either corrupted communication channels or loss of state. They must be ignored.

3.6. State Requirements

Each node participating in a secure group must maintain state. The state required depends on whether the node is a group manager or a group member. A group manager is considered to also be a group member, and so must maintain both sets of state.

It is particularly difficult to discuss the state of distributed systems because, invariably, different nodes will have different ideas of what state is "current" versus "future" or "past". This is primarily caused by data loss and delay during communication, which means that having perfectly synchronized state is impossible. Further, network partitioning may cause segmenting, or partitioning, to occur in the group.

Throughout this discussion, and the rest of this document, the current state of the source of the information will serve as the reference. Since the information must pass from the source to other nodes, the other nodes may be "behind" the source, but can never be "ahead" of the source.

A further complication occurs during partitioning and merging. In these cases there may be many different "sources" at the same time. In these cases notations must be used to describe the particular state being discussed.

There are three types of nodes involved with SGMP: the Authentication Server, the Group Manager and a Member. The first two types of node can be the source of state. Group members never source state, although they use it. These sources are indicated with subscripts on the state values of "AS", "GM" and "MEM" respectively.

The following sections describe the different state values from the perspective of the source. Note that as values pass from one type of node to another the subscript used changes. This both indicates the source at a given time, but also indicates that there is the possibility that the value doesn't match the "current" value at the original source.

3.6.1. $STATE_{AS}$

The AS is the central source of almost all security information. Each node that desires to communicate securely must communicate with the AS, either directly or indirectly.

The following AS state is directly related to secure group communication, and is in addition to the security information about individual nodes that it maintains.

TRP is always used to communicate with the AS, and so it is used to distribute this information.

$EPOCH_{AS}$ This number represents the epoch of KEK_{AS} along with the other group parameters. Any change to the AS state associated with a secure group must result in the $EPOCH_{AS}$ value increasing. This includes parameters not directly discussed in this document, but does not include security parameters that apply to the member nodes and not to the group itself.

KEK_{AS} This represents the KEK at the AS.

THB_{AS} This controls how fast Heartbeat commands are sent.

$TMAX_{AS}$ This controls the delay between KEK_{AS} validations for group members.

$TMIN_{AS}$ This controls the delay between KEK_{AS} validations for group managers.

$SSID_{AS}$ This identifies the group, and is part of the security state delivered to all nodes. It is used in the DPP Security Header.

$MODE_{AS}$ The information is used to define the security mode used by the group.

3.6.2. $STATE_{GM}$

The group manager of a secure group must maintain the following state. These parameters are either obtained from the AS using TRP or are maintained by the group manager itself.

$EPOCH_{GM}$ This number represents the epoch of KEK_{GM}. It is obtained from the AS while joining the group and then checked periodically.

KEK_{GM} This represents the KEK at the group manager. The group manager must remember this value in order to allow new group members to join the group. It is obtained from the AS while joining the group and then checked periodically.

THB_{GM} This controls how fast Heartbeat commands are sent. It is obtained from the AS while joining the group and then checked periodically.

$TMIN_{GM}$ This controls the delay between KEK_{AS} validations for group managers. It is obtained from the AS while joining the group and then checked periodically.

$TMAX_{GM}$ This controls the amount of time that a protocol version is active following a join. It is the same as $TMAX_{AS}$.

$SSID_{GM}$ This identifies the group, and is part of the security state delivered to all nodes. It is used in the DPP Security Header.

$MODE_{GM}$ The information is used to define the security mode used by the group.

$PERIOD_{GM}$ This is the current period of the group manager, which changes in sync with K_{GM}. It is assigned and maintained by the group manager.

K_{GM} This is the current session key (the group key) for the group. It is assigned and maintained by the group manager.

3.6.3. $STATE_{MEM}$

Each node in the secure group must maintain the following parameters. For each parameter a brief note is made as to how the parameter is obtained and how it is maintained, although the remaining sections discuss this more fully.

The following state is sourced by the AS, and is distributed to nodes in the group using TRP:

$EPOCH_{MEM}$ This number represents the epoch of KEK_{MEM}. This is obtained from the AS using TRP.

KEK_{MEM} This represents the KEK at the node. This value is required by a node until it becomes a member, at which point it is no longer needed.

$TMAX_{MEM}$ This value controls the delay between KEK validations for group members. It is obtained from the AS using TRP.

THB_{MEM} This value is used for some member timeouts. The value is obtained using TRP as the KEK is obtained.

$SSID_{MEM}$ This identifies the group, and is part of the security state delivered to all nodes. It is used in the DPP Security Header.

$MODE_{MEM}$ The information is used to define the security mode used by the group.

The following state is obtained by communication with the group manager.

$GMADDR_{MEM}$ The transport address of the group manager. This is required so that a group member can send unicast notifications to the group manager. It is managed through the Heartbeat command.

$PERIOD_{MEM}$ This is the current period of the member, which changes in sync with K_{MEM}.

K_{MEM} This is the current session key (the group key) for the group. It is distributed in the Request Group response and changed using the Rekey command.

3.7. Versioning

SGMP is unique in its capabilities because it both controls the group and also distributes a key that allows the members of the group to communicate with other members. This causes problems if multiple SGMP versions are used on the same network. The reason is that without coordination, multiple group managers could exist for the same "group" as defined by the Authentication Server, meaning multiple groups using the same $SSID_{AS}$ values but using different group managers and different K_{GM}. This causes the traffic of the two groups to be received by all the nodes, but only be understandable by the set of group members managed by the same group manager (using the same group key).

Solving this problem means either coordinating the secrets between group managers (which is very difficult), or ensuring that the group manager knows how to manage groups using either version. This is the solution used by SGMP, and it imposes several requirements on potential group managers that are detailed in the remainder of this section.

First, when a new version of SGMP is introduced it must require the ability to speak all past versions. Second, group managers must prioritize themselves so that the active group manager is the one that speaks the highest SGMP version possible. In order to determine this easily an ordering to the APPIDs is used to denote the different versions that must be known. The requirement is included here (which is already required in general but is more specifically required here).



Different SGMP versions must use increasing APPIDs. `dof-2008-3-spec-3`

Any version of SGMP must have an APPID that is numerically greater than all previous versions.



Backward compatibility of SGMP. `dof-2008-3-spec-4`

Any implementations of SGMP on a node that supports secure multicast groups for a particular version must also understand and implement all prior versions of SGMP. This requirement does not apply to nodes that will never be group managers, which may include secure unicast nodes or reduced functionality nodes, however in the multicast case this should be rare.

The commands in SGMP may each specify further version compatibility requirements.

3.8. Epoch Comparison

The epoch as used in this protocol represents an indication of change rather than an absolute number of changes. This means that comparison of epochs cannot be made in an absolute sense. Throughout this protocol all references to epoch comparison refer to a sliding- or floating-window comparison. This is accomplished by treating the two epochs as unsigned quantities, performing a subtraction, and then treating the result as a signed number. If the result is zero then no change has occurred. If the result is positive then the first epoch is greater than the second. If the result is negative then the first epoch is smaller than the second.

Epoch comparison will always be indicated as E_1 (op) E_2, where (op) may be equality or other comparisons.

3.9. Group Manager Priority

Each member of a secure multicast group should be able to become the group manager if necessary. Even so, it may be more or less difficult for some nodes to do this compared to other nodes because of resource

limitations. In order to allow a node to have some control over whether it will become the group manager there are several factors that are used.

This priority does not apply in the secure unicast situation because the group manager is fixed. The following factors are used to determine the priority. The factors are evaluated in the following order until one produces a "winner".

3.9.1. Versions Understood

The most important factor in choosing a group manager is to use the one that speaks the most recent SGMP version. As discussed earlier, the most recent version must have the highest version number. This number is passed in both the Request Group response and Heartbeat command.

However, this requirement does not apply to secure unicast groups as the group manager is fixed.

3.9.2. Node Desire

Each node has a ranking of its desire, which is a number between zero (0) and two hundred fifty five (255). The "normal" value, expressing neither a strong desire nor a strong lack of desire to be group manager is one hundred twenty eight (128).

If a node is resource constrained then it should use a lower desire. A node that desires to be the group manager should use a higher desire.

This ranking parameter is passed in both the Request Group response and Heartbeat command.

For the first $2*THB_{MEM}$ after obtaining an initial KEK value, the desire of a secure multicast node must be zero. This introduces a bias toward nodes that have been part of the group already and have shared state. Once the node with an artificially low desire joins the group and has the state of the group (which will happen within $2*T_{HB}$) then it is free to become group manager using the shared state.

The group manager of a secure unicast group should have a desire of two hundred fifty five (255). A member of a secure unicast group should have a desire of zero (0). Even so, the rules for secure unicast groups mean that the values of these parameters do not matter.



