DOF Security Specification

DOF Protocol Specification

Technical Steering Committee

DOF Security Specification: DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-16
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	DOF Security Specification, OpenDOF TSC, [dof-2008-16] (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. Common Aspects of Security 10
        2.1.1. Encryption 10
        2.1.2. Data Integrity 10
        2.1.3. Authentication 11
        2.1.4. Access Control 12
    2.2. Common Security Vocabulary 13
        2.2.1. Security Components 13
        2.2.2. Security Policy 13
3. Credentials and Identification 15
    3.1. Identities in DOF Security 15
    3.2. DOF Authentication Methods 16
    3.3. Source Identifier Security 19
4. Security Domains 21
    4.1. Domain Identification 21
    4.2. Security Information 22
        4.2.1. Node Identity 22
        4.2.2. Node Authentication Information 23
        4.2.3. Node Permissions 23
    4.3. Related Domains 23
5. Permission Groups and Scopes 25
    5.1. Compatibility 26
    5.2. Security Scope Usage 28
        5.2.1. Receiving an Operation 30
        5.2.2. Routing 30
        5.2.3. Output 30
6. Permissions and Access Control 33
    6.1. Permission to Connect 34
    6.2. Permission Storage 34
    6.3. Aggregate vs. Terminal Permissions 34
    6.4. Requesting Permissions 34
        6.4.1. Normalization 35
        6.4.2. Complementing Permissions 36
        6.4.3. Comparison of Permissions 36
        6.4.4. Intersection of Permissions 37
    6.5. Permission Validation for Sessions 37
        6.5.1. Validation of 2-Node Secure Sessions 38
        6.5.2. Validation of N-Node Secure Sessions 38

6.5.3. Validation of Single Nodes 39
6.6. Permission Validation for Operations 39
6.7. Permission and Permission Request Formats 40
6.7.1. Permission Types 41
6.8. Security Scope Results 42
7. Securing Communications 45
7.1. General Secure Session Considerations 45
7.2. General Key Distribution Considerations 46
7.3. Point-to-Point Security 47
7.3.1. Key Exchange 47
7.3.2. Lifecycle 47
7.3.3. Permission Definition 48
7.3.4. Timing 48
7.3.5. Replay Attack Prevention 48
7.3.6. SID Security 48
7.4. Multipoint Security 48
7.4.1. Common Solutions 49
7.4.2. Multicast 50
7.4.3. Point-to-Multipoint 50
7.5. DOF Security Modes of Operation 52
7.5.1. Common Behaviors 53
7.5.2. Security Strength 53
7.5.3. Modes of Operation 54
8. Cryptography 55
8.1. References 55
8.2. Security Primitives, Upgrade, and Failure Mitigation 55
8.3. Intellectual Property Statements 56
8.3.1. AES 56
8.3.2. CCM 56
8.3.3. SHA-256 57
8.3.4. HMAC 57
8.3.5. KryptoKnight 57
8.4. Common Security Blocks 57
8.4.1. Key Request 57
8.4.2. Key Grant 58
8.4.3. Session Blocks 59
8.5. Notation 60
8.6. Hashing 61
8.7. Three-party Cryptography 61
8.8. Two-party Cryptography 63
9. Service Security 65
9.1. Security in the Universal Approach 65
9.2. Security in the Extended Approach 65
9.3. Requestor-Specific Access 66
10. Distributed Security 67
10.1. Security through Distributed Sessions 67
10.2. The Effects of Key Duration 68
11. Security Examples 71
11.1. DOF Security Pseudo-Protocol 71
11.2. General Example Formatting 73
11.3. Basic Session 73
11.3.1. Success 73
11.3.2. Failure 74
11.4. Group Join 74

11.4.1. Success 74
11.4.2. Failure of One Node 74
11.5. Proxy 75
11.5.1. Proxy with Pre-Authorized Session 75
11.5.2. Proxy with Failed Permissions 76
11.6. Changed Permissions 77
12. External PDU Reference 79
12.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 79
12.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 81

List of Figures

2.1. Credential Resolution 12
5.1. Different Phases in Security Bridging 29

List of Tables

8.1. General Three-Party Parameter Block Names 61
8.2. General Two-Party Block Names 64

Specifications and PDUs

Security Node Identifier **dof-2008-16-pdu-1** 15
Security node identifiers must not include attributes and must not be broadcast/multicast.
dof-2008-16-spec-1 16
Base Credential Format **dof-2008-16-pdu-2** 17
Identity Resolution **dof-2008-16-pdu-3** 17
The Base Credential Format, Identity resolution, and any Storage format are unambiguous.
dof-2008-16-spec-2 18
Identity Persistence **dof-2008-16-pdu-4** 18
All DOF credential types must be registered with the ODP-TSC before use in product.
dof-2008-16-spec-3 19
Multipoint groups must use the group SID and each member has an implied IAM for that SID.
dof-2008-16-spec-4 19
All SID use must match the IAM and ACTAS permissions on the secure session. dof-2008-16-spec-5 19
SIDs are not allowed to cross domain boundaries, including from unsecured to secure. They may cross from secure to unsecure, however. dof-2008-16-spec-6 20
Without the use of a security gateway or application protocol definition, all communication must remain in the same domain. dof-2008-16-spec-7 21
Security Domain **dof-2008-16-pdu-5** 21
Security domain identifiers must not include attributes and must not be multicast. dof-2008-16-spec-8 22
Node authentication information must be unique and only shared with the AS of a single domain.
dof-2008-16-spec-9 23
Permissions granted to a node are always associated with a set of permission groups. dof-2008-16-spec-10 23
Permission to tunnel remote domains must be negotiated and enforced. dof-2008-16-spec-11 24
Permission-controlled operations must be scoped by the permission group. dof-2008-16-spec-12 25
Permission group identifiers are non-negative and less than 2^{30}, with identifiers greater than 2^{29}-1 being reserved. dof-2008-16-spec-13 25
Security Scope **dof-2008-16-pdu-6** 26
Permission-controlled communication must always be between nodes where the session security scope is not empty. dof-2008-16-spec-14 28
Unsecured communication must be separated from secured communication unless configured otherwise. dof-2008-16-spec-15 28
Each secure session must be assigned a session security scope. dof-2008-16-spec-16 28
Operations are handled in each permission group identifier as if received independently.
dof-2008-16-spec-17 29
Operation security scopes can be merged by proxy nodes, but responses must remain in their assigned security scopes. dof-2008-16-spec-18 29
Permission group identifiers are only valid in their assigned domain. dof-2008-16-spec-19 30
Security group identifiers passed on the wire must limited to the operation security scope as determined by the receiver. dof-2008-16-spec-20 30
Senders must ensure that the operation security scope is not expanded by the receiver.
dof-2008-16-spec-21 31
Operations using the global security scope must be explicitly marked, even when the sender and receiver agree that no expansion is occurring. dof-2008-16-spec-22 31
Operations must only require terminal permissions. dof-2008-16-spec-23 35
Permission checks must be effectively made between normalized permission sets. dof-2008-16-spec-24 35

Permission normalization must not extend permission scoping. dof-2008-16-spec-25 35
Nodes must reject requests to enforce unrecognized permission types. dof-2008-16-spec-26 36
Validation requirements for unknown permission types. dof-2008-16-spec-27 37
Permission dof-2008-16-pdu-7 40
Permission types must be registered with ODP-TSC before use in product. dof-2008-16-spec-28 41
Ignore the reserved range of permission types. dof-2008-16-spec-29 41
Permission Request dof-2008-16-pdu-8 41
Permission Validation dof-2008-16-pdu-9 42
Permission Security Scope dof-2008-16-pdu-10 42
Secure sessions must reject all unsecured datagrams except those belonging to security modes of operation. dof-2008-16-spec-30 46
Security state must not be shared outside the session. dof-2008-16-spec-31 46
Node identities must not change during their participation in a session. dof-2008-16-spec-32 46
Security Mode of Operation Negotiation dof-2008-16-pdu-11 52
Security Mode of Operation Initialization dof-2008-16-pdu-12 53
All DOF security strengths must be registered with the ODP-TSC before use in product. dof-2008-16-spec-33 54
Security modes of operation must be registered with ODP-TSC before use in product. dof-2008-16-spec-34 54
Only use security primitives with other nodes that have been approved by the AS. dof-2008-16-spec-35 56
Key Request dof-2008-16-pdu-13 57
The maximum Nonce length is 16 bytes, the minimum is 1 byte. dof-2008-16-spec-36 58
Key Grant dof-2008-16-pdu-14 58
Session Initiator Block dof-2008-16-pdu-15 59
Session Responder Block dof-2008-16-pdu-16 59
Authentication Response Block dof-2008-16-pdu-17 60
Nodes must use appropriate random number generation. dof-2008-16-spec-37 60
The ticket hash must be recreated and validated before the key can be used. dof-2008-16-spec-38 63
Two party authentication parameters must not alias when concatenated with other security blocks. dof-2008-16-spec-39 64
Compressed Unsigned 16-Bit dof-2009-1-pdu-1 79
Compressed Unsigned 24-Bit dof-2009-1-pdu-2 79
Compressed Unsigned 32-Bit dof-2009-1-pdu-3 80
Object Identifier (OID) dof-2009-1-pdu-4 80
Object Identifier Attribute dof-2009-1-pdu-5 81
Attribute/Value Pair dof-2008-1-pdu-16 81

1. About This Document

This document presents the technical specification of DOF security. It builds on the foundation of the guidelines presented in *Guidelines for DOF Security*. This document is much more technical.

An understanding of the topics covered in *Guidelines for Networking with DOF* is helpful to understanding this document, as is some experience with Internet security protocols.

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.

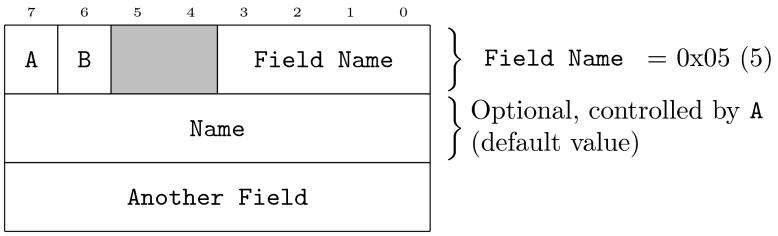
1.3.4. Fields

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 presents common protocol definitions related to DOF security. Higher-level information about DOF security can be found in *Guidelines for DOF Security*. Information that is presented in that document is not repeated here. Discussions about attack prevention and possible modes of attack are also not discussed in this document.

This document is meant to provide only the aspects of DOF security that are shared among all application protocols, or at least within families of application protocols. Specific security information can be found in the different application protocol specifications. For example, the common aspects of key distribution are discussed here, although the actual protocols used are described in their own documents.

Security modes of operation and permissions are also discussed here as they are universally used, but specific documents cover the technical details of each mode or permission.

DOF security is based on centralized authentication and permission information that is stored in and distributed from an authentication server (AS). Communication, either direct or indirect, with an authentication server is implied whenever security information is known. The authentication server is usually denoted as the "AS".



In this document the term "AS" refers to a logical entity and may correspond in reality to any software component that has access to the required security information.

DOF security is provided by many different components, each providing different parts of a complete security solution. Accepted industry standards are utilized where possible - especially in the areas of encryption and key distribution.

DOF security is based on established standards and research:

SSL (Secure Socket Layer)	SSL is the de-facto standard for IP-based secure communications. DOF implementations can use SSL where appropriate.
AES (Advanced Encryption Standard)	AES is the successor to DES, and is accepted by the U.S. Government for encryption of classified information.
KryptoKnight and Kerberos	KryptoKnight was developed by researchers at IBM as a follow-up to Kerberos, and defines a secure method for authentication, key distribution, and information validation. DOF access control, authentication and key distribution are patterned after KryptoKnight. Since KryptoKnight is not a standardized protocol, the DOF specification has incorporated its principles directly into the key-exchange protocols.
SRP (Secure Remote Password)	SRP solves issues regarding dictionary attacks on low-entropy passwords.

These components provide the foundation for DOF security. However, none of these components meets all of the requirements of a secure solution. This means that they must be combined in order to be effective. This combination of components is provided by both DOF components (like the authentication server) and protocols (for key distribution, permission management, etc.).

The core components of DOF security are each accepted standards or currently accepted research, and the probability of there being fundamental flaws in any of them is extremely low. Still, there is nothing in DOF implementations that forces these core components to be used - they are negotiated as part of session establishment or set as part of security policy as discussed later in this document. This provides

the flexibility that if a security breach is identified then the protocol that exhibits the flaw can be rejected. Devices with alternate encryption or security methods are able to sense these failures and negotiate alternate methods.

For example, a node that is able to encrypt with both AES and TWOFISH could sense a rejection of AES and attempt to negotiate TWOFISH. This type of fallback logic, if used appropriately, can provide robust security.



DOF security applies *above* the transport that is being used. For example, it does nothing to protect TCP/IP headers. In order to be transport independent DOF specifications define their own network layer. These headers are not secured either, similar to the handling of Ethernet headers in SSL.

2.1. Common Aspects of Security

There are four primary aspects of security: Encryption, Data Integrity, Authentication, and Access Control. While each of these aspects is unique, the systems that provide them are frequently interrelated. For example, key distribution is frequently used as both an aspect of authentication and of encryption/data integrity. Authentication is a core requirement for access control.

The following sections describe each of these security aspects, and provide an overview of how DOF implementations provide the functionality.

2.1.1. Encryption

Encryption is the process of transforming a plain-text message into cipher text. Without knowledge of a shared encryption key, it is extremely difficult to obtain the plain-text message from the cipher text.

DOF security can use two fundamentally different methods for encryption: transport provided and DOF provided.

