Ticket Exchange Protocol (V1)

DOF Protocol Specification

Technical Steering Committee

Ticket Exchange Protocol (V1): DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-4
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	Ticket Exchange Protocol (V1), OpenDOF TSC, [dof-2008-4] (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
    2.1. References 9
3. Specification 11
    3.1. Application Format 11
    3.2. Security Primitives 11
    3.3. Assumptions 12
    3.4. Negotiation 12
        3.4.1. Protocol Options 12
        3.4.2. Node Roles 13
        3.4.3. Security Desires 13
        3.4.4. DSP Behaviors 13
        3.4.5. Rules 13
    3.5. Invalid Opcodes 14
    3.6. Handling Key Expiration 14
    3.7. Handling Permission Extensions 14
4. Command Reference 15
    4.1. Request/Accept/Confirm/Confirm ACK 15
        4.1.1. Flows 17
        4.1.2. Details 17
        4.1.3. Security Considerations 19
        4.1.4. Retry Behavior 20
        4.1.5. Cryptography 20
    4.2. End Session 20
        4.2.1. Details 21
    4.3. Reject 21
        4.3.1. Flows 22
        4.3.2. Details 22
    4.4. Session Ending 23
        4.4.1. Details 23
5. External PDU Reference 25
    5.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 25
    5.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 27
    5.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018) 29

List of Tables

3.1. TEP Negotiation Rules for Clients 13
3.2. TEP Negotiation Rules for Servers 14
4.1. Security State Finalization – Lossy Sessions 19
4.2. Security State Finalization – Lossless Sessions 19
4.3. TEP Request - Cryptographic Parameters 20
4.4. TEP Error Codes 22

Specifications and PDUs

TEP Application Format `dof-2008-4-pdu-1` 11
TEP Command/Response Format `dof-2008-4-pdu-2` 11
DSP Authentication Attribute `dof-2008-4-pdu-3` 12
Handle invalid TEP operations appropriately. `dof-2008-4-spec-1` 14
Request/Confirm `dof-2008-4-pdu-4` 15
Accept/Confirm ACK `dof-2008-4-pdu-5` 16
Session Initialization `dof-2008-4-pdu-6` 17
Nodes may only extend permissions inside a secure session. `dof-2008-4-spec-2` 17
Sessions may have a single Request (K=1) operation at a time. `dof-2008-4-spec-3` 18
Security state expiration terminates sessions. `dof-2008-4-spec-4` 19
End Session `dof-2008-4-pdu-7` 21
Reject `dof-2008-4-pdu-8` 21
Session Ending `dof-2008-4-pdu-9` 23
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 25
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 25
Object Identifier (OID) `dof-2009-1-pdu-4` 26
Object Identifier Attribute `dof-2009-1-pdu-5` 26
General Application Header `dof-2008-1-pdu-13` 27
DSP Command/Response Format `dof-2008-1-pdu-15` 27
Attribute/Value Pair `dof-2008-1-pdu-16` 28
Configuration Negative Acknowledge `dof-2008-1-pdu-19` 28
Configuration Reject `dof-2008-1-pdu-20` 28
Attribute Data List `dof-2008-1-pdu-21` 29
Security Node Identifier `dof-2008-16-pdu-1` 29
Base Credential Format `dof-2008-16-pdu-2` 30
Security Domain `dof-2008-16-pdu-5` 30
Security Scope `dof-2008-16-pdu-6` 30
Permission `dof-2008-16-pdu-7` 31
Permission Request `dof-2008-16-pdu-8` 31
Permission Validation `dof-2008-16-pdu-9` 32
Security Mode of Operation Negotiation `dof-2008-16-pdu-11` 32
Security Mode of Operation Initialization `dof-2008-16-pdu-12` 32
Key Request `dof-2008-16-pdu-13` 33
Key Grant `dof-2008-16-pdu-14` 33
Session Initiator Block `dof-2008-16-pdu-15` 34
Session Responder Block `dof-2008-16-pdu-16` 34
Authentication Response Block `dof-2008-16-pdu-17` 34

1. About This Document

This document defines the Ticket Request Protocol, Version 1 (TEPv1, or TEP). TEP is a security protocol built on the DOF Protocol Stack (DPS). The *DOF Protocol Specification* defines the DPS and identifies how TEP fits into the stack.

This information is highly technical. This document serves as a specification and reference against which someone could implement networked devices that interoperate 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.

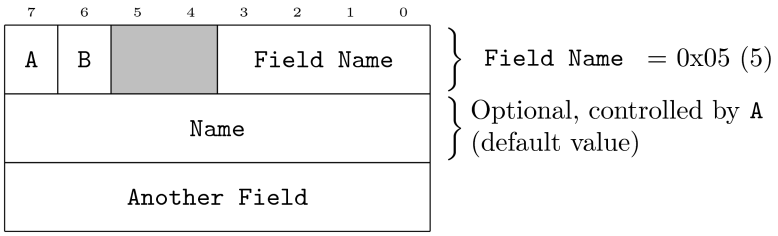
1.3.4. Fields

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 document describes the Ticket Exchange Protocol (TEP). Nodes use the Ticket Exchange Protocol to create an authenticated, secure link on which access control rules are applied. As discussed in the *DOF Security Specification*, TEP uses a three-party authentication system based on information stored in a security domain. The security specification covers the details of communication with the security domain. The security domain (represented by an authentication server, or AS), uses a family of protocols called "Ticket Request Protocols," or TRP, to communicate.

In fact, there is a very close relationship between TEP, TRP, and the AS. In order to fully understand the use of TEP it will help to understand the basics of DOF security as well as TRP.