Secure multicast nodes have a desire of zero for the first $2*T_{HB}$ seconds. `dof-2008-3-spec-5`

This allows nodes to obtain shared state before becoming group manager.

3.9.3. Key

When the version and rank are equal then a determination of which group manager out of multiples will remain the group manager must be based on other factors. The first of these factors is the group key.

This factor cannot be used in the Heartbeat command case, because of necessity the group key must be equal or the command would not be understood.

In the Request Group response case the following applies. In this case both managers must know the same value of KEK_{GM} because otherwise SGMP would not have been able to distribute a key. However, each manager will be using a different session key K_{GM} (except for the almost impossible case of both

group managers using the same K_{GM}). We will denote the newly received session key as K_{MEMnew}, and the receiving manager's current session key as K_{GM}. The receiving manager will compute V_{OTHER} as K_{MEMnew} xor KEK_{GM}, and V as K_{MEM} xor KEK_{GM}. If $V_{OTHER} > V$ using an unsigned comparison of bits then the receiving manager ceases to be group manager.

Note that the exclusive-or operation helps to prevent knowledge of the relative key values from being determined by an attacker.

3.9.4. Randomized Tie Break

Each group manager chooses a random non-zero 32-bit number when it becomes group manager. This number is used in the Heartbeat command to break ties. Note that in the case of multiple group managers in the same group that many of the previously discussed items will be identical by definition. This random field allows one of the group managers to remain. The random number is unsigned, and the node with the higher value has the higher priority.

3.9.5. Ties

There are extremely rare circumstances (because of key comparison this would be one in 2^{256} chance or practically impossible, or one in 2^{32} in the case they are in the same group) in which all of the above properties will match. In the case of a priority tie both group managers must cease being manager. This will force a new group manager to be chosen, which will resolve the tie (likely by the choice of a new K_{GM} value).

This is done because it is better to have no group manager rather than two.

3.10. Timing and Behavior

SGMP is used by nodes that desire to participate in a secure group. Secure groups in the DOF protocol stack are based either on multicast or unicast. For multicast the commands in this section are typically sent using a multicast transport using multicast transport addresses. In the unicast system all of the commands are sent unicast and multicast is typically not used at all.

A node must follow certain rules in order to participate in a secure group. This section describes these rules with a particular emphasis on timing and state requirements.

The following flowcharts indicate the general states and transitions that occur. The full requirements are detailed in the sections that follow and in the command reference.

There are three fundamental phases to SGMP. The first is actually not managed by SGMP, but rather through communication with the domain through an authentication server. A node that is configured to be a member of the group will be able to successfully communicate with the domain and obtain state that allows SGMP to be used at all. A node that cannot obtain this state cannot even begin the process of joining a group.

Once a node authenticates to the domain, it can begin the process of authenticating to the group itself. At this point, the node is not a member of the group, but as long as it has authenticated to the domain then it will eventually become a member or will become the group manager.

While either a member or manager, if the node fails to authenticate to the domain, then it must leave the group. To leave the group means to delete group state and notify the application that the group membership is lost. This would be the same as a session closing.

Figure 3.1. SGMP Join Multicast Group Process

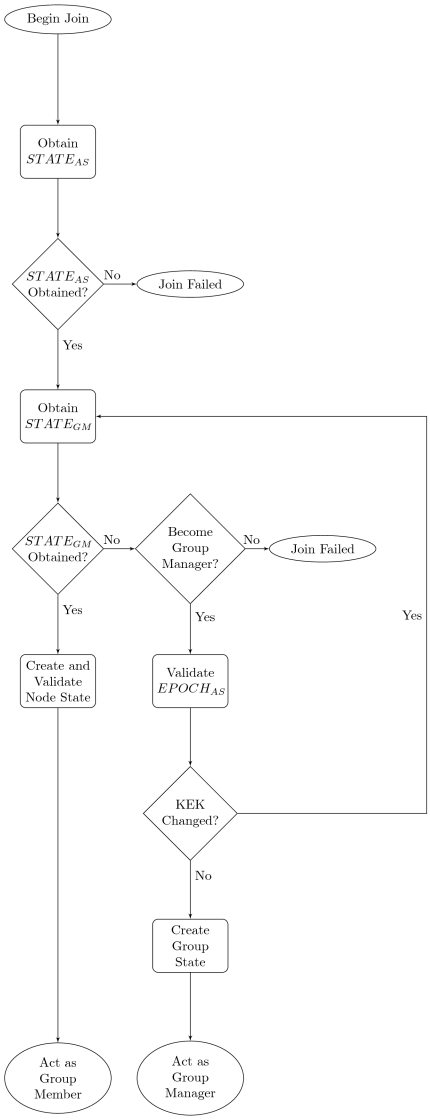


Figure 3.2. SGMP Multicast Group Manager Process

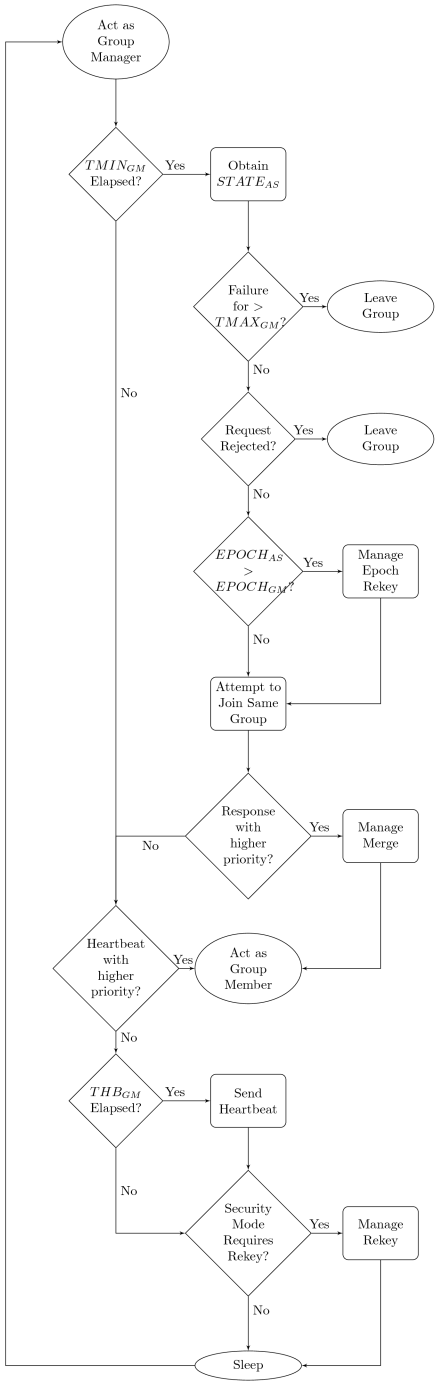
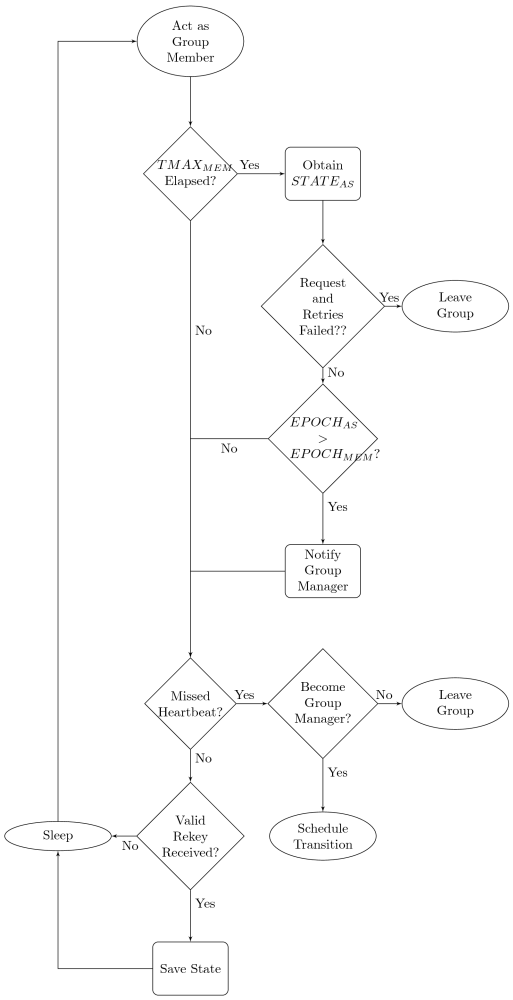


Figure 3.3. SGMP Group Member Process



3.10.1. Obtaining $STATE_{AS}$

The first step in joining a secure group (either multicast or point to multipoint) is obtaining the state managed by the AS, identified as $STATE_{AS}$. $STATE_{AS}$ is managed by the authentication server, and a key distribution protocol like TRP allows an authorized node to obtain the state including KEK_{AS}. In addition, the state will include timing and other parameters including $EPOCH_{AS}$. Some of these parameters govern the nodes timing behavior. Once retrieved, these parameters are stored in the node state as part of $STATE_{MEM}$ ($EPOCH_{MEM}$, KEK_{MEM}, THB_{MEM}, $TMIN_{MEM}$ and $TMAX_{MEM}$). The reason for this renaming is that once parameters are received they may immediately become out-of-sync with the sender.