2.1.1.1. Transport Provided

Any point-to-point or group-based encryption methods provided by the transport can be leveraged in DOF security. Commonly provided methods include SSL, HTTP/S and IP-SEC. DOF security makes use of transport-provided encryption by not re-encrypting that data and by assuming the integrity of the data.

Note that some of these methods (like SSL) already include some form of unidirectional or bidirectional authentication. DOF security does not require this authentication information, and always uses the authentication server (AS) for its authentication. Any server or client authentication provided by the transport is an added benefit. Keep in mind that DOF security does *not* base security on physical node identity (like SSL), but rather on application identity. This is a key point in its distributed nature.

2.1.1.2. DOF Provided

DOF encryption is flexible, and can leverage many encryption methods. The actual encryption and message validation algorithms are configurable at a session level. The encryption associated with a session is established along with the security mode. DOF implementations can leverage a wide variety of block encryption algorithms.

2.1.2. Data Integrity

The process of ensuring that data is not modified between being sent and being received is often related to encryption, but uses completely different technologies. DOF security can ensure the integrity of data

separate from encryption, although the processes used rely on a shared key in much the same way that encryption does.

Data integrity is managed by the mode of operation and is negotiated as part of the security mode for each session.

2.1.3. Authentication

In order to provide for access control, the identity of the agent that is requesting the operation must be known. DOF security provides this verified identity through authentication of each secure network agent. For point-to-point communication, each agent is authenticated to the other, meaning that each agent can securely identify the agent on the other end. For group communication, no such identification is known, apart from the node having permission to be in the group. This means that group communication is authenticated only to the "group", and not to the individual node.

Authentication is often solved through the demonstrated knowledge of a shared secret along with some identifier. A good example is a username/password, where the password represents the shared secret. This would provide little security if the username/password was passed in the clear on the network, so secure "private" communication of some sort is required.

The methods of authentication supported by DOF security will fall into some major categories. Each authentication method uses a particular set of networking protocols to accomplish its task. The authentication server provides all DOF authentications by using a KryptoKnight-based key exchange. DOF identification is extremely flexible and extendible. This document discusses "authentication servers" in an abstract sense. In reality, the features of an authentication server may be integrated with other components. Different topologies for providing security are presented in *Guidelines for DOF Security*.

Authentication information is stored in the authentication server. It is always associated with a security domain (discussed later). Key to the definition of authentication is the concept of credentials, discussed here.

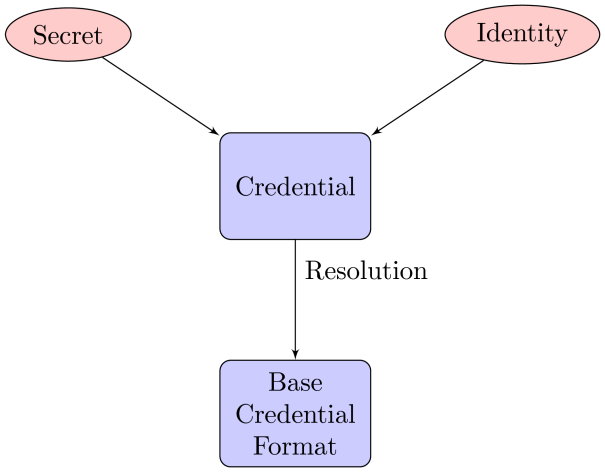
2.1.3.1. Credentials

DOF security uses the term "credentials" to mean information that uniquely identifies an agent, independent of the type of agent. For example, a human user may use a "username" and a "password" as credentials. However, many different types of credentials are possible, even given the same "inputs" of username and password (or other information). Because of this the DOF term "credential" is also related to the method or process used to validate the provided information.

Several common aspects of credentials are specified in this document. DOF credentials have a common base format, and define a flexible way of determining the base format from a variety of different inputs. For example, a fingerprint might be a valid input, but is not in the common base format. The process of moving from a fingerprint (in some format, possibly vendor-defined) to the common base authentication format is called "resolution". Resolution is managed between the node and the AS.

This flow of information is important to understand. Refer to the following diagram:

Figure 2.1. Credential Resolution



Note that almost all security operations work exclusively with base credential formats. All credentials that are not already in that format must undergo resolution before they can be used.

2.1.4. Access Control

In DOF security, all access control is based on the authenticated identity of the requesting agent. Access control is not symmetric, meaning that an operation may be allowed in one direction that is not allowed in the other direction on the same session.

A primary goal of security is to control access to resources. DOF security includes support for access control based on the specific requestor and provider identities (that are authenticated) and is managed through the request and grant of permission sets.

The effects of these permission sets are ideally separated from the specific protocols used by DOF implementations, but there is necessarily a "matching" of operations to permissions for each protocol. This document defines the common format shared by all permissions, and individual application protocols are then free to define the specific permissions that they use to control their own operations. It is permissible for one application to use the permission format of another application, and is encouraged to do so. Once a permission format is defined, its format (and the meaning of the format) is fixed.

Once the relationship between the initiator and responder (which represent the "session") is established, permissions can be requested. The two ends of the session are referred to as the "initiator" and the "responder". Both the initiator and responder have responsibilities to ensure that security is maintained. Both nodes must ensure that the other has been granted the required permission for each operation it sends. Each node must also ensure that the other has permission to send the responses that it accepts.

The principle of proxy has a huge impact on access control. In DOF security, nodes can forward operations to other nodes for actual processing. However, in forwarding the operation the node must take on both the role of initiator and responder. It must ensure that the node that it uses to get the response does not violate security requirements. The original node must ensure that the proxy also has all the required permissions. In this way, proxy is transparent to the original requesting node. It is also transparent to the actual responding node. The additional requirements apply to the proxy node itself.

In order to achieve high performance it is useful to "turn off" permission checking on some links. This is done by requesting permissions that use wildcards and so can apply to many operations. In this way

the security checks still apply, but can be done quickly and without the need to request new, specific permissions for each operation.

For example, it may be common for clusters of nodes to delegate detailed permission checking to the non-cluster sessions and agree to exchange all information using "wildcard" permissions inside the cluster. Further, to optimize performance, such a cluster that is physically secure may choose to use transport provided security (or even no encryption at all) on its "internal" sessions. This kind of flexibility is allowed by DOF security, and for the most part is managed by the authentication server.

The permissions that control access are always stored in the authentication server and are associated with a security domain.

2.2. Common Security Vocabulary

This section presents a common vocabulary and method for discussing DOF security. This method will be used throughout this document, and in the protocol specifications that discuss security. The vocabulary and method chosen is mathematical, and relies extensively on the symbols of set notation.

2.2.1. Security Components

Security is based on the sharing of information between nodes. This shared information is referred to as the "security state" of the node or session involved. In DOF implementations there are several different sets of state that are necessary.

2.2.1.1. Initiator

The initiator is the node that starts the creation of a secure session. The initiator therefore creates the first security state related to the session.

2.2.1.2. Responder

The responder is the node that accepts the creation of a secure session. The initiator and responder both maintain their own security state.

2.2.1.3. Session

The session is an abstract entity that represents the ability of nodes to communicate. The session security state is distributed between all the communicating nodes. However, there is some security state that must be shared in order for secure communication to be possible. For example, the encryption key used to encrypt messages must be known by all nodes in the session if they hope to understand the encrypted messages. This means that some of the secure session state must be shared, while other secure session state may be distributed and not known by all nodes.

2.2.1.4. AS

As mentioned earlier, the AS (Authentication Server) is the common repository of security information in a system. The AS manages security state of the initiator and responder, and both of these components must share some state with the AS in order to establish a secure session.

The AS also manages the security policy for the domains that it manages.

2.2.2. Security Policy

Security policy refers to the requirements placed on security-related information in an DOF system. For example, a requirement that passwords be longer than 7 characters is part of a security policy. For the most

part, the security policy is controlled by the management tools associated with the system, and not directly controlled by DOF protocols themselves.

An exception to this general rule deals with permission and session key revocation, discussed later.

3. Credentials and Identification

One of the most basic components of DOF security is identification and the creation of credentials. There is a small difference between these two terms: identification relates to methods of uniquely identifying a node, and a credential is the data that not only identifies the node but contains whatever proof is necessary to authenticate the node.

For example, a common identification method may be an email address. However, that same identity may be associated with several different authentication methods (a password, a shared key, etc.). Combining the identification with one of the authentication secrets forms a credential.

By necessity, the secrets associated with a credential are not passed on the wire. The identification information, however, must be. This means that the credential information and the identification methods are very closely related.

3.1. Identities in DOF Security

A brief review of identity is helpful. In DOF security a node has an identity. However, the actual identity of the node is usually not important. This apparent contradiction is related to the principle of proxy discussed in *Guidelines for Networking with DOF*. What really matters (most of the time) are the objects that the nodes provide and use. The nodes rely on the permissions (associated with their identities) to ensure that they are using the correct objects and in the allowed ways.

However, there may be times when the identity of the nodes is important. DOF security ensures that the node's identity is authenticated and known to both parties involved in communication, with an exception for group communication. In all point-to-point communication, the identities are known.



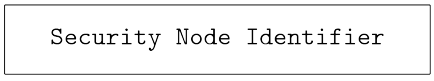
Do not confuse the word "identity" as used here with the identity of an individual. What is being identified is a node, not a person. If the node identifier described next includes personal information then there may be a link between the node identifier and a person's identity, but that is an application/management choice and not a requirement of the DOF specifications.

All base identities in DOF security are object identifiers, but they have some restrictions. The following PDU is used whenever an identity is required in the base format:



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.



Security node identifiers must not include attributes and must not be broadcast/multicast. `dof-2008-16-spec-1`

This reduces length and complexity.

Note that the contexts which allow a security node identifier are *always* scoped to a security domain. This means that, unlike general object identifiers, there is not a requirement for *global* uniqueness, only uniqueness within a given security domain.



This lack of global uniqueness is a key aspect of allowing authentication without identification.

3.2. DOF Authentication Methods

DOF security allows for a flexible definition of authentication methods. The word "authentication" is used in many different ways, and so to be clear in this section we are talking about two-party authentication (between the AS and a node). Note that the typical DOF key distribution relies on three-party authentication, and those techniques are *not* described in this section.

Authentication, in the context of this section, is the process of securely identifying a user or node and determining a shared secret between that node and the authentication server. For example, a very simple form of authentication is a user providing a user name to a server. The user knows a secret password. The server looks up the user in a database and retrieves the secret password. At the end of the process, the two sides know both the user name and the shared, secret password. Both sides can now use that information to proceed toward *proving* their knowledge of that information, or authenticating the user. This indicates the difference, but close relationship, between identification and authentication.

However, that simple approach does not cover all of the possibilities. In general, the process of identification and establishing shared knowledge of a secret is more complicated. DOF security uses the following process. This process applies to a node and the AS, although other nodes may proxy the actual communication. This process is called "resolution". If successful, it results in a credential that is in the base credential format.

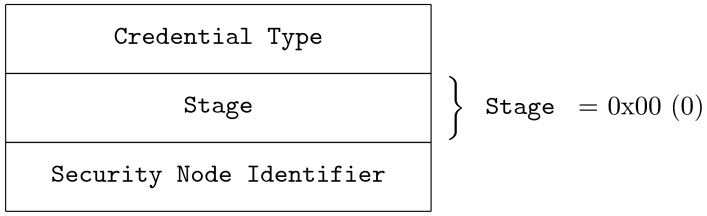
- The node creates a credential. This will always include some form of identity, and some information used to authenticate the identity. This could include any of a variety of information, but contains at least a domain (representing the scope of the identity), an identity or lookup key and some authentication information. In order to be useful, the information must have corresponding information in the AS. Part of creating a credential is determining the type of authentication that will be used. This is represented through the "type" of credential.
- If the domain is not known then it must be determined either from the user or by using discovery. Domain discovery is discussed in the specific protocols that are used with the AS.
- Based on the type of credential and the information it contains, a public request is made. The type of credential determines the format and content of the public request.
- The public request is sent to the AS, which sends a public response or failure.
- The process of sending public requests and receiving public responses continues until the credential determines the shared secret associated with the credential on the AS.
- With the exchange of information complete, the credential contains the domain identifier, the security node identifier and the shared 256-bit secret associated with the credential. If the initial information obtained by the node includes all this information then there is no need to make a public request.
- The credentials are then used, most likely as part of a 3-party authentication request. Credentials in this final state are said to be "resolved".

Each authentication method is discussed in separate documents as indicated below. The common base credential format is defined in this document. Note that credentials are always associated with a security domain, but the domain itself is not part of the base credential format. All uses of credentials must clearly define the associated domain.

The different uses for identification methods fall into three categories: a base credential format, a resolution command, and a resolution response. In addition, a credential needs to define storage formats appropriate to the implementation. Some provision is made for supporting this, but in general it is beyond the scope of this specification.

Each authentication method must specify the formatting for each of the previous categories. However, they are constrained by the following general formats that each identification method must follow:

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

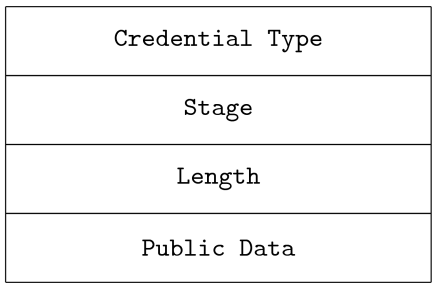


`Credential Type` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

`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.