TEP is an DOF Authentication Protocol, and can be negotiated using DSP to satisfy a security requirement on a session.

The Ticket Exchange Protocol distributes cryptographic tickets that authenticate two nodes with each other, called the initiator and the responder. There are essentially two stages in the Ticket Exchange Protocol.

In the first stage the initiator, which is always the client, sends a Request command to the responder. The responder may then request tickets from the authentication server using TRP, or it can send the TRP ticket request message to the initiator, requesting that the initiator obtain tickets from the authentication server. If the server is going to "reflect" the TRP command then it requests TRP during DSP negotiation.

The second stage, which begins after the receipt of the tickets, distributes the tickets and confirms to each node that the other node successfully processed its ticket. The responder begins this stage by issuing an Accept response to the initiator. This message also contains a confirmation that the responder understood its ticket and initializes its security state. The initiator then sends a Confirm command, which confirms that it understood its ticket. Upon acknowledgement of the confirmation, the two nodes are authenticated to each other and ready to communicate securely.

At times, the initiator may need to renew its security state due to expiration of the current ticket grant. This proceeds in the same way as the initial exchange, but takes place within the secure session that already exists.

Additionally, either the initiator or the responder may need to obtain additional permissions. This permission extension uses the same commands, with some modifications in the handling of the resulting security state.

Note that implementations may start TEP sessions over both lossless and lossy transport sessions, but that TEP requires 2-node sessions. It is also possible to begin multiple TEP sessions using the same "base" session. In these cases, a security state identifier assigned by the server identifies the new TEP session.

2.1. References

Please see the *DOF Security Specification* for information about the cryptography behind TEP. The specific requirements for validating TEP commands and responses are included in the individual command documentation.

3. Specification

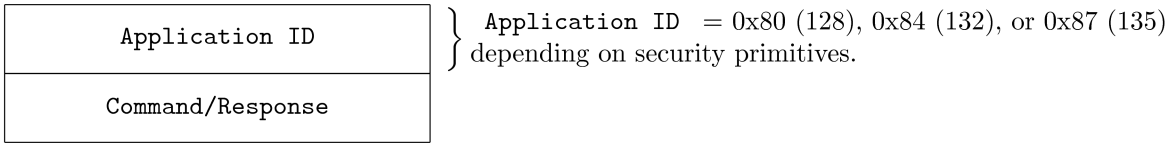
This section contains the specification of the Ticket Exchange Protocol.

3.1. Application Format

TEP uses an application identifier of 0x80 (128), 0x84 (132), or 0x87 (135) based on the security primitives desired. When used in the DPS, the following format is used.

⇄ **TEP Application Format `dof-2008-4-pdu-1`**

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

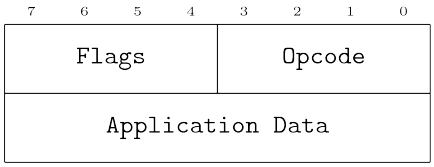


`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Command/Response` **Instance of** TEP Command/Response Format`dof-2008-4-pdu-2`.Contains the command or response data.

Each TEP PDU uses the following format.

⇄ **TEP Command/Response Format `dof-2008-4-pdu-2`**



`Flags` **Four bits.** Each specific **Opcode** defines the use of this field.

`Opcode` **Four bits.** This defines the specific operation.

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

3.2. Security Primitives

This protocol version uses the following security primitives in place of the general primitives defined in the *DOF Security Specification* based on APPID:

APPID	ENC	HASH	MAC
0x80 (128)	AES	SHA-256	HMAC
0x84 (132)	Twofish	SHA-256	HMAC
0x87 (135)	SMS4	SHA-256	HMAC

3.3. Assumptions

This protocol can use either lossless or lossy sessions.

On lossless sessions the protocol does not allow retries and the state transitions are streamlined.

On lossy transports, the individual commands and responses require retries in order to ensure reception of a response (or the next command in sequence). Implementations should consider both their state and the state of the other end of the session in determining desired behavior, and should cache any data necessary to retry the packet efficiently if required.

The implementation determines the specific timing of retries and retransmissions, and it should use knowledge of the transport behavior in determining these rates. The transport likely handles retransmissions directly, while retries are the responsibility of the application implementation.

3.4. Negotiation

TEP differs from other DOF protocols in that only one end of a new session can initiate it. This node is the 'initiator'. The other end of the session is the 'responder'. At the beginning of new session, the initiator is always the transport client. This makes TEP a client/server protocol, with a special requirement that the transport and application client are the same. This requirement is relaxed once the new session is established, and so permission extension (which also uses TEP) is peer-to-peer.

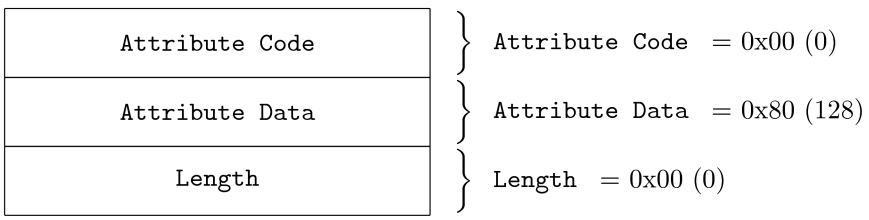
It is always the responsibility of the client to negotiate TEP. The server may suggest that TEP be negotiated, but can only reject the session if it is not.