If $STATE_{AS}$ cannot be obtained then the node may not join the group. The timing parameters govern how frequently $STATE_{AS}$ must be validated (checked for change). This check is made by querying the AS and obtaining the parameters again. If $EPOCH_{AS}$ is not equal to $EPOCH_{MEM}$ then the parameters have changed and the newly received parameters must replace the ones currently in use by the node. The required logic in these cases is discussed more fully later on.

3.10.2. Obtaining $STATE_{GM}$

Once $STATE_{MEM}$ has been obtained the next step is to obtain the $STATE_{GM}$. This state includes the current session key (K_{GM}). The distribution of group state information usually includes security mode state, which is distributed along with the session key.

This information is given out by the group manager. Each group should have a group manager, but ensuring this in a distributed system is difficult and so the presence of a group manager cannot be ensured at all times.

Finding the group manager is accomplished through either configuration (for secure unicast groups) or through multicast. In both cases it is done with the Request Group command. This command assumes knowledge of KEK_{GM}, and that KEK_{GM} must match KEK_{MEM}. The response will include the group state necessary for the node to communicate with the group, although additional node state may still required before encrypted datagrams can be sent.

There are two possible outcomes when looking for the group manager:

- A valid response arrives. If this occurs then the node becomes a group member as defined below. There may also be multiple valid responses, in which case the response with the highest priority should be used.
- No response arrives. The node should retry the request in order to increase the probability that the group manager receives the command and that a response can be received by the sender.

If no response arrives then the node must make a decision about what course of action to take. This determination is based on several factors, some of which may not be known by the implementation and are therefore up to the application. However, for secure unicast groups a failure to communicate with the group manager is a permanent failure. For secure multicast groups the node may continue as described here.

The node may be purely a requestor, and provide nothing. In this case the node may not care if there are any members in the group (there would be no providers in this case) and so it may decide to try to join the group again later. Providers should be willing to be group managers.



Act as group manager if there are no group managers found. `dof-2008-3-spec-6`

With the exception of secure unicast group members and resource constrained requestor-only nodes, all secure multicast group members should be willing to act as group manager.

Another example is a resource-constrained node that may not be able to be a group manager. Without some other group manager these nodes cannot join the group, so they would try to join again later.

Most commonly a node will determine that it should become the group manager, as described below.

However, if a node that is joining the group for the first time discovers an existing group manager that has a lower priority than the new node would have as group manager; it treats this as equivalent to receiving a heartbeat with a lower priority and triggers the same logic defined for heartbeats, which results in the node becoming the group manager.

Note that in this case the process of becoming group manager requires $STATE_{MEM}$, which includes some security state that may take some time to determine. The process of becoming group manager cannot begin until this state is determined.

3.10.3. Becoming Group Manager (New Group)

The group manager for a secure unicast group is configured. In this case the configured group manager does not need to attempt to discover another group manager and follows the steps outlined below immediately. In the case of a node that has tried to join a secure multicast group but cannot find a group manager the node may become the group manager.

In either of these cases the node:

- Validates that $EPOCH_{MEM}$ is equal to $EPOCH_{AS}$, indicating that $STATE_{AS}$ has not changed. This logic may be based on the amount of time that the node searched for the group manager, and may be skipped if this time is very short (less than a few seconds). If the state has changed then it may elect to search again for a group manager.
- Promotes its current state (obtained from the AS) to "group manager". For example, $EPOCH_{MEM}$ becomes $EPOCH_{GM}$.
- Creates the group state. It must create the group key K_{GM} (it may choose a *secure* random number or it may use a key distribution command).
- Determines its security state. This is described in the section on DOF Security.
- Marks all supported protocols as inactive and waits for a Request Group to activate them.

The node then continues as group manager, as described next.



Follow the correct process before becoming group manager when none is found. `dof-2008-3-spec-7`

This includes validating $STATE_{AS}$, creating $STATE_{GM}$, and determining security state.

3.10.4. Acting as Group Manager

The group manager's responsibilities are based on the timing parameters obtained from the AS as part of obtaining the KEK, and the type of group that it is managing.

Group managers for unicast groups:

- Validate that $EPOCH_{GM}$ is equal to $EPOCH_{AS}$, indicating that the state has not changed, each $TMIN_{GM}$ seconds using a key distribution protocol like TRP. If it is actively denied the KEK then it immediately leaves the group. If the epoch has changed (meaning that the epoch returned from the AS is different than $EPOCH_{GM}$) then it proceeds as described below. If it cannot contact the AS then it continues assuming that the epoch has not changed, however, if they cannot validate the $EPOCH_{AS}$ within $TMAX_{GM}$, then they must leave the group.
- Use the Rekey command to establish a new K_{GM} for the group if the security mode requests it. It may use its own *secure* random number generator or use a key distribution command to determine the new group session key. This would take place, for example, if the security mode were running low on available slots in the case of CCM.
- Proxy key distribution commands from other nodes to its configured authentication server. This allows the group manager to act as the authentication server for its group members. The configured group manager can be a single point of contact and the members do not require their own AS access.

Correctly handle group manager responsibilities in secure unicast groups. `dof-2008-3-spec-8`

This includes validating $STATE_{AS}$, managing $STATE_{GM}$ and acting as a proxy for TRP commands.

Multicast group managers check the epoch and handle rekey requests in the same way as unicast group managers. They are not required to proxy TRP. *In addition* they:

- Send a Heartbeat command each THB_{GM} seconds for each active protocol.
- Use the Request Group command after each epoch validation (described above) to determine if there are other active group managers for the same group. If it receives a valid response from a group manager with a higher priority than its own then it follows the behavior for merging a group described below. Other responses are ignored.
- Listen for Heartbeat commands. If it receives a heartbeat from a node with a higher priority than its own then it immediately stops acting as group manager and continues as a normal group member. Heartbeats with a lower priority cause a Heartbeat to be sent to the group. Heartbeats received of equal priority are resolved by using the randomized tie break discussed above.

Correctly handle group manager responsibilities in secure multicast groups. `dof-2008-3-spec-9`

This includes validating $STATE_{AS}$, managing $STATE_{GM}$, using SGMP_130_Request_Group to check for other group managers, and sending and listening for SGMP_130_Heartbeat commands.

3.10.5. Merging a Group

This section only applies to multicast group managers.

If a group manager determines that it is no longer the group manager by receiving a valid Request Group response from another group manager with a higher priority, then it should use the Rekey Merge command to change the session key of the group that it manages to the newly distributed key (K_{GM} from the responding group manager). The rekey command should be retried enough times to ensure a good chance that all group members will hear it. Further, this rekey is special in that it invalidates any negotiated security mode of operation state of the members of the group as documented in the Rekey Merge command. This includes the sender's own security mode of operation state (but not before finishing its rekey retries).

If the group manager that is merging its group is aware that invalidating the state will require communication with the new group manager (due to its own understanding of its security mode – which is shared by all group members), it may optimize the process by sending a Heartbeat to the group manager of the surviving group. This may be sent unicast, and will trigger the surviving group manager to send its own Heartbeat.

Merge partitioned groups as a group manager when a higher priority manager is found. `dof-2008-3-spec-10`

This requires the use of the SGMP_130_Rekey_Merge command and possibly maintaining multiple sets of state for a brief period while retries are sent.

3.10.6. Handling Updates to $STATE_{AS}$

Nodes (normal group members and group managers) determine that the epoch has changed either directly (through a key distribution response) or indirectly (through the Rekey Epoch command). If they hear about the change indirectly, then they always validate that the epoch really changed, and obtain updated AS parameters, by using a key distribution protocol command. A node that is not the group manager that determines that the epoch has changed directly must notify the group manager by sending a unicast Epoch Changed command to the group manager.



Check for changes to $STATE_{AS}$ at the correct frequency. `dof-2008-3-spec-11`

Once $STATE_{AS}$ has been obtained it must be validated at either $TMIN_{GM}$ or $TMAX_{MEM}$ according the current role of the node. The node must be prepared for these parameters to change whenever $EPOCH_{AS}$ changes.



Send Epoch Changed commands as appropriate. `dof-2008-3-spec-12`

These are sent by members of secure unicast and multicast groups upon learning that $STATE_{AS}$ has changed to an epoch that is later than the latest one used by the group manager (for example, in a SGMP_130_Rekey_Epoch command), and by the group manager to notify the group of an epoch change.