Identity Resolution `dof-2008-16-pdu-3`



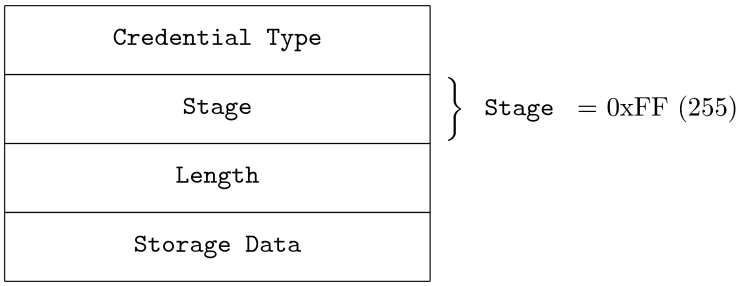
`Credential Type` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

`Stage`	**One byte.** The stage of the request. This value must be greater than zero and less than 255.
`Length`	**Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.Controls length of **Public Data**.
`Public Data`	**Variable length. Optional, controlled by** `Length`. The public data associated with the **Credential Type** and **Stage**.

The Base Credential Format, Identity resolution, and any Storage format are unambiguous. `dof-2008-16-spec-2`

Since the same code may read all these different formats, these PDUs must be unambiguous. This means that the headers (through the **Stage**) must be identical. The **Stage** field distinguishes the use. The value zero indicates the base format, greater than zero and less than 255 indicate resolution (which includes **Length**), and 255 defines the storage format. The storage format must be defined in the specification for each credential type as an instance of Identity Persistence.

Identity Persistence `dof-2008-16-pdu-4`



`Credential Type`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.
`Stage`	**One byte.** The stage of the request.
`Length`	**Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.Controls length of **Storage Data**.
`Storage Data`	**Variable length. Optional, controlled by** `Length`. The public data associated with the **Credential Type** and **Stage**.

Each credential type must define its **Credential Type** and the format of the **Private Data**, **Public Data**, and **Storage Data** in both the request and response for each **Stage**.

For a current list of registered identification methods, please contact the ODP-TSC or go to https://opendof.org/registry-credentials.

All DOF credential types must be registered with the ODP-TSC before use in product. `dof-2008-16-spec-3`

All credential types must be registered with ODP-TSC before use. This includes the specification of the public exchanges with the AS that are necessary to resolve a shared secret. Credential type zero (0) is reserved.

3.3. Source Identifier Security

Associated with each unique identity in a security domain (discussed in a later section) is a SID, or source identifier.

Source identifiers are discussed in the *DOF Protocol Specification*, and are used by nodes to uniquely identify operations that they create. This unique identification means that in secure systems the use of particular SIDs must be validated and controlled – otherwise a malicious node could interfere with another node's operations.

A node must have one of two permissions in order to use a SID: IAM or ACTAS. These permissions are further defined in the *DOF Protocol Specification*, although the specific permissions are governed by the DPP version in use.

The requirement to configure permissions to use a SID seems to conflict with the need for SIDs to be unique. If a node picked a unique *random* SID then it would have no way of having a configured permission. On the other hand, a configured permission would be a problem if two different nodes used the same credentials and requested the same IAM permission.

To resolve this conflict the AS can provide a SID for each credential/identity that it manages. These SIDs are unique to the domain of the credential. Further, the AS ensures that each request to obtain this SID results in a unique SID and that IAM permission requests on each of the different unique SIDs are granted correctly.

A further special case of SID use is on secure groups. Group security does not allow individualized permission based on each group member, which means that the IAM permission is not useful. ACTAS can apply to the group to allow forwarding. To resolve the lack of IAM, group members are assigned a consistent SID by the AS. Each member must use that SID for group operations, or the group must have ACTAS for a proxied operation.

Multipoint groups must use the group SID and each member has an implied IAM for that SID. `dof-2008-16-spec-4`

The implied IAM is part of joining the group. The use of other SIDs on the group must be managed through granted permissions.

All SID use must match the IAM and ACTAS permissions on the secure session. `dof-2008-16-spec-5`

This applies to DPP use in operation identifiers and any commands that explicitly reference operation identifiers or SIDs directly. It does not apply to unsecured communication. Lack of required permissions is handled as a permission violation.

SIDs are not allowed to cross domain boundaries, including from unsecured to secure. They may cross from secure to unsecure, however. `dof-2008-16-spec-6`

This means that any security gateway that bridges from unsecured to secure must create a new operation related to the unsecured operation. This applies even to global operations, which are normally required to use consistent SIDs. This requirement prevents a trivial attack on operation identifiers by having an unsecured node "pollute" the operation identifier space with its own operations.

4. Security Domains

Security domains are abstract containers for security information. They also represent the highest-level of node and communication grouping. Only nodes in the same domain are able to communicate without the use of security gateway (see *Guidelines for DOF Gateways*). Security domains are uniquely identified and require a minimal amount of security configuration information. The following sections present the requirements of a security domain.

Without the use of a security gateway or application protocol definition, all communication must remain in the same domain. `dof-2008-16-spec-7`

Trust is established inside a domain, and so without a security gateway there is no way to pass information securely between domains. This partitioning also applies to remote domains.

The following apply to a security domain:

- Each security domain is uniquely identified. Name collisions are not critical, but should be avoided by using information that is unique to the organization in the domain identity.
- Each security domain uses consistent permission group identifiers (defined later).
- A node is a member of a security domain if it shares at least one secret key with that security domain.
- A node may also be considered a member of a domain *B* if:
  a. The node is a member of domain *A*, and
  b. Domain *A* is related to domain *B*, meaning that the two domains share a key.
- A node must *not* reuse the same secret key with multiple security domains, although it may share different keys with different security domains (and therefore be a member of multiple security domains).
- All permissions assigned to a node are part of a security domain.

Each security domain consists of a configuration that includes at least all nodes in the domain and all permissions assigned to those nodes. An authentication server that has access to this configuration can act for the domain.

4.1. Domain Identification

Each security domain has an associated identifier, which is an DOF object identifier. This identifier is required for all interaction with the authentication server.

Security domains are represented as object identifiers but are not allowed attributes. To make protocol references clear, commands and responses will reference this PDU for security domains.

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 domain identifiers must not include attributes and must not be multicast. `dof-2008-16-spec-8`

This reduces length and complexity. Certain PDUs allow broadcast to be used for the security domain identifier.

4.2. Security Information

A security domain consists of several different sets of related information. While the specifics of how this information is stored are not standardized, the information and relationships are standardized.

Nothing in this section is meant to restrict the information that can be stored about nodes in an DOF network. What is presented represents the minimum that must be provided in order for the DOF protocols to function.

The most important information contained in a security domain is the list of nodes that are in the domain. Each of these nodes is identified by a security node identifier (Security.8: Security Node Identifier). Each also contains at least the following:

- Authentication information (usually a set of **Error! Reference source not found.**). This information is based on the different types of authentication allowed, and is flexible. A node may reference multiple sets of authentication information that allow it to authenticate in multiple ways. Note that some authentication methods require secondary indexes that may not use security node identifiers, but the base authentication format always uses a security node identifier.

- A set of permissions (possibly empty) that are granted to the node and the permission groups associated with each. The methods used to store and associate this information are not standardized.

The details of each of these items are discussed in further sections.

The relationship between permission groups and permission sets is interesting. Whether the permissions are associated with the permission group or the permission group is associated with the permission is an interesting question. There are situations, like management, where it is easier to think about permission sets being associated with permission groups. However, there are also instances, like node authentication, where it is easier to have the permission groups associated with the permission. This document is written as if the permission groups are associated with the permission sets.

4.2.1. Node Identity

Each node in a security domain must have a unique security node identity (Security.8: Security Node Identifier).

The node identity is used for management and for authentication. When a node attempts to communicate securely, it must use this identity. However, the process of resolution may start with other identifiers. For example, a fingerprint is not a security node identifier, but may be used to start resolution. A database of fingerprint data would be used, and this would then be consulted to determine the actual security node identifier.

4.2.2. Node Authentication Information

DOF security supports a flexible method for providing authentication information; however, they are all based on the resolving of a 256-bit key. All methods result in the distribution of a 256-bit key (typically unique to the node and to the authentication method), and that key is then used to authenticate.



Node authentication information must be unique and only shared with the AS of a single domain. `dof-2008-16-spec-9`

Authentication information must only be shared between a node and a single security domain. This ensures isolation of one domain from another, and prevents a breach of security in one domain from affecting the security of other domains. In particular, the use of a common shared secret for many nodes is not permitted.

4.2.3. Node Permissions

Each node in a security domain may have a set of associated permissions, and those permissions must be associated with a set of permission groups. The specifics of the permission handling are discussed later.



Permissions granted to a node are always associated with a set of permission groups. `dof-2008-16-spec-10`

The mechanism for making this assignment is not specified. However, it is not possible to assign permissions to a node without having at least one associated permission group, and the scope of the permission grant is defined by the scope of the associated permission groups.

4.3. Related Domains

Domains may form relationships by sharing a key. Related domains may allow nodes to authenticate into either domain using the same set of credentials, although this requires communicating with the authentication servers of *both* domains.

A session always corresponds to communication in a single domain. Each node participating in the session must obtain credentials in the domain of the session in order to obtain the required security state (including the session key). When there is a difference between the domain of the credentials possessed by a node and the desired domain of the session, the node may use domain relationships in order to obtain credentials in the desired domain.

When this is done there is always a sense of the "local" and "remote" domain. The local domain actually shares a secret with the node, indicating that the local domain is not the domain requested for the session. The remote domain does not share a secret with the node, and so key distribution must be to the local domain, which in turn distributes it to the node. The remote domain represents the desired session domain.

In distributing the key from the remote domain the local domain also assigns a domain identifier to the remote domain. This domain identifier is known as a remote domain identifier, or RDID. The RDID

represents the actual domain of the session within the local domain. Typically, operations originating on a session inherit the security state of the session (including the domain). In the case of a session with an RDID, the security state of the session includes the RDID. This means that for the purposes of domain comparison there are two issues: the local domain must be equivalent *and* the assigned RDID must be equivalent. Note that RDID assignments are only valid within the local domain.

It is critical to note that the traffic originating in the remote domain is *not* actually treated as if it is in the local domain, although it is compatible with the local domain. It is partitioned from the local domain through the use of the RDID. Further, operations that are partitioned in this way do not have their permissions evaluated according to the operation, but rather by the ability to tunnel the specific RDID. However, the requirements related to domain partitioning apply not only to the local domain but also to the RDID – it is invalid to allow operations with one RDID to forward to sessions using a different RDID.

This means that when an operation that is assigned to an RDID is asked what permission it requires that it will respond with the permission to tunnel the RDID, not the permissions required for the actual operation. This is required because the local domain has no knowledge of the security policy of the remote domain.

Note that this tunneling of remote domains also requires preserving the operation security scope from the original (remote domain) session. This security scope is restored when the operation passes back into the remote domain.

Remote domain identifiers are referenced in many different protocols. In all cases they are limited to 30-bit numbers, with 0x3FFFFFFF reserved.



Permission to tunnel remote domains must be negotiated and enforced. `dof-2008-16-spec-11`

Tunneling of remote domains takes place as part of DPP, and the specific permissions required depend on the DPP version used.

5. Permission Groups and Scopes

There are two fundamentally different methods for controlling operation execution and forwarding. The first is through permissions. This method is standardized as discussed in this document, and is fully supported through the DOF protocol stack. The second is based on other protocol-defined factors. Each operation will state which of the two methods it uses. If protocol-defined, then the rules will be found in the specific operation specification. If permission-defined, then the rules of this section apply.

An example of the first method (permission-based) is the Object Access Protocol. Each of its operations requires certain permissions. An example of the second case (application-based) is the Ticket Request Protocol. Its operations do not specify required permissions, and so each operation must define its own rules for control.

As discussed in *Guidelines for DOF security*, the permission-based approach uses permission groups to relate permissions to each other. Permissions between nodes are only allowed to be compared within the same permission group. Each permission group is associated with a domain-unique permission group identifier.

Note that permission group identifiers are always associated with the permission-based approach, and they have no impact on the application-based approach. This section does not apply to application-based operation control.



Permission-controlled operations must be scoped by the permission group. `dof-2008-16-spec-12`

This means that operations are not allowed outside a permission group without the use of a security gateway. It is possible for an operation to execute at the same time in multiple permission groups depending on the implementation.

Permission groups (and permission group identifiers) represent groups of nodes in a security domain that are allowed to communicate. The communication is allowed through the granting of permissions to each of the nodes in the same permission group. Permissions may be granted in multiple permission groups, and the set of these groups is called the permission security scope.

Permission group identifiers are unsigned 30-bit numbers. They may be passed on the wire using a compressed format, and so only numbers from 0 to 2^{30}-1 are valid.



Permission group identifiers are non-negative and less than 2^{30}, with identifiers greater than 2^{29}-1 being reserved. `dof-2008-16-spec-13`

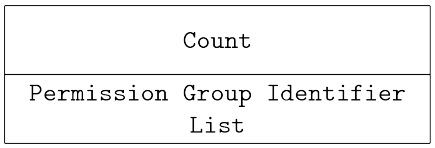
This restriction allows wire compression. This also means that smaller numbers are preferred to larger numbers. The reserved range is used for special flags.

All permission-based operations in DOF security must go through a permission check on the receiving node. The output of this check is not just a "yes/no" answer of whether to allow the operation, but rather is a set of permission group identifiers in which the operation is valid. The result is a set because the operation may be allowed by multiple permission grants in different permission groups, and because the same permission may have been granted in multiple permission groups. The rules for working with operations are discussed later. The set of permission group identifiers associated with an operation is called the operation security scope. Note that since communication is always restricted to a single domain, an operation security scope always will be limited to a single domain.

Security scopes are always passed in a context where the domain is understood, and so the domain identifier is not included. Only a list of permission group identifiers is required.



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`.

In order to efficiently manage permission requests on proxy nodes, DOF security defines a node security scope. The node security scope is the **union** of all permission group identifiers associated with a node.