3.4.1. Protocol Options

The following format is required for negotiating TEP using DSP:

⇄ **DSP Authentication Attribute** `dof-2008-4-pdu-3`

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



`Attribute Code` **One byte.**

`Attribute Data` **Two bytes, MSB.**

Length	**One byte.**

3.4.2. Node Roles

For any session, each node takes the role of client or server. This role corresponds directly with the sense of who is establishing the session (the client), and who provides the ability to establish sessions (the server).

Note that it is very possible for both nodes to be *capable* of both roles, but for any given session, one node is the client and the other is the server. The initial TEP initiator is always the same as the transport client.

In the case of permission extension, either node may take the role of the initiator. The other node will be the responder.

3.4.3. Security Desires

Each node involved in establishing a session has a desire for some type of security. The desire can be one of the following:

- Encrypted: The session must be fully secure (authenticated and encrypted), or it will not be acceptable.
- Authenticated: The messages exchanged must be authentic and use access control, but encryption is not required.
- Not Secure: The session must be unsecured. Note that this is usually because the node is unable to use the required security algorithms. If a node can use security, it should usually be willing to do so.
- Do not Care: It is possible that the node has no preference one way or another. In this case, the desire of the other node is used. Two "Do not Care" nodes will connect without security.

The 'Encrypted' and 'Authenticated' desires are both 'Secure'.

3.4.4. DSP Behaviors

DSP negotiates first client to server and then server to client. The client proposes options first, which the server accepts or rejects. The server then negotiates any addition options.

The client is responsible to negotiate TEP. This means that a server will never include the TEP option in its requests. The server may include the TEP option in responses to the client (in order to suggest that a session be secure).

There are three possible responses to an DSP request: ACK, NAK, or Reject. Each has a different meaning, as discussed in the *DOF Protocol Specification*. The rules discussed in the next section indicate which of these responses is appropriate for each combination of role and security desire.

3.4.5. Rules

The following table identifies the rules that apply for each role and security desire. The rows represent the security desire; the columns contain the details based on the current negotiation state. The first table indicates client behavior; the second table indicates server behavior.

Table 3.1. TEP Negotiation Rules for Clients

Desire	Initial Request	Upon Receiving a NAK with TEP	Upon Receiving a Reject of TEP
Secure	Request TEP.	(not possible)	Close the session.

Desire	Initial Request	Upon Receiving a NAK with TEP	Upon Receiving a Reject of TEP
Not Secure	Do not request TEP.	Close the session.	(not possible)
Don’t Care	Do not request TEP.	Request TEP.	(not possible)

Table 3.2. TEP Negotiation Rules for Servers

Desire	Request Includes TEP	Request Doesn’t Include TEP
Secure	Send Acknowledge.	Send NAK including TEP.
Not Secure	Send Reject.	Send Acknowledge.
Don’t Care	Send Acknowledge.	Send Acknowledge.

Once negotiation is complete, either the session allows TEP or not. This controls the security of the session.

If the client negotiated TEP then it has the responsibility to make the initial TEP request.

3.5. Invalid Opcodes

Handling invalid opcodes depends on the current state of the session.



Handle invalid TEP operations appropriately. `dof-2008-4-spec-1`

Invalid opcodes signal either corrupted communication channels or loss of state. Unsecured, lossy sessions ignore invalid commands. For lossless sessions, whether secured or not, the receipt of an invalid command causes the session to close.

3.6. Handling Key Expiration

One of the responsibilities of the initiator is to request the duration of the session key. It is possible that a session will outlive this duration, and so a new session key will be required. It is always the responsibility of the client/initiator to initiate requests for new session keys.

Because of clock synchronization problems, the server should send the Session Ending command as described in the section on that command. The client can use this command as a signal to initiate a new session key request (send a new Request).

3.7. Handling Permission Extensions

Either node may find that it needs additional permissions on a session. When this occurs then either node may begin the process of extending the permissions with the Request command. In this mode, the system treats the TEP protocol as a peer-to-peer protocol, since the server did not initially negotiate it. In this case, the sending node takes the role of initiator (even though it may not be the client). Failure to extend permissions has no effect on the duration of the session itself.

4. Command Reference

The Ticket Exchange Protocol consists of commands and responses sent between an initiator and a responder. The following sections describe these commands and responses.

The *DOF Security Specification* introduces a common pseudo-language for security primitives. The following security statements apply to TEP:

B?: A # B with perms: $\{P\}_A$, $\{P\}_B$, where a Request command is used with **K** set or clear.

A: A # B as S_1 with perms: $\{P\}_A$, $\{P\}_B$; key: K, where a Request with **K** set completes.

A: A # B as #, where a Request command fails (a Reject is used)

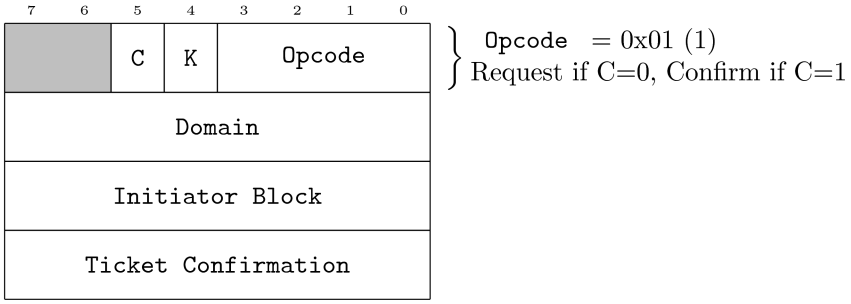
A: A # B continues S_1 adding perms: $\{P_1\}$,$\{P_1^{-1}\}$; ignoring key, where a Request with **K** clear completes.