If a group manager indirectly hears about a new epoch through the Epoch Changed command then it validates its AS parameters using a key distribution protocol command as soon as possible. If the epoch has changed then it should begin managing a rekey of the group. This rekey process (sending of Epoch Changed and Rekey Epoch) not only establishes a new key but it also signals the nodes in the group of the change in epoch. This rekey must be protected with the updated KEK_{GM} so other group members will know that they must obtain KEK_{AS} before they can extract the new session key. The group manager must also be prepared to allow new group members to join with the new KEK_{GM}.



As group manager, manage the rekey process when $STATE_{AS}$ changes. `dof-2008-3-spec-13`

This process is triggered by a valid response to a key distribution command as part of obtaining the KEK. The process involves sending an SGMP_130_Epoch_Changed and SGMP_130_Rekey_Epoch commands (including retries). The SGMP_130_Epoch_Changed should be resent during the specified time frame including at least right before the new session key becomes active.

A group manager may also receive a Request Group command that references an epoch that is different than $EPOCH_{GM}$. How the group manager handles these cases depends on implementation. If the epoch in the request is greater than $EPOCH_{GM}$, it should not naively attempt to immediately update its AS parameters because this facilitates a method of DOS attack on the AS. It may, however, decide to check sooner than it might normally in order to provide a response to the (potential) new node. Based on the epoch in the Request Group command it is also possible that the epoch referenced is not known to the group manager and is also older than $EPOCH_{AS}$. In this case there is no way to answer the requesting node. However, the new node should determine that it has old parameters when it attempts to become group manager. An alternate implementation would be for the group manager to keep a short list (based on time) of prior epoch's parameters in order to answer these queries. Care must be taken to prune this list, and parameters older than a few seconds should not be allowed.

As group manager, allow nodes to join with the most recent $STATE_{AS}$. `dof-2008-3-spec-14`

The node must be careful to intelligently use the $EPOCH_{MEM}$ field of the SGMP_130_Request_Group command to trigger validation of $STATE_{AS}$.

A group member that determines that there is a new epoch will notify the group manager using the Epoch Changed command (unicast) and wait for a Rekey Epoch command that references the new epoch. The group manager should, in turn, use the Epoch Changed command to notify the group members of the new epoch and given them adequate time to determine $EPOCH_{MEM}$. However, if a node receives a Rekey Epoch command that includes an $EPOCH_{GM}$ that is greater than $EPOCH_{MEM}$ then it should randomly determine a time to obtain $STATE_{AS}$ before the scheduled rekey. This logic, and the combination of Epoch Changed and Rekey Epoch, gives the group manager control over the rekey timing. Note that the $TMIN_{AS}$ parameter is ignored except by the group manager.

A node that becomes a group manager during these transitions must keep track of the fact that a new epoch exists, and be prepared to either continue with any scheduled rekey (potentially by retrying it to ensure that all nodes know about it) or to initiate the rekey if the previous group manager did not. There is at most one rekey involving a new epoch at a time, and it is the one with the highest $EPOCH_{GM}$ value. This does not mean that there may not be other rekeys that do not involve a new epoch.

As a new group manager, continue or establish required epoch rekey operations. `dof-2008-3-spec-15`

Managing this state is only required on nodes that can become the group manager.

The rekey operation, when it occurs, updates $STATE_{MEM}$, including the encryption mode and other parameters distributed from the AS to the group. For example, changes to the encryption mode or key size take effect when the next epoch becomes effective.

Correctly update all node state when new epochs become effective. `dof-2008-3-spec-16`

This can include a requirement to have two security modes of operation, one that is currently in use and another that will be used with the new epoch. Epochs become effective either on joining a group or at the time defined by SGMP_130_Rekey_Epoch. All of $STATE_{MEM}$ is updated for the new epoch.

3.10.7. Acting as a Normal Group Member

Normal group members:

- Validate that $EPOCH_{MEM}$ is equal to $EPOCH_{AS}$, indicating that $STATE_{AS}$ has not changed, each $TMAX_{MEM}$ seconds using a key distribution command. If the node is actively denied the KEK then it immediately leaves the group. If the epoch has changed (meaning that the epoch returned from the AS is different than $EPOCH_{MEM}$) then it proceeds as described above by sending a unicast Epoch Changed command to the group manager. If it cannot contact the AS then it continues assuming that the epoch has not changed, however, if they cannot validate the $EPOCH_{AS}$ within $TMAX_{MEM}$, then they must leave the group. This means that they may initiate their request for $EPOCH_{AS}$ at some point before $TMAX_{MEM}$ in order to allow for retries.

- Listens for Heartbeat commands if a member of a secure multicast group. If more than one and one-half of THB_{MEM} seconds elapse without hearing a heartbeat the node continues as described next in becoming group manager. For example, if THB_{MEM} is 60 seconds then it would wait for 60*1.5 or 90 seconds. This is only required if the node can become a group manager, however, a node that does not hear a heartbeat for the group for three THB_{MEM} periods in a row must assume the group has been terminated and immediately leave.

- Listens for all types of rekey commands and responds appropriately.

As a secure multicast group member, listen for and respond to missing heartbeats. `dof-2008-3-spec-17`

This requires keeping track of the time since the last valid heartbeat.

As a secure multicast group member, listen for and respond to rekey commands. `dof-2008-3-spec-18`

The specific rules are outlined in the rekey command reference.

3.10.8. Becoming Group Manager (Existing Group)

If a member of a multicast group fails to hear a heartbeat from the group manager as described above, or if it hears a valid heartbeat from a group manager that has a priority less that its own, then it decides if it desires to become group manager. If it does not desire to be a group manager then it can wait to hear if another node becomes the manager. However, if it does not desire to become manager then it should leave the group. This means it should notify the implementation that it is no longer a member of the group, and stop trying to communicate or receive communications from the group. The application may begin the process of joining the group again later. If it desires to potentially become the group manager then it picks a random time in the next one-half THB_{MEM} seconds. At that point, unless it has heard a heartbeat with an equal or higher priority from another group manager, it starts the process of becoming the group manager.

Nodes with higher priority must become group manager. `dof-2008-3-spec-19`

There are two instances where a node that is willing to be a group manager can determine that the current group manager has a lower priority: after receiving a SGMP_130_Request_Group response and after receiving a SGMP_130_Heartbeat. Both cases should trigger a random delay followed by the process of becoming group manager as defined.

That process involves the following steps, taken at the random time picked above:

- If no heartbeat was heard, attempting a Request Group. If it receives a valid response then it remains a normal group member. This step is not required in the case of hearing a heartbeat with a lower priority. The step exists to trigger an existing group manager that has stopped heartbeats of a lower protocol version to again send heartbeats.

- If the request fails or is not required then it sends a Heartbeat command. At this point it begins acting as the group manager.

- It validates the epoch using a key distribution command. It updates $STATE_{GM}$ based on the response.

- It begins (or continues) the process of an appropriate rekey. This will not invalidate security mode of operation state but will change K_{GM}. This should be scheduled as quickly as possible, although in the case of hearing a lower priority heartbeat some delay is acceptable if the other group manager has not yet started a rekey. It does this because the failure of the existing group manager may indicate a network partition, which may cause state collisions since there is no way to validate unique mode of operation state. During the time between the failure and the rekey done by the new group manager there is the possibility of state collisions.



Take over as group manager if required. `dof-2008-3-spec-20`

This must be done at a random time as described and include attempting to join the existing group. If no response is obtained then a heartbeat is sent. Appropriate rekeys must be continued or started.

During this process the node must ignore group managers that have a lower priority than it will have. This prevents a newly joining node from becoming group manager, introducing a bias toward group managers that possess the shared state of the group over potential (new) members of the group that do not yet share that state.

Note that if it determines that that there is a new epoch (or if it already knows there is) then it must also schedule a KEK rekey if one is not already scheduled (in addition to a normal rekey).

3.11. Sessions

SGMP is responsible for establishing and maintaining N-node sessions. This can cause confusion with regards to the security requirements of the different PDUs. This is because some of the PDUs are used to establish a session (or join an existing N-node session), and other PDUs are used to maintain membership in the session.

Typically, when used with security requirements, "session" is used to indicate that the security of the PDU is left to the session (whether or not that session is secure or not). However, for this protocol the word "session" is used to denote only the secure session (when "session" is used with security). PDUs that are allowed both outside the session and inside the session will indicate it.

4. Command Reference

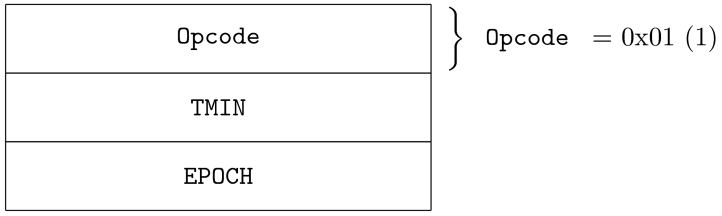
The Secure Group Management Protocol consists of messages sent between a requesting node and nodes in a secure group or a configured group manager. This section describes the commands and responses that make up the SGMP.