For example, many proxies will automatically request permissions on sessions when they desire to forward operations. Without knowledge of the node security scope, permissions would need to be requested on *all sessions in the domain*. Doing that would yield the correct result, but most of the sessions would fail to negotiate the permission. Since this situation arises during forwarding, the security scope of the operation being forwarded is known. Comparing the operation security scope against the node security scope of the session being considered will narrow down the set of possible sessions. Requesting the permission may still fail, but many failures are avoided by comparison of the node and operation security scopes.

5.1. Compatibility

It is possible for an operation to be associated with multiple permission group identifiers, as discussed earlier. Compatibility refers to how the operation security scope restricts where the operation can be forwarded and who can respond. An operation that is associated with a single permission group is limited to that group, meaning that a single permission group identifier is only compatible with itself. This section applies only to operations that are controlled by permissions.

When operations are passed over a session, it is possible to attach security scope information. In all cases, the effective security scope of the operation can only be further restricted by the sender, and the receiver must validate the security scope of the operation during the permission check. This means that the effective operation security scope will be the **intersection** of the security scope passed on the wire and the security scope determined by the receiver.

Note that the effective operation security scope will always be a subset of the connection security scope. Further, note that the effective security scope will always be associated with a single domain.

During permission validation the authentication server evaluates the node security scopes of the involved nodes, resulting in a session security scope determined by computing the **intersection** of the node security scopes. Permission checking can then be done only in permission groups that are part of the session security scope.

There are several permission group identifiers that are used as flags and have special meanings. This table includes them all.

Flag	Description
0x3FFFFFFF (2^{30}-1)	Defines compatibility with all permission group identifiers. Can appear in permission, node, and session security scopes (although the intersection operation of the security scope restricts the cases to where both permission security scopes use the flag). When passed in a permission validation block it is a shortcut for the corresponding session security scope.
0x3FFFFFFE (2^{30}-2)	Indicates that an aggregate permission does *not* mask any other permission grants. This is passed as a flag in the permission security scope. See the discussion on permission validation (below) for more information on permission masking.
0x3FFFFFFD (2^{30}-3)	Defines a permission grant in the node security scope, without reducing or extending that scope. Note that when validated this flag will be reduced to the corresponding session security scope. This means that the permission validation block can always use the 0x3FFFFFFF flag in its place. However, that flag has a very different meaning in a permission grant and so this additional flag is required.

With respect to the "all permission groups" flag (0x3FFFFFFF), note that there is a critical difference between the uses of this flag in a granted permission security scope and the node/session security scopes. In the granted permission security scope a single use of this permission group would override any other more restricted grant. This means that the effective granted permission security scope of any permission that includes this flag is just this flag itself – it takes the place of all others. However, the same is not true in the permission validation list, node, and session security scopes, where this permission group identifier can appear like any other.

Examples of permission security scopes are (assuming the domain identifier is OID_1):

- S_1 - [OID_1: {2,7}]
- S_2 - [OID_1: {0x3FFFFFFF}]
- S_3 - [OID_1: {3,6}]
- S_4 - [OID_1: {6,3,2}]

Compatibility involves set intersection, and is independent of order. This means that the ordering of elements in the set has no impact on the result. Using the previous example security scopes there are 6 comparisons possible:

- $S_1 \cap S_2$ = [OID_1: {2,7}] $\cap$ [OID_1: {0x3FFFFFFF }] = [OID_1 : {2,7}]
- $S_1 \cap S_3$ = [OID_1: {2,7}] $\cap$ [OID_1: {3,6}] = [OID_1 : $\varnothing$]
- $S_1 \cap S_4$ = [OID_1: {2,7}] $\cap$ [OID_1: {6,3,2}] = [OID_1 : {2}]
- $S_2 \cap S_3$ = [OID_1: {0x3FFFFFFF }] $\cap$ [OID_1: {3,6}] = [OID_1 : {3,6}]
- $S_2 \cap S_4$ = [OID_1: {0x3FFFFFFF }] $\cap$ [OID_1: {6,3,2}] = [OID_1 : {6,3,2}]

- $S_3 \cap S_4 = [OID_1: \{3,6\}] \cap [OID_1: \{6,3,2\}] = [OID_1 : \{3,6\}]$

This indicates that permission based operations are allowed between node S_2 and all other scopes, S_1 and S_4, S_3 and S_4 because they have non-empty results.

Permission-controlled communication must always be between nodes where the session security scope is not empty. `dof-2008-16-spec-14`

When permission sets are requested, the resulting permission grants may be empty if there are no permission groups in common.

The handling of unauthenticated communication is up to the application. Applications may bridge unauthenticated data into a secure system, but in doing so, they must determine immediately what the security scope of the unsecured traffic will be. Once this assignment is made then the traffic is treated exactly like other secured traffic.

Unsecured communication must be separated from secured communication unless configured otherwise. `dof-2008-16-spec-15`

Unsecured communication is not associated with a domain unless through a security gateway.

With the general information detailed above, the following relates to the understanding of security scopes between two communicating nodes:

- Although the definition of a security scope always includes a domain identifier, the domain identifier is rarely communicated on the wire. It either is assumed (implied on the wire) or is passed separately (as in the security protocols). During the establishment of a session, the domains involved, including the authenticated domain of each communicating node is established.
- Inside the same domain, permission group identifiers and security scopes have the same meaning. Two nodes that authenticate using the same domain can assume that permission group identifiers can be compared. This extends to other authenticated sessions *only if the domains are verified to be the same*. See the note below.
- Authentication always results in at least knowledge of the session security scope. This provides a base understanding for every session.

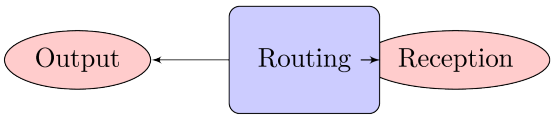
Each secure session must be assigned a session security scope. `dof-2008-16-spec-16`

This assignment is made by the domain through the authentication server during key distribution. The session security scope is the intersection of the node security scopes associated with each node.

5.2. Security Scope Usage

This section defines the requirements for obtaining and forwarding security scopes. The rules for mapping an operation to a security scope are presented in the section on access control. Note that these requirements apply only to operations that are controlled by permissions.

Figure 5.1. Different Phases in Security Bridging



Conceptually there are three phases to each communication. Each of these phases is presented and the requirements for each are identified.

Although these discussions present security scopes in a unified way, it is important to note that each operation exists in each permission group identifier separately. This is particularly difficult when the same operation arrives over different sessions, each with a different security scope.

Even if a single datagram arrives, if the operation is assigned to multiple permission group identifiers then it must be treated as if multiple copies of the operation arrive. This requires that each of these operation "copies" can have its own source, and that the DOF Presentation Protocol and the application protocol must treat each copy as distinct.

However, a literal implementation of this "copy" logic would result in inefficiencies that are not required. Balancing the "correct" handling with "realistic" handling takes careful attention to all of the required behavior for the operation, ensuring that no specifications are violated while still maintaining performance.

Operations are handled in each permission group identifier as if received independently. `dof-2008-16-spec-17`

This requirement means that the operations must maintain lifecycle and source in each permission group. Implementations should optimize this behavior as the typical case is for an operation to arrive, be assigned multiple permission group identifiers, and remain in those permission group identifiers during the entire lifecycle of the operation.

The use of separate sources for each permission group identifier may cause duplication of responses if the permission group identifiers are aliased through the use of application mapping. This can be minimized if providers are consistently assigned to a single permission group identifier.

Nodes that proxy operations are free to consolidate the permission group identifiers of inbound operations (using the same operation identifier) when forwarded. However, the nodes must preserve the permission group identifiers on responses, and only forward them to the appropriate source.

Operation security scopes can be merged by proxy nodes, but responses must remain in their assigned security scopes. `dof-2008-16-spec-18`

This requirement means that a single operation that arrives over two sessions where each assigns different operation security scopes can be merged by the proxy node and treated as a single operation for output. However, responses to those operations must remain in their assigned operation security scopes and cannot be forwarded to all sources. This logic may cause responses to be duplicated in some cases.

5.2.1. Receiving an Operation

As discussed earlier, an operation may have an associated security scope when passed on the wire. In this case, the effective security scope is the intersection of the security scope determined by the receiver as part of permission validation and that passed on the wire.

The result will always be a subset of the session security scope, which is the intersection of the node security scopes involved in the session.

Permission group identifiers are only valid in their assigned domain. `dof-2008-16-spec-19`

This means that they are comparable only in that single domain. This limits security scopes to a single domain.

Security group identifiers passed on the wire must limited to the operation security scope as determined by the receiver. `dof-2008-16-spec-20`

This limits the communication to the security scope determined by the receiver.

Security scope information received from a foreign domain (determined during session establishment) may be passed only to other members of that domain. This is a special case of the rule about domain communication. In this case, the identity of the foreign domain was validated by the domain itself, and so the security scope information can be trusted only *by the foreign domain*.

The following applies when communication is received. The session has already been assigned to a domain, denoted D_C. The operation may include security scope information, denoted SS_W.

The methods used to obtain the security scope are not discussed here. More information can be found in the security protocol specifications.

In all cases, the result of reception is the determination of $[D_C:SS_R]$. The determination of SS_R is made by the receiver as $SS_W \cap SS_{OP}$. SS_{OP} is defined in the section on access control.

5.2.2. Routing

Routing in this context is the process of determining the sets of sessions available to the node that are compatible with the operation. Remember that the process of receiving the operation resulted in the determination of SS_R.

Each session on the node is assigned its own $[D_T:SS_T]$, as discussed above. Logically, the router computes $SS_R \cap SS_T$ for each session where $D_R = D_T$, and builds a set S of sessions where $SS_R \cap SS_T \neq \varnothing$. Note that this will always be true for the source session, and it must always be excluded.

The communication, including $SS_R \cap SS_T$, may be forwarded to each session in the set S. The specific rules for routing depend on the operation and application protocol. These rules define the maximal set S over which the communication may pass. It is known that it may *not* pass to any session that is not included in S.

5.2.3. Output

The previous section determined the set S, which is the maximal set of sessions to which an operation may be forwarded. It was briefly mentioned that the security scope of the operation might change due to the

evaluation of $SS_R \cap SS_T$. A proxy may determine whether to pass security scope information on output. Unless the scope is being restricted in some way, there is no advantage in passing the information on the wire, as the receiver must always determine the security scope on its own.

It is possible that additional permissions are required on a session in S before the operation can be forwarded. It never makes sense to forward an operation over a session that does not already have the required permission – the receiver will reject the operation.

A final check must be applied before sending an operation to ensure that the receiver's understanding of the operation security scope is not larger than the sender's. For example, an operation in security scope A may be passed over a session with session security scope A and B. Depending on the permissions, the operation may be "expanded" to include B by the receiver. This behavior is known by the sender, who has access to all the same permission information. In this case the sender is required to pass the operation security scope on the session.



Senders must ensure that the operation security scope is not expanded by the receiver. `dof-2008-16-spec-21`

This prevents an operation from leaking from one security scope to another, and keeps it limited to the security scope that it originated in.

A special case of this rule is that operations that are in *all* security scopes (0x3FFFFFFF) must always be specified on the wire. These cases are rare, and so this requirement does not usually incur a large penalty. This requirement makes these special operations very clear to the receiver.



Operations using the global security scope must be explicitly marked, even when the sender and receiver agree that no expansion is occurring. `dof-2008-16-spec-22`

These types of operations are rare and potentially very powerful. This requirement makes them explicit, forcing both sides of the session to apply special logic.

6. Permissions and Access Control

Access control in DOF security is a combination of many of the topics already discussed. This section discusses the general theory and formats that govern access control, but it is up to each application protocol to define the specific permissions that control its own operations.

Permissions represent any kind of authorization that can be requested and granted. The DOF definition of permission would include typical permissions (like access permissions), but also a broader range of possibilities. Permissions typically do not include administrative features, like "Administrator" as these functions are typically not involved with DOF application protocols. However, the AS may have a sense of these types of permissions and may map them to a corresponding set of DOF permissions.

Permission sets are groups of permissions that can be requested and authorized as a group. Permission sets form the basis of access control. Each secure session has a permission set associated with each node. Permission sets are mostly logical, although the term is applied to groups of permissions as they are exchanged on the wire as well as the logical groupings of permissions that are associated with a session.

Permission sets are always established in the context of a node and some other node or set of nodes, and are always associated with sets of security scopes. The application of security scopes was discussed previously. Permissions are used to map operations that they allow to a security scope. The security scope is a set of permission group identifiers that are shared between the communicating nodes and associated with the required permission. In this case, each permission group identifier in the security scope identifies nodes that are compatible with the operation.

Permission grants are always associated with an expiration time. It is typical for the initiator to request the amount of time that the associated session will exist as the proposed expiration time, and for the AS to respond with the actual expiration time. DOF security does not provide permission revocation except through expiration, and so the maximum expiration time allowed is a key aspect of the system security policy. Since these times affect the implementation, they are supported by the different security protocols directly.

It is possible that the required permission to send a command and that required to send a response are different. This means that both the requestor and provider will usually negotiate permissions at the same time (as part of secure session negotiation). Each of them negotiates a permission set, and the combination of the two permission sets governs the communication. This allows (but does not require) that permissions may be granted "between" nodes, or at least can be granted or denied based on the knowledge of both nodes.

Note that protocols usually deal with these simultaneously negotiated permission sets separately, and not as a 'combined' permission set.

The rules for identifying the appropriate permission sets are complicated by the fact that the security scope associated with an operation does not always contain a single permission group identifier. In the case of multiple permission group identifiers, the operation must be evaluated in the context of each. Of course if permission cannot be obtained then the security scope will be empty and the operation rejected.

Permissions are integral to many different DOF application protocols, and it is important that the permission structure be generic enough that an application can skip over permissions that don’t apply to it without missing permissions that do apply. This needs to be done without relying on knowledge of the specific format for each permission type.