4.1. Request/Accept/Confirm/Confirm ACK

Requests and confirms a new session key (client/server) or extends permissions (peer-to-peer).
Session: Lossy (2-node), Lossless
Addressing: Unicast
Security: See the Security Considerations section, below.
Permission (command): None
Permission (response): None

⇄ **Request/Confirm** `dof-2008-4-pdu-4`

COMMAND/DIRECTED
OPID
Instance of TEP Command/Response Format`dof-2008-4-pdu-2`



`C` **One bit.** Controls the presence of the **Initiator Block** and **Ticket Confirmation** fields. When it is equal to one the **K** field must be zero.

`K` **One bit.** Controls the presence of the **Domain** field in the command and the **Responder Initialization** fields in the corresponding response. When equal to one, indicates that the resulting security state applies to the session. Otherwise, the request is for authentication only, and the resulting security state

will not apply to the session (although the permissions will). This flag must be zero when **C** is one.

`Opcode` **Four bits.** This defines the specific operation.

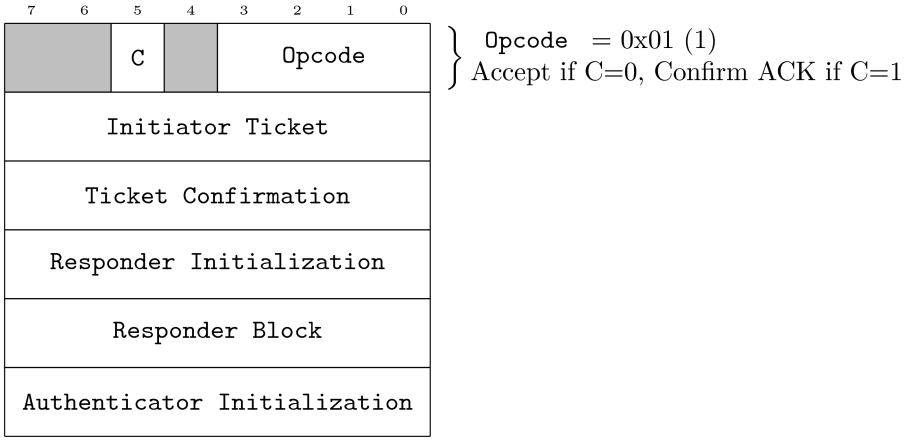
`Domain` **Instance of** Security Domain`dof-2008-16-pdu-5`.**Optional, controlled by** `K`. This field contains the security domain of the request. It is present when the **K** field is one and the command is unsecured.

`Initiator Block` **Instance of** Session Initiator Block`dof-2008-16-pdu-15`.**Optional, controlled by** `C`. Present if **C** is zero. The parameters associated with the request. If **K** is zero then the desired duration field must be zero (0).

`Ticket Confirmation` **Thirty two bytes. Optional, controlled by** `C`. Present if **C** is one. This contains TC_I.

Accept/Confirm ACK `dof-2008-4-pdu-5`

RESPONSE
Instance of TEP Command/Response Format`dof-2008-4-pdu-2`



`C` **One bit.** Controls the presence of all remaining fields.

`Opcode` **Four bits.** This defines the specific operation.

`Initiator Ticket` **Instance of** Key Grant`dof-2008-16-pdu-14`.**Optional, controlled by** `C`. Present if **C** is zero. This contains the block of initiator authentication information.

`Ticket Confirmation` **Thirty two bytes. Optional, controlled by** `C`. Present if **C** is zero. This contains TC_R.

`Responder Initialization` **Instance of** Session Initialization`dof-2008-4-pdu-6`.**Optional, controlled by** `C`. Present if **C** is zero and **K** of the corresponding command is one.

`Responder Block`	**Instance of** Session Responder Block`dof-2008-16-pdu-16`.**Optional, controlled by** `C`. Present if **C** is zero. A block representing the information passed to the authentication server by the responder.
`Authenticator Initialization`	**Instance of** Authentication Response Block`dof-2008-16-pdu-17`.**Optional, controlled by** `C`. Present if **C** is zero.

Session Initialization `dof-2008-4-pdu-6`

State Identifier
Initial State

`State Identifier`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.Present for unsecure responses. This field contains the assigned security state identifier for the new session. If possible a value of zero is used, which matches the default value in DPP.
`Initial State`	**Instance of** Security Mode of Operation Initialization`dof-2008-16-pdu-12`.This block contains the initial security state distributed by the sender.

4.1.1. Flows

Request() → Accept(), Confirm() → Confirm ACK: Success

Request() → Accept(), Confirm() → TEP No-op: Success on a new session

Request() → Reject(): Failure

4.1.2. Details

Nodes use this command sequence in the following cases:

- The client uses it to establish a new session with a server. In this case, the client makes an unsecure request. The **C** field is zero and **K** field is one. The resulting session uses the distributed security state to secure the session.

Nodes may only extend permissions inside a secure session. `dof-2008-4-spec-2`

This means that both **C** and **K** may only be zero at the same time inside a secure session.

- The client uses it to extend an existing session. In this case, the client makes a secure request using an existing session. The **C** field is zero and **K** field is one. The session continues securely but transitions to the new security state once confirmed.

- The client or the server uses it to augment the permissions established on a session. In this case, the initiator makes the request securely using the session that needs the permissions. Both the **C** and **K** fields are zero. The session ignores the distributed security state after confirmation.
- The initiator uses it to confirm a Request operation. In this case, the command may be secured or unsecured and must match that of the corresponding Request, but the **C** field is one and the **K** flag is zero.