4.1. Epoch Changed

Notifies the secure multicast or unicast group manager, or group members that the epoch has changed.
Session: Lossy (n-node)
Addressing: Unicast, Unicast
Security: Session.
Permission (command): None
Permission (response): N/A

⇄ **Epoch Changed** `dof-2008-3-pdu-3`

TADDR: The entire group, or $\text{GMADDR}_{\text{MEM}}$.
TYPE: Either broadcast (to the group) or unicast (to the group manager).
COMMAND/DIRECTED
NOOPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`



`Opcode` **One byte.**

`TMIN` **Two bytes, MSB.** The number of minutes after which a rekey should occur.

`EPOCH` **Two bytes, MSB.** Either $\text{EPOCH}_{\text{MEM}}$ or EPOCH_{GM}.

4.1.1. Flows

The following flows are allowed for this command:

Epoch Changed() $\rightarrow$ (none) : The epoch has changed (send unicast to the manager).

Epoch Changed() $\rightarrow$ (none) : The epoch has changed (send multicast or unicast to the group).

4.1.2. Details

The Epoch Changed command is used to both notify the group manager that the epoch has changed and to notify the group that the epoch has changed. It can be sent to the group manager by any node in the group.

It can be sent to the group only by the group manager. If a group member receives this command from any node other than the group manager then it must silently ignore it.

Each node is periodically contacting the AS, each $TMAX_{MEM}$ seconds. If the newly obtained epoch $EPOCH_{AS}$ is greater than $EPOCH_{MEM}$, then this command should be unicast to the group manager at $GMADDR_{MEM}$. In this case the **TMIN** field should contain zero (0), but must be ignored. The command should use $EPOCH_{MEM}$ of the sending node for **EPOCH**. The group manager will then manage the required rekey.

If this command is received by the group manager, and if **EPOCH** is greater than $EPOCH_{GM}$, then the group manager should validate $STATE_{GM}$. If there is in fact a new validated epoch then a rekey must be initiated by the group manager as described elsewhere.

The first step in the rekey process, which can take time because all group members must obtain the new $STATE_{MEM}$, is for the group manager to multicast the Epoch Changed command to the group. In this case the command will include $EPOCH_{GM}$ and $TMIN_{AS}$.

When a group member receives this command from the group manager, and if **EPOCH** is greater than $EPOCH_{MEM}$, it picks a random time within **TMIN**. At that time, it will contact the AS to obtain updated $STATE_{MEM}$. This new time takes the place of any previously determined time based on previous Epoch Changed commands. This randomization is done in order to avoid flooding the AS with requests.

Once the group members have had time to obtain the new state, the group manager will initiate a Rekey Epoch. There is an implied relationship between this command and an impending rekey, although the operations are logically distinct.

4.1.3. Routing Rules

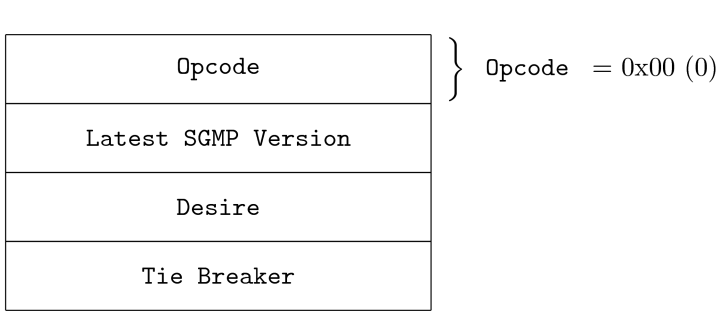
This command must not route.

4.2. Heartbeat

Notifies multicast group members of the presence of a group manager.
Session: Lossy (n-node)
Addressing: Unicast, Unicast
Security: Session.
Permission (command): None
Permission (response): N/A

⇄ **Heartbeat** `dof-2008-3-pdu-4`

COMMAND/DIRECTED
NOOPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`



Opcode	**One byte.**
Latest SGMP Version	**Instance of** Compressed Unsigned 16-Bit$_{\texttt{dof-2009-1-pdu-1}}$.The APPID of the highest SGMP version understood by the sending node.
Desire	**One byte.** The sending node's desire of being group manager as discussed in the specification.
Tie Breaker	**Four bytes, MSB.** A random number chosen when a group manager becomes group manager as discussed in the specification.

4.2.1. Flows

The following flows are allowed for this command:

Heartbeat() $\rightarrow$ (none) : Group manager is operational.

4.2.2. Details

This command is used at least once every THB_{GM} seconds by the multicast group manager. It is not used by unicast managers. In secure multicast groups it signals other members of the group that the group manager is functioning and synchronizes security mode state. This command is typically sent multicast. During the process of merging a group, the group manager that is stepping down (merging its group) may send this command unicast to the group manager of the surviving group in order to trigger his sending a Heartbeat. In this unicast case the sending group manager (that is stepping down) sends his **Tie Breaker** value as zero. This will force the receiving node to remain as group manager.

If the group manager of a multicast group cannot, for some reason, send the Heartbeat command for more than one and one-half THB_{GM} periods then it should step down from being group manager (other nodes use the same timeout to begin the process of establishing themselves as group manager). Note that such a failure would usually result in the node not being able to communicate at all, which is a serious failure.

The transport address of the current group manager must also be determined from this command, which is used for the Epoch Changed command. This is the $GMADDR_{MEM}$ value, and should be obtained from lower layers of the protocol stack for multicast group members. This address is configured for unicast group members.

Members of a secure multicast group that do not receive a heartbeat within one and one-half THB_{MEM} periods of the last heartbeat should start the process of becoming the group manager as described above. Nodes are free to include other factors into this random determination including their ability (or desire) to

act as group manager, but at worst case they must be willing to be group manager or leave the group (as discussed earlier). They should not use a purely non-random determination as this could cause many nodes to begin acting as group manager simultaneously. This problem would resolve itself, but it is not desirable. Nodes that do not (or cannot) act as group manager and do not hear a heartbeat for three THB_{MEM} periods must leave the group.

At that point in time determined by the node it will begin the process of becoming group manager. If, however, it receives a heartbeat from another node that has an equal or higher priority than it does before its chosen time then it assumes that node is the group manager and cancels its own process of becoming group manager.

If it has not determined another group manager has started then it follows the process outlined in the behavior section, above, for becoming a group manager for an existing group. This includes sending a heartbeat, validating the KEK, and rekeying the group.

Nodes that receive heartbeats with a priority lower than their own also begin the process of becoming group manager. Nodes that are already group manager and receive a heartbeat that does not cause them to step down should immediately send a multicast heartbeat to their group.

4.2.3. Routing Rules

This command must not route.

4.2.4. Multiple Versions

At any time, if a multicast group manager receives a Heartbeat from a node with a higher priority than it has then it ceases to be the group manager. Part of the priority is the highest version spoken by the sender, and so group managers that speak more SGMP versions are given priority.

As a multicast group manager, Heartbeats must be sent over each active version of the protocol. A version becomes active for $TMAX_{GM}$ once a valid request to join the group using that version is responded to by a group manager. During this time the group should be maintained using that version. After $TMAX_{GM}$ with no additional join requests has passed, then the version becomes inactive and the group no longer needs to be maintained on that version.

If there are still nodes using that version then they will sense the lack of Heartbeat commands, and one will attempt to become the group manager. Part of that process is an attempt to join the group, which (if there is another group manager that has inactivated that version) will re-enable that version and cause a Heartbeat.

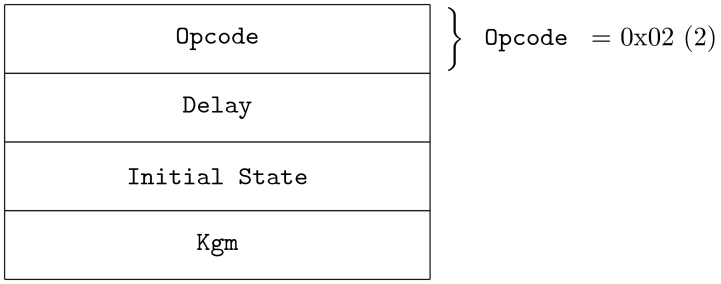
Unicast group managers must maintain transport information for each current member, including version information. The logic described for Heartbeat and Request Group does not apply to unicast group managers.