In order to accomplish this without waste of space (in memory or on the wire) there are two types of permissions. The first type has no associated data, and so if it is not understood then it can be directly skipped. The second type has associated data, and so a length must be present to allow the data to be skipped if not understood.

Each permission type is assigned an identifier, which is passed in either one or two bytes. Odd permission identifiers have associated permission data; even permission identifiers have no associated data.

6.1. Permission to Connect

Sessions are not controlled by permissions. At first, this may seem counter-intuitive, but in fact, DOF security allows any two nodes to connect to one another as long as they can authenticate correctly. This allows operations that do not require any permission to be processed.

6.2. Permission Storage

The domain is the container of all permissions that have been granted to a node. Permissions are "granted" to a node in the context of a permission group, and so each granted permission is associated with at least one permission group identifier.

This is the meaning of the term "granted" permissions – that the domain has associated the permission with a node.

6.3. Aggregate vs. Terminal Permissions

Permissions are grouped into two major categories: aggregate and terminal. This grouping does not refer to the permission types, but to how individual permissions are compared.

A terminal permission is any permission that allows only itself when considering only permissions of the same type. This means that a terminal permission may allow some permission of a different type (and still be terminal). An aggregate permission is any non-terminal permission, which means that all aggregate permissions allow at least two permissions (itself and at least one other) of the same permission type. In addition, this definition relates only to permissions that utilize permission group identifiers.

For permissions of the same type that utilize permission group identifiers, if P_1 is allowed by P_A, and P_2 is allowed by P_A, and P_1 is not P_2, then P_A is an aggregate permission. Whether a particular permission type allows aggregate permissions depends on the definition of the permission type.

The validation process for aggregate permissions is different than that for terminal permissions.

6.4. Requesting Permissions

A permission request is always associated with the distribution of a key. The key distribution is used to validate the request and authenticate the response so that all nodes involved have full trust in the permission validation. The specifics of this validation are discussed in the key distribution protocols.

The validation of a permission request associates a security scope with each permission in the request. The nodes must then enforce that operations associated with the permissions are assigned the correct operation security scope, and then that the operation security scope is correctly enforced as discussed earlier.

Permission sets that are validated represent subsets of the entire permissions granted. In this sense they are "partial" or "incomplete" permission grants. Much of the following information on masking and making additional permission validation requests is due to this incomplete knowledge of the entire set of permission grants. In a worst-case scenario an implementation that cannot validate additional permission grants can treat the set of validated permissions as if it represented the entire set of granted permissions.

Each operation in a protocol that is governed by permissions will have an associated permission that is required in order for the operation to be accepted. If this permission has not already been obtained then a new permission request must be made before the operation can be executed. Note that a single operation may utilize multiple protocol layers, and that each layer may require some set of permissions for the operation. All permissions that are required by an operation must be terminal permissions.

Operations must only require terminal permissions. `dof-2008-16-spec-23`

All operations that use permissions must require only terminal permissions.

The logic for accomplishing permission validation is not as simple as it sounds because of the complex nature of permission grants, operation security scopes, and session security scopes. It is important that all implementations utilize consistent processes for dealing with permissions, and so the following sections will discuss standardized methods and terminology.

There are several concepts that are consistent across all permission types that act as the foundation for later discussions on validation. These concepts are presented here.

6.4.1. Normalization

Procedures involving permissions may require that permissions be "normalized".

A normalized permission strives for a balance between efficiency of space and complexity. Each permission type defines what it means to "normalize" itself. While the rules for creating a normalized permission are explicitly defined, permissions do not need to be stored or passed in this format. Permission checking against any permission must generate the same result as it would if evaluated against the same permission after normalization.

All terminal permissions are normalized. Not all aggregate permissions are normalized.

The definition of normalization logically results in a new permission set. Further, normalizing a permission may result in a different set of permissions. This specification uses the notation |P| to represent normalization. As stated above, |P| results in $\{P_N\}$ or in other words normalizing a permission results in a set of normalized permissions. Normalizing a permission set results in a set of normalized permission sets, meaning |{P}| results in $\{\{P_N\}\}$.

Permission checks must be effectively made between normalized permission sets. `dof-2008-16-spec-24`

This means that the results of all permission checks must be the same as if they had been made against normalized permission sets.

Each permission type defines its process of normalization.

Permission normalization must not extend permission scoping. `dof-2008-16-spec-25`

Normalization must never result in permissions extending their grant either between security domains or between security scopes.

6.4.2. Complementing Permissions

Each permission type defines a mapping of itself to another permission (or set of permissions) called the "complement". The purpose of the complement is to identify permissions that are the "opposite" of the given permission. These can be used during three-party authentication by the responder as an optimization.

Permissions are free to define their complement in any way, including mapping to nothing. Whatever the complement is, the permissions it includes must be granted to the other party in order for communication to occur.

6.4.3. Comparison of Permissions

Comparison on permissions does not result in an ordering, but rather in a determination of "allowed by". This comparison is not associative, but is transitive. "Allowed by" is indicated by the symbol "#". This symbol indicates the lack of associative properties. However, note that the use of this symbol is not meant to imply an ordering in the sense of "less than".

"Allowed by" for permissions is defined for each permission type. In order to guarantee that the meaning of a permission type is fully understood, each permission type is associated with an application protocol version. Application protocol versions may be discovered or negotiated, and part of the acceptance of an application protocol version is the enforcement of all permission types associated with that application protocol version.

This means that it is always acceptable to use related permission types that are associated with a single protocol version. It is also acceptable to use related permissions between protocol versions that are negotiated or discovered in the same session. However, it is never acceptable to negotiate a protocol and then use permissions associated with some other protocol that was not negotiated.

Note, however, that there are many situations in which a node receives packets that may contain permissions but that are not directly related to the node itself. For example, a node that is routing traffic between other nodes may need to forward permissions for those nodes. In this case all permissions must be supported. This is acceptable because the node itself is not responsible to enforce the permission, only to forward it.

Another effect of these rules is that an implementation must always understand the permission requests that it must enforce. A protocol must reject being requested to enforce a permission that it does not understand, and should reject begin requested to enforce permissions associated with protocol versions that were not negotiated in a given session.



Nodes must reject requests to enforce unrecognized permission types. `dof-2008-16-spec-26`

Permission types are negotiated indirectly, as they associated with protocol versions that are negotiated. Nodes cannot enforce permission types that cannot be understood by the nodes involved. Nodes should reject requests to enforce permission sets that include permission types that are not understood. Note that nodes must forward all permissions, whether understood or not, in contexts where they are not being asked to enforce the permission (for example, a routing node). In particular, note that permission comparison is not the same as permission enforcement.

Note that there are several ways in which the $\leq$ operator can be used. Of these, the only way defined by the permission type is $P_N \leq P_G$, where P_N is a normalized permission and P_G is any permission (whether terminal, aggregate, or normalized).

The result of $\leq$ is always { PGID }, or the set of permission group identifiers in which the permission is allowed. The following additional computations extend the use of '#' to non-normalized permissions and permission sets. The methods are defined as follows.

6.4.3.1. $P_N \leq \{ P_G \}$, Normalized Permission Allowed by Permission Set

In this case { PGID } is computed as the union of $P_N \leq P_G$ for each P_G in { P_G }. This has the effect of computing the set of all permission groups allowed by any permission grant.

6.4.3.2. $\{ P_N \} \leq \{ P_G \}$, Set of Normalized Permissions Allowed by Permission Set

In this case {PGID} is as the intersection of each $P_N \leq \{P_G\}$. This has the effect of limiting the permission groups to only the common permission groups of requested permissions. Note that this is the typical comparison for an operation, as each operation requires a set of terminal permissions. Terminal permissions are, by default, normalized.

6.4.3.3. $\{ P \} \leq \{ P_G \}$, Permission Set Allowed by Permission Set

In this case {PGID} is computed as $\{|P|\} \leq P_G$. {|P|} is defined as {\{P_N}}. Each {P_N} results in {PGID}, and in this case they must be intersected to find the final result.

6.4.4. Intersection of Permissions

Each permission type defines the intersection operation '#'. The intersection operation returns a Boolean result. For $P_R \cap P_G$, true is returned if there exists some P such that $P \leq P_R$ and $P \leq P_G$. Note that the operation does not need to actually determine P, only whether some P could exist.

The permission type is responsible to define $P_N \cap P_G$ only where P_N is normalized. This operation is used to determine the 'masking' flag during permission validation, and in this case a set of {PGID} needs to be calculated. The logic for computing this is described below in the sections on validation.

6.5. Permission Validation for Sessions

Permissions are validated either as part of secure sessions, as part of groups, or as a single node. These validations are discussed in the following sections. Validation of a permission set {P} always results in a validated permission set of the form { (P, {PGID}, masking) }, meaning that each permission in the original permission set has a set of permission group identifiers associated with it along with a flag indicating whether there are masked grants.

In all cases, validation of a permission P of an unknown type results in (P, {}, true). This means that the {PGID} result is empty, and the masking flag is set to true. The logic here is that an unknown permission type cannot be understood, and so it cannot be compared or granted. However, returning 'false' for the masking flag may cause the requesting node (who must understand the permission type) is make assumptions about other permissions that the validating node may understand if asked.



Validation requirements for unknown permission types. `dof-2008-16-spec-27`

Validation requests for unknown permission types must return an empty set of permission group identifiers and a masking flag of true.

6.5.1. Validation of 2-Node Secure Sessions

A 2-node secure session involves two nodes, neither of which is a group. Each node will be assigned a node security scope, and the secure session is assigned a session security scope that is the intersection of the two node security scopes.

Note that both the type of session (in this case a 2-node session) and the security mode of operation are available to the authentication server whenever it is requested to validate permissions. This information may be used to modify the effective granted permissions (P_{GRANT}, discussed below). For example, a given permission may only be granted if the session type is 2-node, and effectively not granted if the session type is n-node.

Logically, all permission checking is limited to the session security scope (C_{SS}). Checking permissions in other permission groups wastes effort, because the other node will never have the permission required to communicate in that permission group.

The process for assigning the security scope for a permission request involving two nodes consists of the following steps. Each node will provide a permission request – $\{P_{REQ}\}$ – that is handled identically. The result of this process is the assignment of a security scope P_{SS} to each permission in the request set. In addition there is a flag, 'masking', that is assigned either true or false.

Note that following procedure assumes access to the entire set of granted permissions. This is true for the AS, but may not be true for any other nodes.

For each P_{REQ} in the permission request compute the associated security scope P_{SS} and masking flag as follows:

- Create an empty security scope P_{SS} associated with P_{REQ}. Associate 'false' as the masking flag associated in P_{REQ}.
- Compute $P_{SS} = P_{REQ} \leq \{P_{GRANT}\}$.
- Set masking to true if there exists some P_{GRANT} where NOT($P_{REQ} \leq P_{GRANT}$) AND $P_{REQ} \cap P_{GRANT}$, and the permission groups of P_{GRANT} are greater than P_{SS}.

Note that the calculation of the masking flag can be done in parallel with the calculation of P_{SS} as follows. If the permission is not allowed by the granted permission but the intersection is true, then calculate the union of the granted permission groups. At the end of the comparisons for the permission, if the non-granted permission groups are more than the granted permission groups then set masking to true.

At the end of this process, there will be a parallel set of permission security scopes and masking flags for each requested permission in both permission requests. The result of the validation will include the intersection of both node security scopes, along with the determined set of permission security scopes corresponding to each permission request.

6.5.2. Validation of N-Node Secure Sessions

The process for assigning the security scope for a permission request between a node and a group consists of the following steps. In this case there is a single permission request, although the group has a permission grant that is used.

Note that both the type of session (in this an n-node session) and the security mode of operation are available to the authentication server whenever it is requested to validate permissions. This information may be used to modify the effective granted permissions (P_{GRANT}, discussed below). For example, a given

permission may only be granted if the session type is n-node, and effectively not granted if the session type is 2-node.

Logically, all permission checking is limited to the session security scope (C_{SS}). This is true even though the "session" is between a node and the group. Checking permissions in other permission groups wastes effort, because the other node will never have the permission required to communicate in that permission group.

During the validation the process will construct the session security scope C_{SS}, which starts out empty.

Note that following procedure assumes access to the entire set of granted permissions. This is true for the AS, but may not be true for any other nodes.

For each permission P_{REQ}, in the request:

- Validate the permission as for the 2-node session.
- Add P_{SS} to C_{SS}.

For each permission, P_{GRANT}, in the group's granted permission:

- Create an empty security scope P_{SS} associated with the granted permission (this is not related to the earlier P_{SS}).
- Set P_{SS} equal to the intersection of the granted permission scope and C_{SS}.
- If P_{SS} is not null then include P_{GRANT} in the effective group permission set.

At the end of this process, there will be a parallel set of permission security scopes and masking flags for each requested permission and an effective group permission set consisting of the related permissions. This effective group permission set will also have a parallel set of permission security scopes.

Note that the authentication server does not verify the relationship between the requested permissions and the effective group permission set, other than to verify that they share permission group identifiers are compatible with C_{SS}. This may result in a large effective group permission set, but prevents the authentication server from having to do a detailed comparison of permissions.

6.5.3. Validation of Single Nodes

The process used in this case is identical to that for a 2-node session, but only uses a single permission set for requests instead of one per node.

6.6. Permission Validation for Operations

Validation of operations must occur on receiving nodes and should occur on sending nodes. The validation on receiving nodes is to protect the node from malicious senders, and failure to validate results in the operation being dropped.

The validation on sending nodes is to ensure that the required permissions have been requested on the session in order to prevent the receiver from dropping the operation. The sender should apply the same permission checks as the receiver (described below), although if the permission validation fails the sender may attempt to negotiate the required permission (before sending the operation) rather than dropping the operation.