In these cases, the node sending the Request command is the initiator. The other node is the responder.

If a node sends this command non-securely, then it indicates a request for a new session. If a node sends it using an existing session then it indicates a change of security state for that existing session (including permissions).

When used to confirm the session (**C** field is one and the **K** field is zero) the command is a retry of the already existing Request operation. The initiator takes this step after receiving a response (Accept) to the initial Request. This second stage completes on the receipt of a Confirm ACK (response with the **C** field one – except when establishing a new connection as discussed below). On lossy sessions, care is required to be able to retry the Confirm until receipt of the required response (Confirm ACK or TEP no-op).

This command sequence may also used to modify the permission set of an existing session. Both the initiator and responder may use this capability once a session is established. The handling of this command with **K** equal to zero (the case of extending permissions) is identical to the handling with **K** equal to one except:

- The **Desired Duration** of Session Initiator Block must be zero.
- The **Desired Security Mode** of Session Initiator Block must match that of the existing session.
- The command does not modify the duration of the established session.
- The response does not include any security state information (the S block is empty).
- The session ignores the distributed security state once the sequence is complete.
- Authenticated permissions *augment* existing permissions.

A command sequence with **K** equal to one replaces the existing permissions of the session with those from the new security state.

In order to send an Accept, the responder will interact with an authentication server to validate and respond to the Request. The initiator then sends a Confirm command to finalize session startup. Note that the Confirm is a presentation layer retry of the same operation.

The responder may in fact use TRP provided by the initiator to communicate with the AS. This is possible if the responder has no AS session and the initiator allowed TRP during DSP negotiation.

Note that the use of this command to extend a session (a secure Request with **K** equal to one) is exclusive. This means that only a single key extension may be active at a time. This restriction does not apply to the permission extension case.



Sessions may have a single Request (K=1) operation at a time. `dof-2008-4-spec-3`

Any attempt to have two outstanding extension Requests at the same time must cause the session to close.

Each secure Request (key extension) establishes a new session key and session key expiration that is equal to the current time plus the **Granted Duration** from Authentication Response Block. Before this expiration, a new session key is required or the session must terminate.



Security state expiration terminates sessions. `dof-2008-4-spec-4`

DOF security chains security state in point-to-point communication. This means that if the security state expires, there is no way to negotiate a new security state, and so the session must terminate.

The responder is able to respond to a Request command once it obtains tickets from the AS. The response tells the initiator which parameters the responder used, and proves the validity of the tickets and the responder's ability to understand the responder ticket (which the initiator never sees) through its ability to extract the same session key. The responder responds to the Confirm command in order to finalize the command sequence and notify the initiator in the case of lossy transports.

The initiator and responder establish security state as described in the tables, above.

4.1.3. Security Considerations

As discussed above, nodes use this command in several different ways. In all cases, the sequence results in authenticated knowledge of a shared security state. The promotion or finalization of security state depends on the session type and takes place for transmission and reception separately. The following table indicates the finalization for both lossless and lossy sessions.

Table 4.1. Security State Finalization – Lossy Sessions

	Send State Finalized	**Receive State Finalized**
Initiator	After receiving Confirm ACK.	After receiving Accept.
Responder	After receiving Confirm.	After receiving Confirm.

Note that lossy sessions maintain the ability to transmit and receive using different security state. This allows the reception of out-of-order packets.

Table 4.2. Security State Finalization – Lossless Sessions

	Send State Finalized	**Receive State Finalized**
Initiator	After sending Confirm.	After receiving Accept.
Responder	After sending Accept.	After receiving Confirm.

Note that in the lossless case, finalization of transmission and reception occurs on both the initiator and responder in response to the same PDU. This is possible because of the lossless nature of the transport, which allows a temporal synchronization of both nodes.

The lossy case is much more complicated, for two reasons: first, the initiator must allow for packet loss and retry the command until receiving a response; second, the responder finalizes transmission and reception state on receiving a Confirm, but the initiator may retry with the previous security state, as its transmission state does not finalize until receiving a Confirm ACK.

This means that both the initiator and responder must be able to differentiate the security state used, must be able to finalize transmission and reception state separately, and must be able to receive using at least two different security states at the same time.

Nodes send packets with the new send state after finalization if **K** is one.

Note that this requirement causes problems for the case of session initialization. In this case, the Confirm command is sent unsecured (the initiator send security state is not yet finalized). However, the responder's send state finalizes based on the Confirm, and so the Confirm ACK should be sent secured. As a general principle, this is impossible. Security requirements will not allow a secure response to an unsecured command and the OPID (likely) must change between the unsecured command and the secured response.

In order to avoid this contradiction, in the case where the Confirm is unsecured, the Confirm ACK is not used, and instead a TEP no-op (empty command) is used. The stack allows a no-op in all cases, and it is a secured packet. Since the only time this cases arises the initiator can have a single outstanding Request the response (of a secure no-op) is not ambiguous.

The purpose of the Confirm ACK is to resolve the ambiguity of multiple secure Requests that may be outstanding at the same time (implying that **K** is equal to zero).

4.1.4. Retry Behavior

Lossless sessions do not allow retries.

Lossy session require retries. The implementation must send retries until it receives the corresponding response or reaction.

A lossy transport likely sends retransmissions as well, meaning that lossy sessions may receive both retransmissions and retries.