4.3. Rekey

Establish a new key for a variety of reasons.
Session: Lossy (n-node)
Addressing: Unicast, Unicast
Security: Session.
Permission (command): None
Permission (response): N/A

Rekey `dof-2008-3-pdu-5`

TADDR: The entire group, or group members individually.
TYPE: Multicast or unicast.
COMMAND/DIRECTED
NOOPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`



`Opcode` **One byte.**

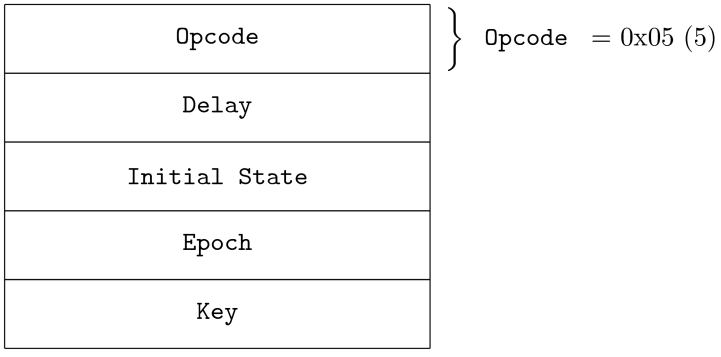
`Delay` **One byte.** The number of seconds until the rekey becomes effective. This may be zero, and can be a maximum of 15.

`Initial State` **Instance of** Security Mode of Operation Initialization`dof-2008-16-pdu-12`.The initial state, which is understood by the security mode of operation.

`Kgm` **Thirty two bytes, MSB.** The new session key.

Rekey Epoch `dof-2008-3-pdu-6`

TADDR: The entire group, or group members individually.
TYPE: Multicast or unicast.
COMMAND/DIRECTED
NOOPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`

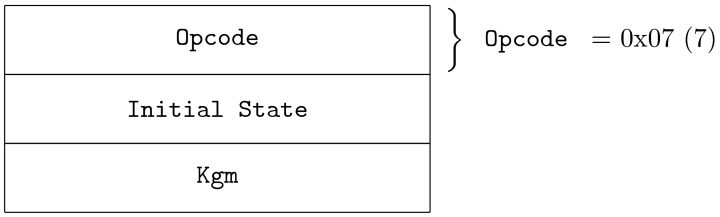


`Opcode` **One byte.**

`Delay`	**One byte.** The number of seconds until the rekey becomes effective. This may be zero, and can be a maximum of 15.
`Initial State`	**Instance of** Security Mode of Operation Initialization`dof-2008-16-pdu-12`.The initial state, which is understood by the security mode of operation.
`Epoch`	**Two bytes, MSB.** The epoch associated with the new key.
`Key`	**Thirty two bytes, MSB.** The new key, passed as K_{GM} xor KEK_{AS}.

⇄ **Rekey Merge** `dof-2008-3-pdu-7`

TADDR: The entire group.
TYPE: Multicast.
COMMAND/DIRECTED
NOOPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`



`Opcode`	**One byte.**
`Initial State`	**Instance of** Security Mode of Operation Initialization`dof-2008-16-pdu-12`.The initial state, which is understood by the security mode of operation.
`Kgm`	**Thirty two bytes, MSB.** The new session key.

4.3.1. Flows

The following flows are allowed for these commands:

Rekey $\rightarrow$ (none) : The key change is scheduled.

Rekey Merge $\rightarrow$ (none) : The key change takes place immediately and receiver's security mode state is flagged as possibly invalid.

Rekey Epoch $\rightarrow$ (none) : The key change is scheduled based on the new epoch.

4.3.2. Details

The rekey commands are only used by the group manager and then depending on the situation. Rekey is used as defined in the specification and as required by the security mode of operation. It can be scheduled in the future. The Rekey Merge is used when a group manager merges its group with another as defined in the specification. This rekey invalidates security mode state and takes place immediately. The Rekey Epoch

is similar to a normal Rekey, but indicates that a new epoch exists. Rekey Epoch protects the distributed session key with the new KEK.

In the case of Rekey Merge some of the state that is distributed to the receiving nodes may have been delivered directly to an existing group manager. In this case the security state may not be applicable to all nodes of the group.

In all cases, security state that is distributed to all nodes in the group needs to be applicable for all receiving nodes.

Since the acting group manager is the only node who should rekey the group, the group manager ignores all incoming rekey commands.

Rekey and Rekey Epoch include a delay after which the change takes place. This is always with reference to the time that the command is received. For Rekey Merge the change takes place immediately, and the key and state are updated as soon as the datagram is received. This is different than Rekey with a zero delay because of the additional invalidation of node state information that is assigned by the security mode of operation.

The allowable delay has an upper limit of THB_{MEM}. If a longer delay is received then THB_{MEM} should be used. This ensures that rekey operations will occur in less time than a new group manager can become active, ensuring in normal circumstances that management of a rekey operation will not need to pass from one manager to another.

In the case of a non-epoch rekey (Rekey and Rekey Merge) the new key is sent directly as part of the command.

In the case of an epoch rekey (Rekey Epoch) the new key is sent exclusive-or'ed with the KEK that corresponds to the declared epoch. If the receiving node is not aware of the KEK for that epoch then it must go to the AS to obtain it. This must occur before the delay, although the exact timing must be random so as not to overwhelm the AS. If the node is not able to obtain the correct KEK then it should leave the group, as it is unable to obtain the new group key and will shortly be unable to communicate with the group.

There are situations when multiple rekeys are attempted that overlap. The rules are:

- A single Rekey Epoch can be scheduled. If another arrives with a larger epoch then it takes priority over the existing scheduled one. However, a new group manager sending a Rekey Epoch is allowed to override an existing Rekey Epoch even in the same epoch.
- A single normal Rekey can be scheduled. If another arrives with a smaller delay then it replaces the existing scheduled one.
- Rekey Merge takes effect immediately and cancels all other rekeys, including Rekey Epoch.
- When a Rekey Epoch becomes effective, all other rekeys are cancelled. ## 1. Routing Rules

This command is not routed.

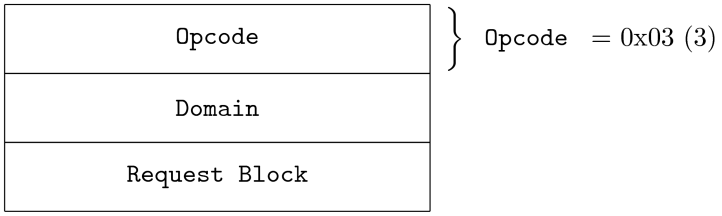
4.4. Request Group

Request a group key from a group manager.
Session: Lossy (n-node)
Addressing: Unicast, Unicast
Security: Session.
or
None.
Permission (command): None

Permission (response): None

⇄ Request Group (command) `dof-2008-3-pdu-8`

TADDR: The entire group or the group manager if known.
TYPE: Multicast or unicast.
COMMAND/DIRECTED
OPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`

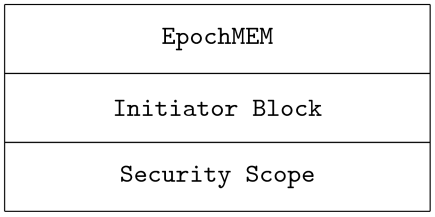


`Opcode` **One byte.**

`Domain` **Instance of** Security Domain`dof-2008-16-pdu-5`.The security domain of the group.

`Request Block` **Instance of** Request Group Request Block`dof-2008-3-pdu-9`.Information that identifies the request and is part of the cryptography described in the specification.

⇄ Request Group Request Block `dof-2008-3-pdu-9`



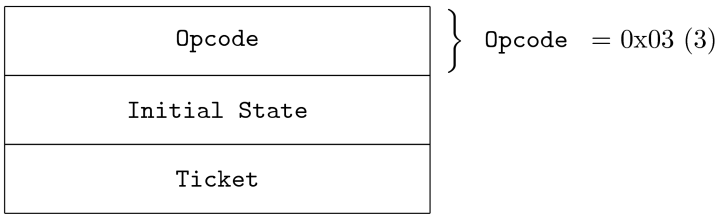
`EpochMEM` **Two bytes, MSB.** The epoch associated with the KEK obtained from the AS.

`Initiator Block` **Instance of** SGMP Key Request`dof-2008-3-pdu-12`.A block of information representing the initiator. Restrictions on its contents are listed below.

`Security Scope` **Instance of** Security Scope`dof-2008-16-pdu-6`.The security scope of the node, which can be used by the group manager to limit flooded operations.

⇄ Request Group (response) `dof-2008-3-pdu-10`

RESPONSE
OPID
Instance of SGMP Command/Response Format`dof-2008-3-pdu-2`

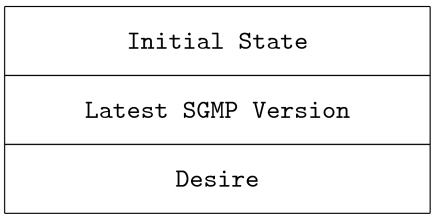


`Opcode` **One byte.**

`Initial State` **Instance of** Request Group Request Block`dof-2008-3-pdu-9`.The initial state distributed by the sender.