This typically means that the sender checks the permission set on the session, and if no match (qualified below) is found pending the operation while the session permissions are augmented. The specific behaviors in this case are defined by the application, which may be configured not to allow this kind of augmentation to occur.

Operation validation requires a validated permission set, which means that it has available { (P_G, {PGID}, masking) } from earlier validation. This means that the entire set of granted permissions is not known, and so the masking flag must be used. If a requested permission is not allowed by the current validated permission set then the application may extend the permission set (on the sending node).

Operation validation is done between $\{P_{OP}\}$ and a validated permission set. It computes SS_{OP} (which is {PGID}) by:

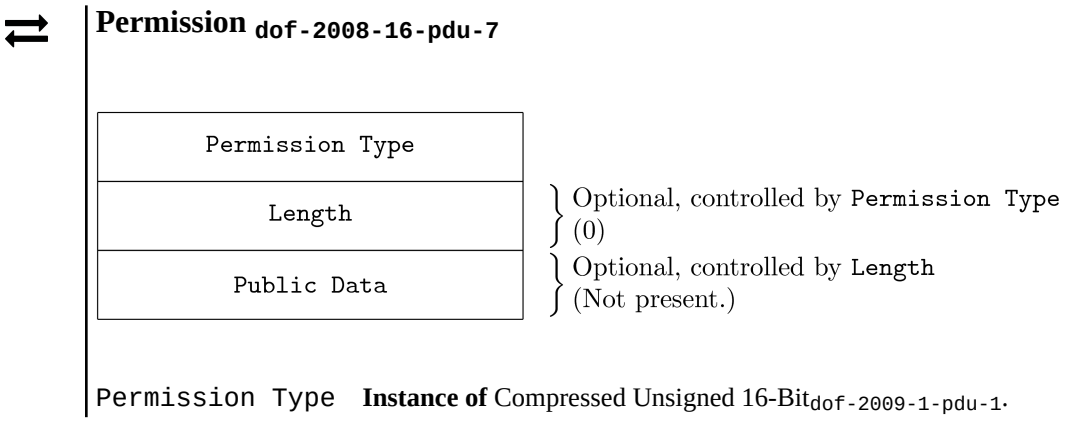
- Computing $|\{P_{OP}\}|$. In this case the individual permissions do not need to be kept separate, so this results in $\{P_N\}$ (a set of normalized permissions). Note that permissions required for an operation must be terminal, and so $|\{P_{OP}\}|$ should be equal to $\{P_N\}$.
- For each P_N, computing {PGID} by $P_N \leq P_G$ for each P_G in the validated permission set. If allowed by is true and the masking flag is set to true and the permission set can be extended, stop this process and request $\{P_{OP}\}$ directly.
- Compute SS_{OP} by taking the intersection of all the resulting {PGID} for each P_N.

Note that if a permission requires all permission grants to be known in order to correctly evaluate "is allowed by" (as indicated by the specification of the permission type) then the logic defined above must not be used, and the permission in question must be directly requested. Implementations may choose to cache these specific requests much as they do the validated permission sets. If, at the end of the process, SS_{OP} is empty then the operation has failed its access control checks and is not sent.

6.7. Permission and Permission Request Formats

Each permission format must be understood by multiple applications and protocols, including the domain, so the format is centrally defined. Revising this format would require simultaneous revision of all protocols that work with the permission format, which would be very difficult. This is one reason for the rather generic format.

Each permission follows the same format:



`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**.

The **Permission Type** identifies a specific permission. Types that are frequently used should be assigned type numbers less than 128 because those values can be passed in one byte.

If the **Permission Type** is odd, then a **Length** field follows. A **Length** of zero is valid, and indicates that there is no **Data**.

The **Data** is a type-specific block of information describing an individual permission. This format allows skipping permission types that are not understood.

6.7.1. Permission Types

Permissions are formatted on the wire according to their permission type.

For a current list of registered permissions, please contact the ODP-TSC or visit https://opendof.org/registry-permissions.

Permission types must be registered with ODP-TSC before use in product. `dof-2008-16-spec-28`

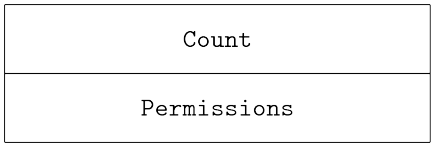
This includes the definitions of the operations that they control as well as the wire format and meaning for each permission. Once assigned, the formatting, meaning, and comparison rules of a permission must not change.

Ignore the reserved range of permission types. `dof-2008-16-spec-29`

The reserved range is for management application use. They should not appear in permission sets, but if they do they must be ignored.

Permissions are grouped together in a permission request as follows:

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



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

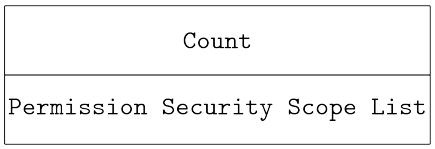
`Permissions` **Array of** Permission`dof-2008-16-pdu-7`.**Optional, controlled by** `Count`. Defines a set of permissions. The ordering of permissions inside the request does not affect the meaning and must not influence the result. However, the ordering must be stable because the security scope results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

6.8. Security Scope Results

As discussed above, the validation process of a permission request results in a set of permission security scopes. This information must be passed back to the nodes involved, which takes place in a key distribution request and is controlled by a key distribution protocol.

The format of the response is defined as follows:

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

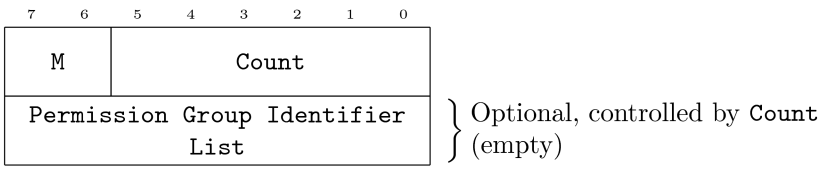


`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls length of **Scopes**.

`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.

A permission validation is included in responses from a domain. It always corresponds to a permission request. The number of permission security scopes in the permission validation must match the number of requested permissions in the permission set. Note that this does not mean that the validation **Count** must match the permission set **Count**, because the validation **Count** allows for blocks to be extended (taking more than a single slot).

⇄ Permission Security Scope `dof-2008-16-pdu-10`



`M` **Two bits.** Controls presence of **scopes**, defines the method used to encapsulate data in the list.

`Count`	**Six bits.** The associated count, with some special meanings depending on the value of **M**.
`Permission Group Identifier List`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `Count`. This list is only present if **M** is not zero.

A permission security scope is included for each permission in an associated permission set. The PDU defined above is a single frame, which may or may not be equivalent to a single permission security scope as defined below.

Special flags may be included in this list to indicate features of the permission security scope. The "no mask" flag (0x3FFFFFFE) is one of these, which is used on aggregate permissions to indicate that the request does *not* mask any other granted permissions.

The following values of **M** are defined:

M Value	Description
0x00 (0)	*Reference*. This value indicates that the permission security scope is the same as that of the permission that appears **Count** + 1 elements previous. In this case, **Count** does not indicate the length of the *Permission Security Scope*, which is not present for this value.
0x01 (1)	*Relative*. This value indicates that the **Permission Security Scope List** is present and contains **Count** entries. Each entry that is less than 2^{29} is a relative index into the *Session Security Scope*, which must exist in the same context as this PDU and contains the actual permission group identifier. Entries that are greater than or equal to 2^{29} are not indexes, but are the permission group identifiers themselves.
0x02 (2)	*Absolute*. This value indicates that the **Permission Security Scope List** is present and contains **Count** entries. Each entry is an individual permission group identifier.
0x03 (3)	*Continued*. This value indicates a continuation of the previous *Relative* or *Absolute* frame's **Permission Security Scope List**. It is only valid if the previous **Count** value of the previous frame is equal to 63. Further *Continued* frames are allowed until one includes a **Count** less than 63.

7. Securing Communications

This section defines the methods that DOF security uses to secure communications. As with other aspects of DOF specifications, these methods are, for the most part, independent of transport. This means that single transport sessions can usually be leveraged by multiple DOF sessions. Even so, the transport characteristics do have an impact in the security methods that are allowed.

There are two main issues involved in securing communications: state management and key distribution. Each of these issues is dependent on the transport characteristics of the communication in the following ways:

- Security state requires that a session exist. This session can either come from the transport itself or can be provided using DOF protocols. However, if the session exists at the transport layer DOF protocols are able to create multiple security sessions, and so in the context of DOF security the session is always a function of DOF implementations.
- Security sessions are always associated with key distribution. The behavior of the different key distribution protocols may be modified by the transport session characteristics. The key that is shared among all session members is typically called the "session key". All session keys are associated with an expiration time. This expiration time is typically requested by the initiator as the intended duration of the session, and the AS then distributes the actual expiration time as controlled by the system security policy. The session key expiration time is a key aspect of the security policy and so it is supported directly by the different security protocols.
- Key distribution requires knowledge of the identities of the parties involved in the session. These identities are integral to the session itself, and must not change for the duration of the session (or a node’s participation in the session).
- Once a secure session is created, session identification must be associated with each datagram transferred on the session. It is possible that the security state information in a datagram comes in part from state associated with the transport session.
- The datagram security state is always divided between common state (provided outside of a specific security mode of operation) and mode-specific state (always managed by the security mode of operation).
- All security sessions either point-to-point or multipoint. However, within a multipoint security session an individual datagram may be identified as point-to-point. However, in a more general case it may also be multipoint-to-point. This terminology indicates that the target is well defined, but the source may be ambiguous.

The different uses of DOF key distribution and security modes of operation are now presented.

7.1. General Secure Session Considerations

When a session is requested by the application, there is an expectation that the communication will either be secure (whether encrypted or not) or unsecured. It is understood that during negotiation, the session will not be secure (unless security is provided by the transport itself).

This means that it is only after security negotiation is complete that the expectation of the application can be met. It is therefore only after security negotiation completes successfully (if required) that the session is said to be established and useable for its intended purpose.

Each protocol must identify its possible uses during these different phases of session establishment. Some protocols have uses both during negotiation and on an established session, others do not.

Many protocols will specify that they are only available for use on established sessions (including the special "None" session, which precludes security). By stating this, these protocols imply that they are not available during the negotiation phase of the session, and that their security requirements follow that of the session. In other words, they can be used either secure or unsecured based on the session, but they are not allowed on the session until it is established. Other protocols will specify the session requirements for each command.

Once a secure session is established, only secure datagrams may be exchanged on the session. There is a single exception to this requirement, which is that security modes of operation may exchange unsecured datagrams even on a secure session.

Secure sessions must reject all unsecured datagrams except those belonging to security modes of operation. `dof-2008-16-spec-30`

This means that once enabled, all datagrams must be secured. The range of APPIDs from 0x6000 to 0x6FFF (security modes of operation) are the only exception.

Security state must not be shared outside the session. `dof-2008-16-spec-31`

Security state is scoped to the session. Security state, particularly state that is not public, must never be shared or synchronized outside the session itself. As an example, synchronizing keys between sessions is prohibited.

7.2. General Key Distribution Considerations

As mentioned above, all secure sessions are linked to the distribution of a session key. It was also mentioned that each datagram on a secure session must be associated with some security state, part of which may be general and part of which depends on the security mode of operation.

Key distribution uses cryptography, and utilizes knowledge of the identities of the parties involved. A general requirement is that the identity that a node uses in a secure session must not change as long as the node’s relationship with the session remains. In a two-party situation, this implies that a change of node identity must cause the session to terminate. In a multipoint session a node must leave the session to change its identity, although in practice it is difficult to detect this case because the node can just rejoin the session with its new identity.

Node identities must not change during their participation in a session. `dof-2008-16-spec-32`

Two-node sessions must verify this by each node checking (during rekey, etc.). In the multipoint session, there is no easy way to validate since the new identity looks like a new node joining the session.

The distribution of required security state is partially discussed in this document in terms of *general* state that applies to all security modes of operation.

However, there are general classifications that can apply to the different security modes of operation. In the specific key distribution protocols this state is abstracted because it is the security mode of operation that defines it but the key distribution protocol that distributes it (see Security.9: Security Mode of Operation Initialization).

In fact, this single PDU often contains a combination of different types of initialization information. Some may apply to all nodes in the session, some may apply to the distributor of the state, and some may apply only to a single receiving node. Further, some of the information may be associated with the session key, to the session itself, or to any number of other mode-specific properties.

The security mode of operation manages all of this different initialization information and ensures that it is distributed correctly. For example, it would be improper for an initialization PDU that contains receiver-specific data to be multicast to an n-node session. This implies that the security mode of operation must understand the context of the datagram that it will use to distribute its state, and specify the initial state information appropriately.

7.3. Point-to-Point Security

Point-to-point security is always associated with a session. The session may be based on a transport session or session, but may also be created by a DOF implementation using a datagram server. In both cases the security session is created as part of the initial key distribution request. This security session, once created, always identifies two agents, and all datagrams are transport unicast (once the security mode has modified the datagram properties).

The establishment of a point-to-point security session is always authorized by the authentication server and requires authentication of the two agents. However, the particular security mode use is dependent on the transport session characteristics. If the session is lossless, then the session can rely on synchronized state. However, if the session is lossy then additional datagram security properties will be required. The transport characteristics must be known by both agents at the time that the security session is established.

A single transport session can "contain" multiple security sessions, each maintaining its own state. Each of these is managed by the key distribution protocols individually, and is logically viewed as a session (because the lifecycle is managed, see below). The different sessions and state are uniquely identified by the security state identifier, established when the session is started.

The security state identifier is unique to the associated server in the case of a datagrams or the associated transport session in the case of sessions.