4.1.5. Cryptography

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

Table 4.3. TEP Request - Cryptographic Parameters

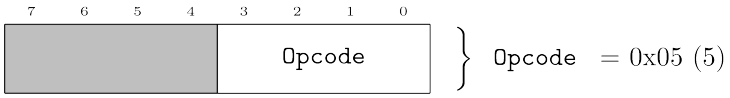
Parameter Name	Parameter Contents
I	Session Initiator Block is in the command.
R	Session Responder Block is in the response.
A	Authentication Response Block is in the response.
S	Session Initialization is in the response. If not present then nodes use an empty block.
K_I	The 256-bit shared key between the initiator and authenticator. This key should be private and only known to the initiator and the authenticator.
K_R	The 256-bit shared key between the responder and authenticator. This key should be private and only known to the responder and the authenticator.
K_D	The 256-bit distributed key, assigned by the authenticator. This will be the session key of the new security state.

4.2. End Session

Ends a session.
Session: Lossy (2-node), Lossless
Addressing: Unicast

Unsecured: Not allowed
Encryption: Allowed
Message authentication: Required
Permission (command): None
Permission (response): N/A

⇄ **End Session** `dof-2008-4-pdu-7`



`Opcode` **Four bits.** This defines the specific operation.

4.2.1. Details

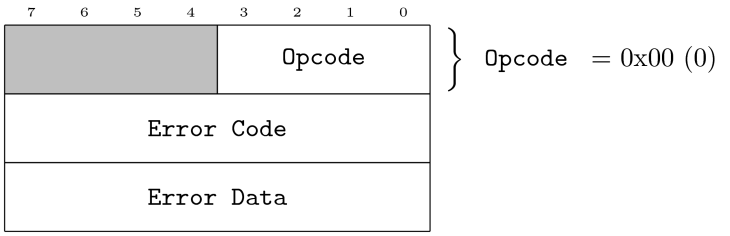
Clients use this command to close a point-to-point secure session. Note that only the session indicated is closed, and other TEP sessions on the same transport session may still exist.

4.3. Reject

Notifies the initiator that the responder has rejected an operation.
Session: None, Lossy (2-node), Lossless
Addressing: Unicast
Security: Must match that of the corresponding Request.
Permission (command): N/A
Permission (response): None

⇄ **Reject** `dof-2008-4-pdu-8`

RESPONSE
Instance of TEP Command/Response Format`dof-2008-4-pdu-2`



`Opcode` **Four bits.**

`Error Code` **One byte.** A code that indicates the condition that caused the request to fail.

`Error Data` **Variable bytes to end of PDU.** Data associated with the error.

4.3.1. Flows

Request()/Confirm() $\rightarrow$ Reject() : The command failed.

4.3.2. Details

Responders use this response in failure cases. The responder may tunnel errors generated by the AS in the **Error Code** and **Error Data** fields, indicated by an **Error Code** with its most significant bit cleared. Errors from the responder have the high-bit set, and come from the following table:

Table 4.4. TEP Error Codes

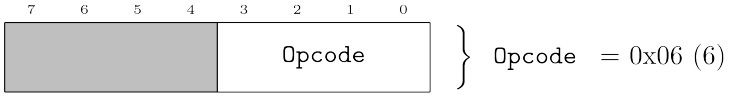
Error Code	Short Description	Error Data	Long Description
0x81	Parse Error	*none*	The responder was not able to parse the request. This is most likely due to a defect or configuration error.
0x82	Mode Error	DSP Command/ Response Format	The requested security mode is not acceptable to the responder. The **Error Data** contains a DSP response that can provide the initiator with guidance on making a future request. The response will be an Configuration Negative Acknowledge or a Configuration Reject based on the responder's capabilities and configuration.
0x83	No Response	*none*	Communication with the authentication server failed.
0x84	Unknown Domain	*none*	The Responder does not know the requested security domain.
0x85	Authentication Failed	*none*	The responder could not authenticate. Responders use this for all responder authentication failures, even if TRP provided more detailed error information.
0x86	Insufficient Resources	*none*	The responder does not have sufficient resources to respond.

4.4. Session Ending

Notifies a client that a session is going to end.
Session: Lossy (2-node), Lossless
Addressing: Unicast
Unsecured: Not allowed
Encryption: Allowed
Message authentication: Required
Permission (command): None
Permission (response): N/A

⇄ **Session Ending** `dof-2008-4-pdu-9`

COMMAND/DIRECTED
NOOPID
Instance of TEP Command/Response Format`dof-2008-4-pdu-2`



`Opcode` **Four bits.**

4.4.1. Details

Servers use this command to notify the client that a session going to end due to the expiration of security state. An issue with long-lived secure sessions is that when the security state expires the session terminates. Because sessions may exist for some time, the issue of clock drift can cause problems.

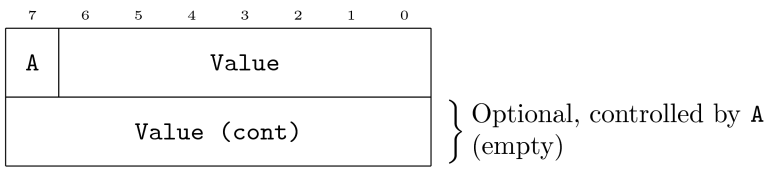
If the client believes the session will expire and it desires the session to continue then it may obtain a new security state and continue the session by using the Request command. However, if the server believes that the session should terminate then its only recourse is to close the session. In the case of a lossy session, there is no notification to the client. Clock drift compounds this problem, where the server's sense of time is faster than the client's is. The client may believe that it still has time to extend the session when the server thinks the session is expiring.

To avoid these problems the server should send this command at some reasonable times before the expiration of the session. These times should be sufficient to allow the client to extend the session if desired. On lossy transports, the server should send the command a sufficient number of times to ensure an opportunity for the client to receive the command.