`Ticket` **Instance of** Key Grant`dof-2008-16-pdu-14`.A block representing the distributed group key K_{GM}.

⇄ Request Group Initialization Block `dof-2008-3-pdu-11`



`Initial State` **Instance of** Security Mode of Operation Initialization`dof-2008-16-pdu-12`.The initial security state distributed by the sender.

`Latest SGMP Version` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The APPID of the highest SGMP version understood by the sending node.

`Desire` **One byte.** The sending node's desire of being group manager as discussed above.

4.4.1. Flows

The following flows are allowed for this command:

Request Group() $\rightarrow$ response() : Success.

Request Group() $\rightarrow$ (none) : Failure to join the group.

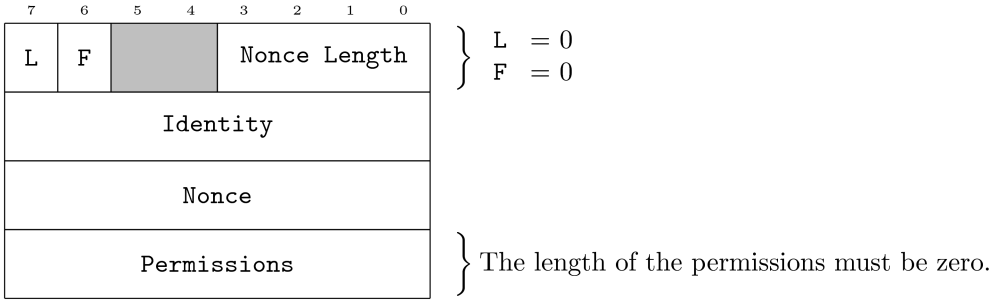
4.4.2. Details

The Request Group command is used after obtaining KEK_{AS} from the AS using a key distribution response in order to join the group. Knowledge of the KEK is required for this command in order to obtain the group key from the response.

The Initiator Block in the command must follow the specified format that follows.

SGMP Key Request `dof-2008-3-pdu-12`

Instance of Key Request`dof-2008-16-pdu-13`



`L` **One bit.** Set to indicate a new authentication.

`F` **One bit.** Set if the request represents a fixed permission set. A node that requests a fixed permission set should not attempt to renegotiate its permissions. This means that the other node can optimize its own permission requests based on the current state of the other node's permissions.

`Nonce Length` **Four bits.** Indicates the length of the `Nonce Length` field.

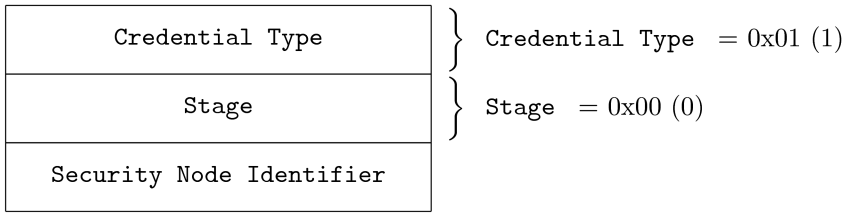
`Identity` **Instance of** SGMP Identity`dof-2008-3-pdu-13`.The identity of the node requesting authentication.

`Nonce` **Variable length of {{ln.name}}+1 bytes.** This number uniquely identifies this request for authentication.

`Permissions` **Instance of** Permission Request`dof-2008-16-pdu-8`.The permissions requested by the node. The list must be empty (no elements).

SGMP Identity `dof-2008-3-pdu-13`

Instance of Base Credential Format`dof-2008-16-pdu-2`



`Credential Type` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

`Stage` **One byte.** The stage of the request. The indicated value is required. This value identifies the PDU as not including a length. The length of the **Security Node Identifier** is contained inside itself, any additional length would be redundant.

`Security Node Identifier` **Instance of** Security Node Identifier`dof-2008-16-pdu-1`.Contains the identity as known by the authentication server.

The outputs of this flow are the group key K_{GM} and the current security mode state associated with the key, both of which are required in order to communicate with the other members of the group. Additionally several parameters are present which indicate the priority of the sending group manager.

There are two fundamentally different types of groups that can be created: unicast and multicast. The sending node must understand which of the two types is being joined. It is possible for a single node to simultaneously join both types of groups.

This command is sent multicast for joining a multicast group, even if the group manager is known. The request is sent unicast for joining a unicast group, in which case the group manager's location must be known. The response is always sent unicast.

A node should only attempt to join a group using the latest version of SGMP that it understands.

The handling of this command is based on the status of the receiver in the identified group; if the node is not the group manager then the command is ignored. If the node is the group manager then a response is sent. Note that implementations that support both unicast and multicast membership and unicast management on the same node must use the transport type (multicast, unicast) of the command to pass the request to the appropriate manager.

The response contains the security mode initialization information along with priority information. This comprises $STATE_{GM}$. It is distributed using the two-party authentication method defined in the *DOF Security Specification* using parameters defined below.

The Request Group command should be sent a reasonable number of times in order to ensure that datagram loss does not interfere with obtaining a result.

If no responses are received then the node must assume that there are no other group members and may become the group manager as described in the specification.

In the case that a node receives multiple responses (indicating multiple acting group managers) it should use the response from the group manager with the highest priority. Multiple responses may be received,

for example, during the time that the node is waiting to validate security state or any other delays required by the encryption mode. During this time the highest priority should be saved along with $GMADDR_{MEM}$ so that responses that arrive in the future can have their priorities compared appropriately.

This command identifies the group that is being joined through the identity of the request.

This command may be sent securely in order to update the group manager's node security state information for the node. In this way changes from the AS regarding the node's security scope can be sent to the group manager.

4.4.3. Cryptography

This command uses the two-party authentication method described in the *DOF Security Specification*. The following blocks are used:

Table 4.1. Request Group Cryptography Parameters

Parameter Name	Parameter Contents
D	Security.7: Security Domain, from the command.
I	SGMP.6.1.1: Request Group Request Block, from the command.
A	SGMP.6.2.1: Request Group Initialization Block, from the response.
K_I	KEK_{GM}. The node must have obtained KEK_{GM} previously, as described.
K_D	K_{GM}. This is the current group key.

4.4.4. Routing Rules

This command must not route.

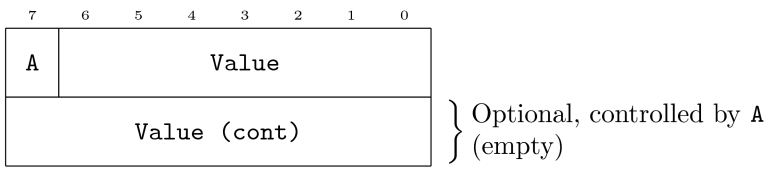
4.4.5. Multiple Versions

If a group manager speaks multiple SGMP versions then it must periodically try to join the group only in the highest protocol version that it speaks. As discussed in the Heartbeat command this join request, if successful, will activate that version and cause heartbeats to be sent.

5. External PDU Reference

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

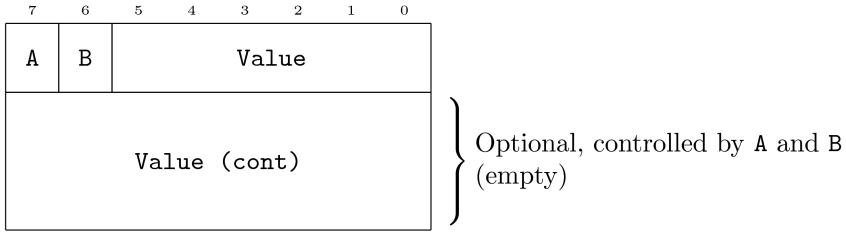
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`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`

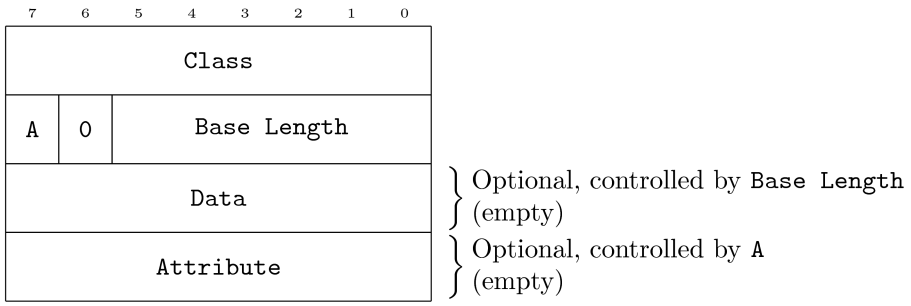


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`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.

Object Identifier (OID) `dof-2009-1-pdu-4`



`Class` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.The OID class, registered with ODP-TSC.