7.3.1. Key Exchange

Key exchange is managed through any of the key exchange protocols. Key exchange controls the creation of new security sessions. All key exchange requests must either reference no security state identifier (in which case the request is for a new session) or an existing security state identifier (in which case the request is in the context of an existing session).

7.3.2. Lifecycle

The lifecycle of a point-to-point security session is directly tied to the session key associated with the session: when the key expires so does the session. While the session is established the client can continue to rekey, and by doing this will extend the session. However, if the client does nothing then the state will be lost and the session terminated.

This means that stack layers above a point-to-point security session always see a connection (meaning that they will be notified of termination by the security layer).

The different key exchange protocols may have methods to end a session before key expiration as well.

Each session on a single transport session maintains its own lifecycle. When no sessions remain then the client may close the transport session. If no sessions exist for longer than the startup delay of a single

session (based on the negotiated key distribution protocol) then the transport session may be terminated by the server.

7.3.3. Permission Definition

Permission management is handled with key exchange protocols.

7.3.4. Timing

Key timing is handled though the key exchange protocols.

7.3.5. Replay Attack Prevention

The security state of each end of the session must include information to prevent replay attacks. The specific methods used depend on the security mode of operation used on the session.

7.3.6. SID Security

Point-to-point SID use uses negotiated permissions. The IAM permission must be negotiated independently by both sides of the session. Any ACTAS permission must also be negotiated.

7.4. Multipoint Security

Multipoint security refers to situations where the security state associated with a session is shared by more than two agents. This causes difficulties associated with consistency and authentication, and additionally makes replay detection much more difficult.

DOF security recognizes two different types of multipoint security: multicast and point-to-multipoint.

Multicast security refers to systems of nodes that securely communicate directly with each other, where the communication can originate at any member of the group and the target can be either all members of the group or a single node in the group.

Point-to-multipoint refers to a situation where a single node desires to communicate with many other nodes, but does not want to incur the overhead of maintaining separate security session state for each node.

The benefit of multicast is reduction in the security state required for many nodes to communicate, and the ability to reduce traffic on the network by communicating with groups of nodes simultaneously.

The benefit of point-to-multipoint is solely the reduction in security state required.

The issues surrounding any multipoint security are complex. Providing security in multipoint sessions requires solutions to the following problems:

- Key Exchange. Members of the group must be able to obtain the same encryption key as other members of the group.
- Permission Definition: Each member of the group must understand the rights of the group as a whole.
- Timing. Keys must have synchronized expirations.
- Group Maintenance: When nodes are removed from the group, the system must guarantee that the nodes are no longer able to communicate with the group within a bounded period of time.

- Replay Attack Prevention. Since multiple nodes share the same key and the state is distributed, some method to prevent replay attacks must be provided.

The following sections detail the solutions provided by the DOF specifications to each of these problems.

7.4.1. Common Solutions

The following solutions apply equally to multicast and point-to-multipoint groups.

7.4.1.1. Key Exchange

DOF security provides for centralized key management through the Authentication Server. The AS uses key distribution protocols to distribute keys.

With respect to multipoint security, the responsibility of the AS is to distribute a Key Exchange Key (KEK) to each authorized member of a group. The AS does not directly manage the entire security state of the group, but delegates this responsibility to the group itself. The KEK is meant to change infrequently. In fact, the KEK is required to change only when the AS group security state changes. In other cases, the KEK should not change. In particular, there are no security issues with having a KEK that never changes – in other words the KEK does not "age".

If a node cannot obtain the KEK for a group then it is impossible for it to join the group. Thus, entrance to the group is managed by the AS.

7.4.1.2. Permission Definition

The ability to share a session key limits the ability of the system to enforce access control. This is because the shared key makes it difficult or impossible to identify the source of a datagram with any authenticated certainty. However, this does not mean that it is impossible, and some modes of operation may be able to distinguish the source of a datagram with certainty. The target of a datagram can usually be uniquely identified (or at least targeted without ambiguity). This means that it is up to the security mode to determine the type of source (single/group) and the type of target (single/group).

If possible, the access control should be handled correctly based on the source and destination.

In the AS group permissions are managed differently than point-to-point permissions. The node requesting access to the group has its associated permissions, and the request is validated against them. However, there is no identified, single node that the AS can check the permissions against – it can only use the permissions of the group. Further, the node requesting access cannot rely on the other nodes in the group correctly enforcing permissions (and it has no context to manage each relationship independently), and so it can only assume that it is communication with the "group". Because of this, the AS compares the permissions requested by the node against the permissions of the group and will not grant permissions that don’t match those of the group. Further, the key distribution will indicate to the node the corresponding permissions of the group, and the node is expected to enforce those permissions.

When a command arrives, and unless the security mode of operation can uniquely identify the source, the source must be treated as the group. In the same way, operations going to the group must be valid for any member of the group unless the security mode of operation can authenticate the particular target node with its specific permissions.

7.4.1.3. Group Management

DOF security manages secure groups in a distributed fashion. First, the AS is used to distribute parameters that are common to the group, including timing parameters and the KEK. From that point the group itself manages distribution (or determination) of the group key and the nonce header (described below).

It is up to the key distribution protocol to further define how the group is managed. This may include the use of a single group manager, or may use other methods.

7.4.1.4. SID Security

Multipoint SID use is controlled by the AS, and the SID for each group member is assigned as the AS grants the node permission to join the group. Each group member must use this assigned SID, allowing each node to validate the SIDs that are received from other members.

Operations that are proxied into the group by group members require the ACTAS permission. In this case the permission is granted to the group, and so all nodes are made aware of these SIDs when they join the group (the AS will distribute the corresponding group permissions).

7.4.2. Multicast

The following applies to multipoint security over transport multicast.

7.4.2.1. Lifecycle

The lifecycle of a multicast group depends on the members of the group. As long as a single member maintains the group, then the group is said to exist. There is no method to "terminate" a group short of destroying the group inside the AS.

7.4.2.2. Timing

The key distribution protocol manages the distribution of timing parameters. The group itself manages most of its own timing parameters. DOF security modes of operation are encouraged not to require synchronized real-time clocks. This means that the security mode of operation must manage whatever timing parameters that it requires.

However, timing that relates to the security of the group state itself is managed through the AS. This will include (at least) the ability to impose a maximum bound on the distribution of updates to the security state. This maximum bound is related to the security policy, and would control (for example) how long it would take to remove a node from the group and ensure that the node can no longer communicate with the group.

When multipoint security is used on multicast there are benefits in being able to easily share state. The security mode of operation is responsible for the distribution and use of this state.

7.4.2.3. Replay Attack Prevention

Prevention of replay attacks over multicast is very difficult. The general problem requires distributing state in such a way that a datagram can be distinguished for another, and that each node remembers enough state about previous datagrams to determine a replayed datagram. This logic requires that each datagram be distributed to the entire group, which is not desired and cannot be guaranteed (datagrams can be blocked by an attacker). In this case the security mode of operation may allow for the unique identification of a single target group member. In this case the ability to replay the datagram to another group member is blocked.

7.4.3. Point-to-Multipoint

The following applies to multipoint security over non-multicast transport sessions. We must be clear that point-to-multipoint security sessions provide no additional security to that of multicast multipoint groups – in both cases the members of the group are likely to be able to impersonate each other. The fact that the server views the communication with each member as a unique session applies at the transport layer, but not at the security layer. Of course the security mode of operation may be able to uniquely authenticate the source and target of a datagram, but that is typically not possible in a multipoint environment.

7.4.3.1. Session Security Scope

An issue with point-to-multipoint groups is that the group is typically in a large number of unique security scopes. However, it is usually not desirable to flood operations to each member, as communication cannot benefit from multicast.

It is not possible to uniquely identify (securely) each member of the group. To satisfy the requirements of scalability in the face of the issue of unique identification, each member of the multipoint group is allowed to specify the session security scope that it desires to use.

In this case the node must not be able to specify a security scope that is larger than that of the group itself. However, the node is able to change the security scope over time. This allows the node to control which operations will be forwarded to it, and provides a mechanism for the point node to limit flooded operations based on security scope.

Note, however, that because there is not unique authentication of each group member, it is possible for a node to "lie" and request a session security scope that does not correspond with its granted permissions.

7.4.3.2. Lifecycle

The lifecycle of a point-to-multipoint group is managed by the server. Since the point-to-multipoint case can be viewed as a set of sessions that share security state, the server has control over each of the sessions and can terminate them at any time. The client also has control over the transport session and can terminate the session if desired.

7.4.3.3. Timing

The key distribution protocol manages the distribution of timing parameters. The group itself manages most of its own timing parameters. DOF security modes of operation are encouraged not to require synchronized real-time clocks. This means that the security mode of operation must manage whatever timing parameters that it requires.

However, timing that relates to the security of the group state itself is managed through the AS. This will include (at least) the ability to impose a maximum bound on the distribution of updates to the security state. This maximum bound is related to the security policy, and would control (for example) how long it would take to remove a node from the group and ensure that the node can no longer communicate with the group.

Multipoint security over point-to-multicast groups has a much more difficult time distributing state. It is more likely to rely on some set of synchronized time, or at least on the relative similarity of time spans between the server and the group members.

7.4.3.4. Replay Attack Prevention

Prevention of replay attacks over multicast is very difficult. The general problem requires distributing state in such a way that a datagram can be distinguished for another, and that each node remembers enough state about previous datagrams to determine a replayed datagram. This logic requires that each datagram be distributed to the entire group, which is not desired and cannot be guaranteed (datagrams can be blocked by an attacker). In this case the security mode of operation may allow for the unique identification of a single target group member. In this case the ability to replay the datagram to another group member is blocked.

7.4.3.5. Group Management

Point-to-multipoint groups are not self-managing the way that multicast groups are. Rather, they are configured. The server node must know that it is the server and the member nodes must know that they are members and must also be configured to know the server node.

7.5. DOF Security Modes of Operation

The DOF Protocol Stack allows for a flexible definition of security modes operation. Each mode is identified by a security mode identifier that is registered with the ODP-TSC.

Each of these identifiers is also an APPID in the range of 0x6000 to 0x6FFF, and they are negotiated through DSP just like other applications, although they use a different attribute type as described in the section on DSP.

There are four ways that the security mode is used:

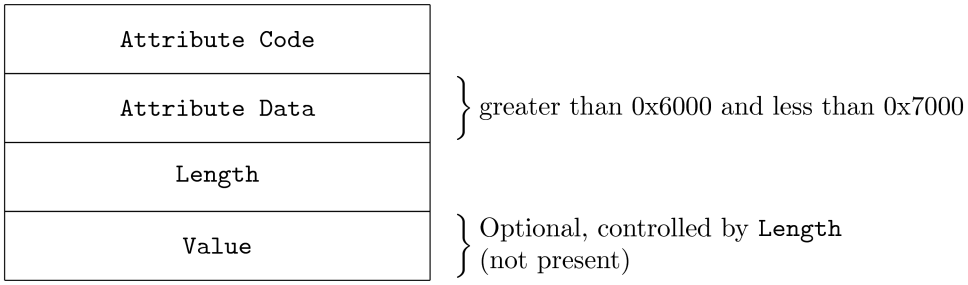
- During DSP negotiation (for 2-node lossless sessions), representing a request to use the negotiated mode and which allows the other node to approve or reject.
- During key distribution, representing a request to use the negotiated mode and which allows the AS to approve or reject. This exchange is also authenticated and prevents man-in-the-middle attacks against the security parameters.
- During key distribution to establish the initial state of the mode. This exchange is also authenticated preventing attacks on the initial state.
- During datagram transmission and reception which is an actual use of the mode.

The first two uses involve the same datagram formats, while the third and fourth use different formats.

When the mode of operation is requested (either of the first two uses above) then the DSP AVP format is used. This format has a mode field, an identifier for the type of mode and variable-length data. The only flexibility that a mode has in this format is the presence and content of the variable-length data. Note that in both of these uses the particular mode of operation that is referenced needs to be identified, and this is done through the DSP Attribute Data. However, since only security modes of operation may be negotiated the following PDU is used.

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

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



`Attribute Code` **One byte.**

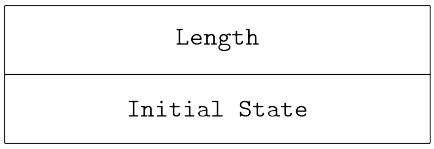
`Attribute Data` **Two bytes, MSB.**

`Length` **One byte.**

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

During the final part of key distribution, the server must initialize any shared state that is required for the mode of operation to begin transmitting and receiving datagrams. This initial state is determined by the mode of operation and always follows this general format:

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.

The reason that this information includes a length is that the datagrams that read and write the data may not have knowledge of the specific security mode formats.

Finally, each mode determines the format that it uses in the DPP Security Header. There is no common formatting for this information, and each mode of operation must specify how it is read and written, along with the meaning of the information.

7.5.1. Common Behaviors

One of the responsibilities of the mode of operation is to validate the datagram properties that are originally set by the transport session or server. Of particular importance is the sense of the source and target addressing type.

For example, a secure session that is point-to-point can safely set the source as node and the target as unicast. However, a multipoint group that cannot authenticate the source must set the source as group. Further, unless the target is uniquely identified the target address for the datagram should be modified to multicast or broadcast rather than unicast (even if the transport datagram was unicast).

Note that security modes of operation are except from the requirement that secure sessions accept only secure datagrams. This allows the security mode of operation to communicate using unsecured datagrams on a secure session.

7.5.2. Security Strength

Many uses of security primitives can use different strengths, particularly encryption algorithms. The strength of the security primitive may come from the following list, although not all security primitives will support all strengths. Security strength may be negotiated by different protocols.

For a current list of registered security strengths, please contact the ODP-TSC or go to https://opendof.org/registry-security-strength.