5. External PDU Reference

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

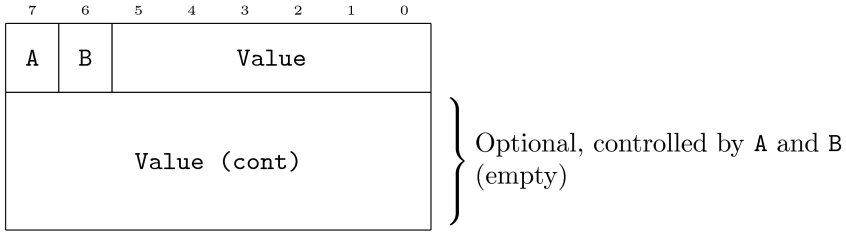
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`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`

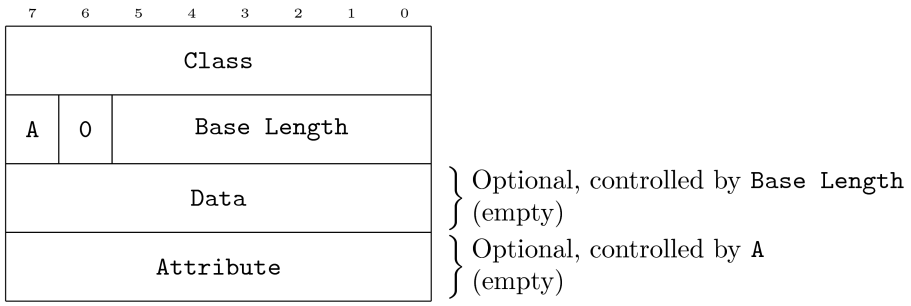


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`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.

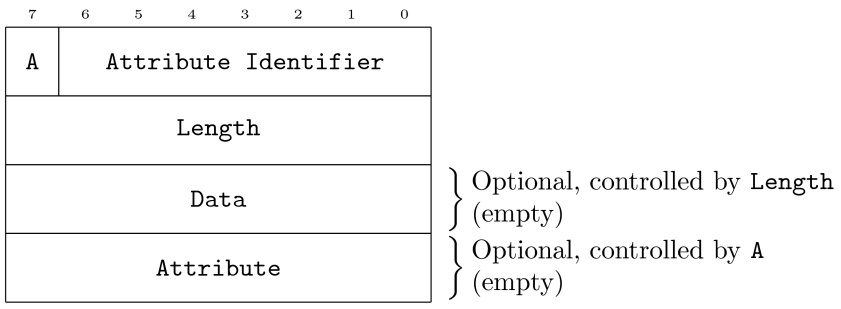
`A` **One bit.** Controls the presence of `Attribute`.

`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.

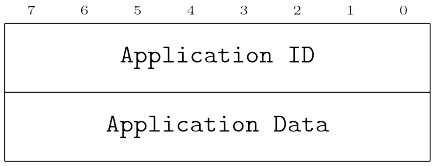
`Length` **One byte.** This field contains the length of the `Data`.

`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)

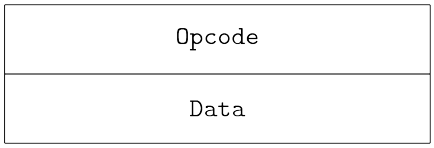
⇄ **General Application Header** `dof-2008-1-pdu-13`



`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.

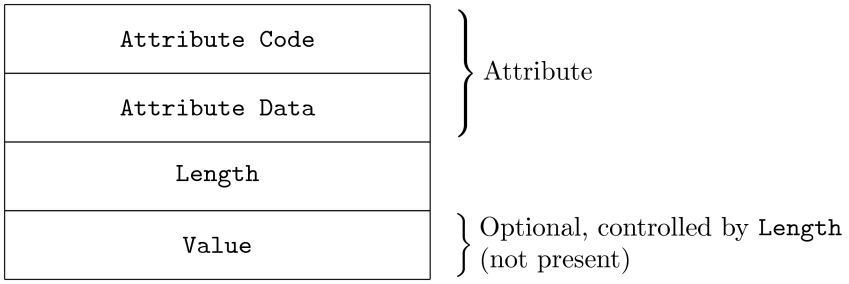
⇄ **DSP Command/Response Format** `dof-2008-1-pdu-15`



`Opcode` **One byte.** The value indicates the particular operation.

`Data` **Optional, variable length.** Each `Opcode` will define the contents of this field.

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.

Configuration Negative Acknowledge `dof-2008-1-pdu-19`

RESPONSE
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`



`Opcode` **One byte.**

`Options` **Array of** Attribute/Value Pair`dof-2008-1-pdu-16`.This field list contains the requested options.

Configuration Reject `dof-2008-1-pdu-20`

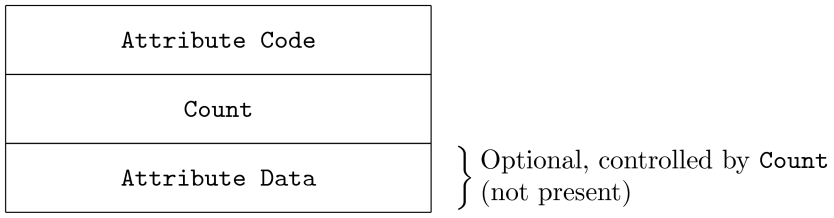
RESPONSE
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`



`Opcode` **One byte.**

`Attributes` **Array of** Attribute Data List`dof-2008-1-pdu-21`.This field list contains the rejected attributes.