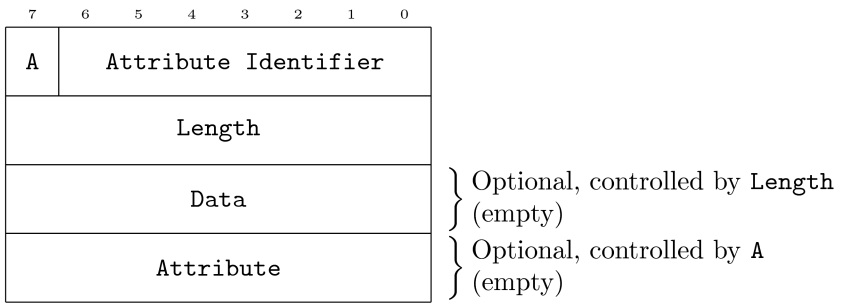
`A` **One bit.** Controls the presence of `Attribute`.

`Base Length` **Six bits.** This contains the length of the `Data`.

`Data` **Variable bytes.** The data associated with the OID. `Base Length` bytes.

`Attribute` **Instance of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `A`. If `A` is one then this field is present. Attributes form an ordered list, with each attribute containing a flag controlling the presence of the next attribute.

Object Identifier Attribute `dof-2009-1-pdu-5`



`A` **One bit.** Controls the presence of `Attribute`.

`Attribute Identifier` **Seven bits.** The attribute type, registered with ODP-TSC.

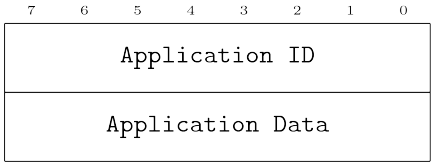
`Length` **One byte.** This field contains the length of the `Data`.

`Data` **Variable bytes.** The data associated with the attribute. `Length` bytes.

`Attribute` **Instance of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `A`. If `A` is one then this field is present. Attributes form an ordered list, with each attribute containing a flag controlling the presence of the next attribute.

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

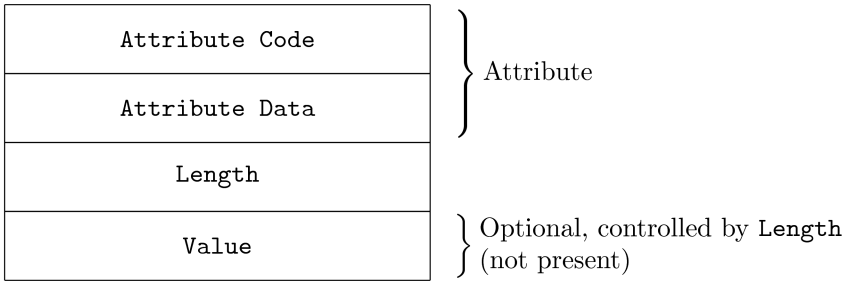
⇄ **General Application Header** `dof-2008-1-pdu-13`



`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.

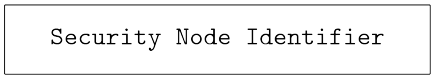
`Length` **One byte.** This field determines the length of the `Value`. It may be zero, in which case the `Value` field is not present.

`Value` **Variable length. Optional, controlled by** `Length`. Contains the value for the associated attribute.

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

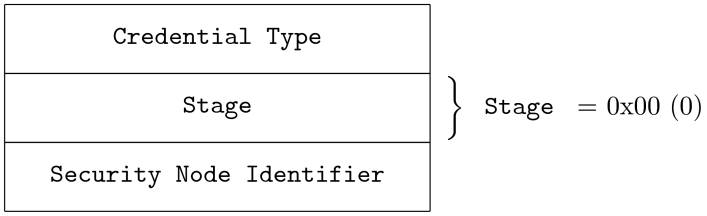
Security Node Identifier `dof-2008-16-pdu-1`

Instance of Object Identifier (OID)`dof-2009-1-pdu-4`



`Security Node Identifier` **Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.This represents the node. It must not have attributes. It must not be broadcast or multicast.

Base Credential Format `dof-2008-16-pdu-2`



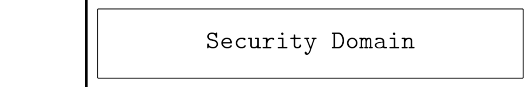
`Credential Type` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

`Stage` **One byte.** The stage of the request. The indicated value is required. This value identifies the PDU as not including a length. The length of the **Security Node Identifier** is contained inside itself, any additional length would be redundant.

`Security Node Identifier` **Instance of** Security Node Identifier`dof-2008-16-pdu-1`.Contains the identity as known by the authentication server.

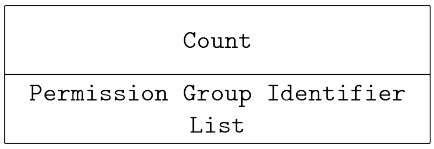
Security Domain `dof-2008-16-pdu-5`

Instance of Object Identifier (OID)`dof-2009-1-pdu-4`



`Security Domain` **Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.This represents the domain. It must not have attributes. It must not be multicast. Certain PDUs allow the use of broadcast for the domain.

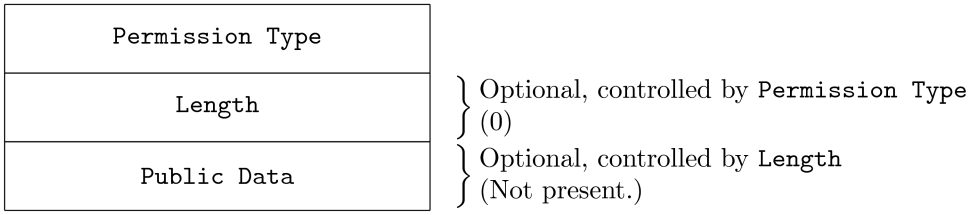
⇄ Security Scope `dof-2008-16-pdu-6`



`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Specifies the length of the **Permission Group Identifiers**).

`Permission Group Identifier List` **Array of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `Count`.

⇄ Permission `dof-2008-16-pdu-7`

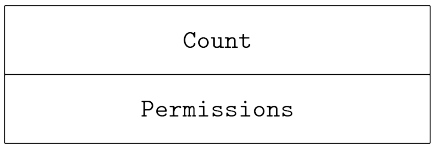


`Permission Type` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by** `Permission Type`. Controls presence of **Data**. Present only if the **Permission Type** is odd. This field controls the size of **Data**.

`Public Data` **Variable length. Optional, controlled by** `Length`. Present only if **Length** is greater than zero. The permission data depends on the **Permission Type**.

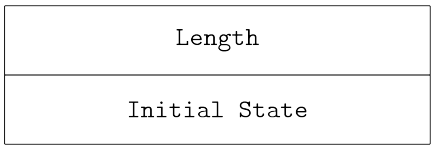
Permission Request `dof-2008-16-pdu-8`



`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls length of **Permission List**.

`Permissions` **Array of** Permission`dof-2008-16-pdu-7`.**Optional, controlled by** `Count`. Defines a set of permissions. The ordering of permissions inside the request does not affect the meaning and must not influence the result. However, the ordering must be stable because the security scope results arrive in the same order as the request.

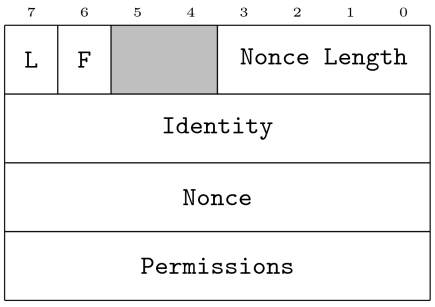
Security Mode of Operation Initialization `dof-2008-16-pdu-12`



`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.

Key Request `dof-2008-16-pdu-13`



`L` **One bit.** Set to indicate a new authentication.

`F` **One bit.** Set if the request represents a fixed permission set. A node that requests a fixed permission set should not attempt to renegotiate its

permissions. This means that the other node can optimize its own permission requests based on the current state of the other node's permissions.

`Nonce Length` **Four bits.** Indicates the length of the `Nonce Length` field.

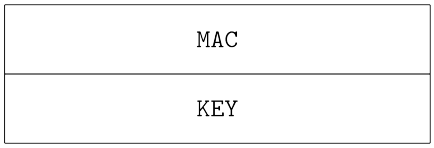
`Identity` **Instance of** Base Credential Format`dof-2008-16-pdu-2`.The identity of the node requesting authentication.

`Nonce` **Variable length of {{ln.name}}+1 bytes.** This number uniquely identifies this request for authentication.

`Permissions` **Instance of** Permission Request`dof-2008-16-pdu-8`.The permissions requested by the node.



Key Grant `dof-2008-16-pdu-14`



`MAC` **Thirty-two bytes.** A hash described in the *DOF Security Specification*.

`KEY` **Thirty-two bytes.** The encrypted MAC field, convoluted with the new session key as described in the *DOF Security Specification*.