⇄ **Attribute Data List `dof-2008-1-pdu-21`**



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

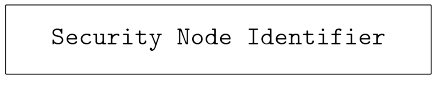
`Count` **Two bytes, MSB.** This field determines the length of the `Attribute Data`. It may be zero, in which case the `Attribute Data` field is not present.

`Attribute Data` **Two bytes, MSB.** This list contains Attribute Data values that are understood and are acceptable with the associated Attribute Code.

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

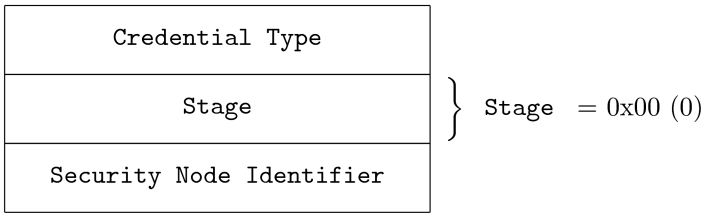
⇄ **Security Node Identifier `dof-2008-16-pdu-1`**

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.

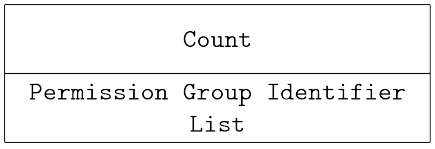
Security Domain `dof-2008-16-pdu-5`

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.

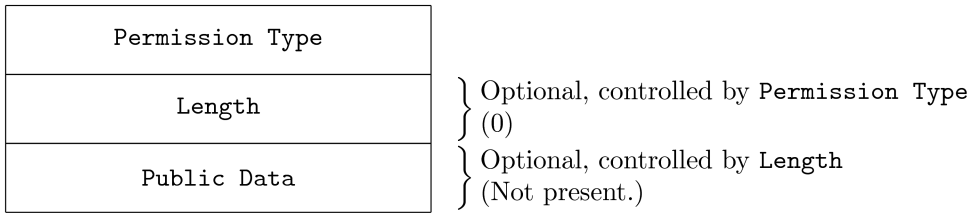
Security Scope `dof-2008-16-pdu-6`



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

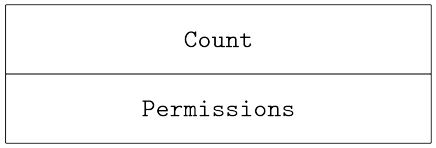
`Permission Group Identifier List`	**Array of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `Count`.

⇄ **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.

Permission Validation `dof-2008-16-pdu-9`

Count
Permission Security Scope List

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

`Permission Security Scope List` **Array of** Security Scope`dof-2008-16-pdu-6`.**Optional, controlled by** `Count`. This is a parallel list to the corresponding permission request.

Security Mode of Operation Negotiation `dof-2008-16-pdu-11`

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

Attribute Code	
Attribute Data	} greater than 0x6000 and less than 0x7000
Length	
Value	} Optional, controlled by `Length` (not present)

`Attribute Code` **One byte.**

`Attribute Data` **Two bytes, MSB.**

`Length` **One byte.**

`Value` **Variable length. Optional, controlled by** `Length`.

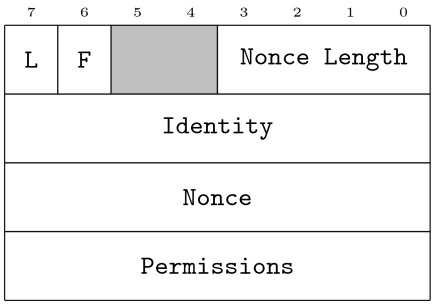
Security Mode of Operation Initialization `dof-2008-16-pdu-12`

Length
Initial State

`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.

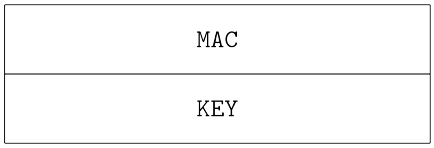
`Nonce Length` **Four bits.** Indicates the length of the `Nonce Length` field.

`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*.

Session Initiator Block `dof-2008-16-pdu-15`

Desired Duration
Desired Security Mode
Initiator Request

`Desired Duration` **One byte.** The desired duration of the secure session. The field is represented in minutes, with eight minute units. This represents the desired expiration time for the associated permission grants and session key.

`Desired Security Mode` **Instance of** Security Mode of Operation Negotiation`dof-2008-16-pdu-11`.The desired security mode of operation.

`Initiator Request` **Instance of** Key Request`dof-2008-16-pdu-13`.Initiator authentication information and permissions.

Session Responder Block `dof-2008-16-pdu-16`

Responder Request

`Responder Request` **Instance of** Key Request`dof-2008-16-pdu-13`.Responder authentication information and permissions.

Authentication Response Block `dof-2008-16-pdu-17`

Granted Duration
Session Security Scope
Initiator Validation
Responder Validation

`Granted Duration` **One byte.** The granted duration of the secure session. The field is represented in minutes, with eight minute units. The granted duration can be less than, equal to or greater than the

	requested duration. This represents the actual expiration time for the granted permissions and session key.
`Session Security Scope`	**Instance of** Security Scope`dof-2008-16-pdu-6`.The security scope of the session.
`Initiator Validation`	**Instance of** Permission Validation`dof-2008-16-pdu-9`.The validation block for the associated initiator request.
`Responder Validation`	**Instance of** Permission Validation`dof-2008-16-pdu-9`.The validation block for the associated responder request.