All DOF security strengths must be registered with the ODP-TSC before use in product. `dof-2008-16-spec-33`

All security strengths must be registered with ODP-TSC before use.

There are a maximum of 255 possible security strengths.

7.5.3. Modes of Operation

For a current list of registered modes of operation, please contact the ODP-TSC or go to https://opendof.org/registry-dps-protocol.

Security modes of operation must be registered with ODP-TSC before use in product. `dof-2008-16-spec-34`

Unrecognized modes must be rejected.

8. Cryptography

Cryptography is the science of protecting information using encryption. This section contains an overview of the cryptography behind DOF security, particularly for key distribution. This is not meant as an exhaustive description of the theories in use, but as an overview of the different primitives and message exchanges.

Please see the references for more detailed information.

8.1. References

The following documents contain information that is relevant to the Cryptography used.

- The Kerberos authentication protocol [http://web.mit.edu/kerberos/www/] is the most common 3 party authentication protocol. Ticket Exchange Protocol (and the Ticket Request Protocol) uses some of the general concepts in Kerberos.
- The KryptoKnight authentication protocol is an improvement of the ideas developed in Kerberos. IBM researched and developed KryptoKnight as a more flexible 3 party authentication protocol with fewer security limitations. The cryptography in the Ticket Exchange Protocol (and the Ticket Request Protocol) is based on KryptoKnight.

  More information can be found in the following paper: Molva et al., 1992, Molva, R., Tsudik, G., Herreweghen, E. V., and Zatti, S. (1992). "KryptoKnight Authentication and Key Distribution System." In Proceedings European Symposium on Research in Computer Security, pages 155-174, .
- The Advanced Encryption Standard (AES) [http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf] was standardized by NIST in 2001. AES is used in Ticket generation in the Ticket Request Protocol.
- The Secure Hash Algorithm was developed by the NSA and standardized [http://www.itl.nist.gov/fipspubs/fip180-1.htm] by NIST in 1995. SHA-256, an extension used by the Ticket Exchange Protocol (and the Ticket Request Protocol), was standardized [http://csrc.nist.gov/CryptoToolkit/tkhash.html] by NIST in 2002 for use with AES.
- The Keyed-Hash Message Authentication Code (HMAC) algorithm [http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf] is a NIST standard for computing Message Authentication Codes. The Ticket Exchange Protocol and the Ticket Request Protocol use HMAC.

8.2. Security Primitives, Upgrade, and Failure Mitigation

NIST SP800-27 contains many principles for sound security design. DOF security adopts these principles to the extent that it can, limited by the fact that DOF security is typically only a piece of a larger product or system.

These principles are complicated by the fact that DOF implementations target a variety of different platforms. Some of these can be easily upgraded, and others may not.

One area where these principles have a large impact is in the ability to implement new technology, and mitigation of security primitive failures (part of resilience).

In order to achieve a system were new technologies can be incorporated and failures can be addressed, each security protocol must list the primitives that it relies on. As a design principle, individual protocols should

not allow for negotiation of security primitives, although they should allow for negotiation of required cryptographic strength (including, for example, MAC sizes, encryption key sizes, etc.).

This design principle means that for any given protocol version, the security primitives that it relies on are known. This has a benefit in that protocol version discovery can be used in a system to discover all security primitives that can be used, and potential problems can be discovered (for example, in the case where a security primitive has a known flaw and should be replaced).

As previously discussed, the Authentication Server is the ultimate authority on security in a system. This includes the "validation" of all security primitives that are used. This is accomplished as the AS allows the use of the security primitives by accepting requests using specific security protocol versions. If the requested protocol uses a security primitive that is not acceptable, then the request is rejected.

There are some limited cases where the AS is not directly involved in the security protocol use. This is typically in group communications, where the group manages its own session key. Note that this is different from two-party key distribution, where the cryptographic blocks used by one protocol are passed through another protocol. In the two-party case the AS still determined the security primitives used. In the group case, the AS had no input to the key distribution process.

However, the AS did approve each node to join the group, and that process used a protocol and security primitives.

In order to allow this beneficial ability of the group to manage its own key, a node should only use security primitives within the group that it used with the AS.



Only use security primitives with other nodes that have been approved by the AS. `dof-2008-16-spec-35`

This approval can be direct (the AS used the primitive itself or directly approved of its use), or transitive (a primitive may be used with another node because it was used with the AS). This logic is restricted to the domain in which the AS approval took place.

8.3. Intellectual Property Statements

A goal of DOF security implementations is to minimize (or eliminate, if possible) use of external intellectual property that requires licensing. However, many DOF protocols allow for extendibility, and it is possible (if desired) to use licensed technology in a particular implementation.

The following statements refer to the technologies that are required (based on the current protocol specifications) related to security and licensing.

8.3.1. AES

AES is standardized as FIPS 197. As part of the submission and final acceptance, the authors made statements about intellectual property rights. They have said that there are none.

See the original submission information at http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html.

8.3.2. CCM

CCM is standardized as NIST SP800-38C. As part of the submission process the authors made statements about intellectual property rights. They have said that there are none.

See the original submission information at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ccm/ccm.pdf.

8.3.3. SHA-256

SHA-256 is standardized in FIPS 180-2. That specification lists that there may be U.S. or foreign patents that cover implementations. In fact, U.S. patent 6,829,355 has been granted. However, the U.S. government has made it available royalty free.

See https://datatracker.ietf.org/ipr/858/ for more information.

There are also references to U.S. patent 5,231,668 (Kravitz). NIST has said publicly that they will not seek royalties for the use of this patent. Further, there are references to U.S. patent 4,995,082 (Schnorr). DOF security does not use SHA-256 in the methods disclosed in these patents.

8.3.4. HMAC

Novell, Inc. holds U.S. patent 5,349,642, which claims keyed hash functions for message authentication. There appears to be prior art (Gene Tsudik, *Infocom*, May 1992), and so this patent is usually ignored.

8.3.5. KryptoKnight

KryptoKnight was patented by IBM. The patent has expired. International patents are still in force, but there appears to be no enforcement.

See http://www.wipo.int/pctdb/en/wo.jsp?wo=1995004416&IA=EP1993001989&DISPLAY=STATUS.

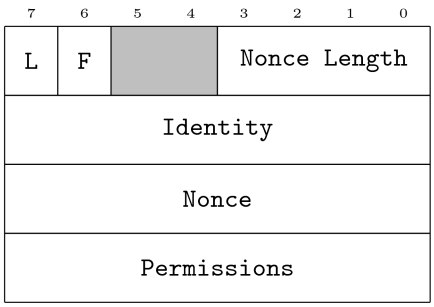
8.4. Common Security Blocks

Several blocks of information are shared among several security protocols. These common blocks are defined here, and references are included in the commands and responses that use them.

8.4.1. Key Request

The Key Request is used by initiators and responders to encapsulate authentication request information. It contains the identity, the nonce, and the permission set used.

⇄ **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.

The maximum Nonce length is 16 bytes, the minimum is 1 byte. **`dof-2008-16-spec-36`**

This is used in the confirmation definition to ensure no nonce aliasing.

The Flags are used to identify whether the request represents a Login (as opposed to a rekey to extend a specific session) and the Nonce Length. The Login flag is informational, but should be set only on new sessions. Its correctness is not validated, but it may be used for identity usage tracking.

The Identity Request contains the required identification information provided by the node to the authentication server. The format is defined later. Note that requests always use resolved credentials, and so the format is always the same.

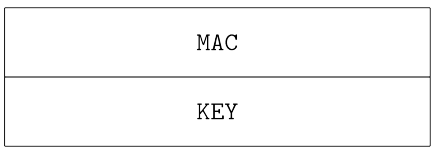
The Nonce is a number that uniquely identifies this request for authentication. It should be generated such that the same number will never be reused before the shared secret (e.g. password, key) between the requesting node and authentication server is changed. This serves to prevent replay attacks that could result in authentication failures. The nonce length can be changed depending on whether a sequential number is used or a random number. If a random number is used then the nonce length should be longer.

The Permission Set is from the point of view of the node that originates the block.

8.4.2. Key Grant

The Key Grant is used by the AS or other responder to securely distribute a session key and to authenticate parameters that are in the Request Block or that are being distributed along with the Ticket.

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



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

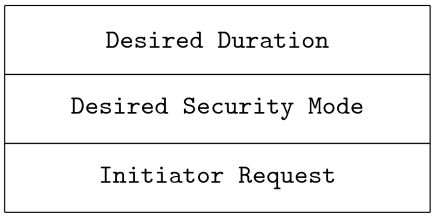
KEY **Thirty-two bytes.** The encrypted MAC field, convoluted with the new session key as described in the *DOF Security Specification*.

The MAC and KEY in the response are tied to the cryptography used with DOF security that is defined later.

8.4.3. Session Blocks



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



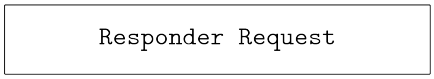
`Desired Duration` **One byte.** The desired duration of the secure session. The field is represented in minutes, with eight minute units. This represents the desired expiration time for the associated permission grants and session key.

`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` **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.

8.5. Notation

The following notation is used for the primitives described below.

- A || B || C: Binary concatenation.
- MAC[k](msg): A message authenticator (secure hash, or signature), where "k" is the security key and "msg" is the message to digest.
- ENC[k](msg): Block encryption of "msg" using key "k". Note that the output bit length will be the same as the input bit length, and that (depending on the cipher used) it may take multiple calls to the cipher to determine the output message. The input message length must be a multiple of the encryption block size.
- A $\oplus$ B: The exclusive-or of A and B.
- RND: A cryptographic random number.
- PAD[n](msg): Zero padding at the beginning of msg, if required, to ensure exactly a length of n.

Nodes must use appropriate random number generation. `dof-2008-16-spec-37`

DOF security relies on the quality of the random number generator used. The most critical of these uses are the session keys, but all uses of random numbers related in any way to security should use

cryptographically secure random numbers. The seeding of the random number generators should also be done appropriately for cryptographic uses.

8.6. Hashing

The hash function MAC[k](msg) deserves special attention. Questions arise as to the exact format of the input fields as the hash requires exact byte inputs in order to achieve matching results between the different nodes applying the hash.

In all cases described here, the format of the data passed to the hash is *exactly the same* as the format passed on the wire. This includes compressed formats, headers (if required), etc. In fact, the definition of what is hashed comes directly from the definitions of the datagrams on the wire.



Special care has been taken to order the hashes such that they can be computed incrementally. This can be a huge benefit to small implementations. Rather than keep the entire set of datagram data, the information known at the time can be added to the hash without finalizing the hash result.

8.7. Three-party Cryptography

The references above are necessary for a full understanding of the cryptography behind DOF security. This section provides the equations that govern the flow of information between nodes and the AS. Note that this is not meant to equate with the commands and responses of the different protocols, but just provide a summary of the information required and how it is combined.



The equations referenced here do not have a direct correlation with commands in the DOF protocols. They are included to provide background information.

DOF security information is tied to an initiator or a responder. The initiator is always associated with the client, the responder with the server. The initiator and responder are both peers with the authentication server.

Each peer provides at least three parameters to the authentication server: its identity, a nonce (non-repeating number) and a permission set. The peer that owns the session, called the initiator, also determines a requested duration for the session, which establishes an expiration time for the negotiated session key and parameter set. The AS responds with an actual duration based on the request and its internal configuration (and possibly its state).

Commands that use this cryptographic method must define the following blocks of information, and must ensure that the blocks meet the specified requirements.

Table 8.1. General Three-Party Parameter Block Names

Parameter Name	Parameter Requirements
D	The security domain identifier of the domain providing the authentication. This is not included directly in any of the other blocks, but is always present in the commands or understood by context.
I	This block must uniquely identify the initiator and contain the initiator nonce and permissions. The command format will identify the specific section. The block is treated as the respective portion of the command just as it appears on the wire.
R	This block must uniquely identify the responder and contain the responder nonce and permissions.

Parameter Name	Parameter Requirements
	The response format will identify the specific section. As with the other blocks, it is used just as it appears on the wire.
A	This block must contain all authenticated information that is distributed from the authenticator.
S	This block contains authenticated initialization information from the responder to the initiator.
N_I	The nonce from block I (above).
N_R	The nonce from block R (above).
ID_R	The responder identity from block R (above). This will be a .
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_S	The 256-bit distributed session key, assigned by the authenticator.
K_D	The 256-bit shared key between the local and remote domain.

Throughout the equations, the use of a subscript 'I' indicates an 'initiator' value, and the use of a subscript 'R' indicates a 'Responder' value.

During the establishment of the key, both peers and the AS end up knowing all of these parameters.

The AS inputs include all parameters that will be authenticated, although some additional information is authenticated from the responder to the initiator dealing with initial security state. How this additional information is authenticated is documented with the specific protocol. The output from the AS contains authenticated parameters from the AS, and a ticket for the requestor and for the provider. Each ticket consists of a MAC and a KEY.

The KEY is defined as:

$KEY_I = ENC[K_I](MAC_I) \oplus K_S$

$KEY_R = ENC[K_R](MAC_R) \oplus K_S$

If the initiator or responder requested (and was granted) authentication into a different domain, then the KEY value becomes:

$KEY_I = ENC[K_D](MAC_I) \oplus K_S$

$KEY_R = ENC[K_D](MAC_I) \oplus K_S$

Note that in this case the receiving node cannot extract the key without sending information to the domain and having it regenerate the ticket using K_I or K_R.

The MAC is defined as:

$MAC_I = MAC[K_I](D || I || R || A || K_S)$

$MAC_R = MAC[K_R](D || I || R || A || K_S)$

Each command that uses this method of key distribution must define the contents of I, R, and A.

The authentication server creates a ticket for each peer consisting of MAC || KEY. This is passed in Security.5: Key Grant.

When a node receives its ticket, it applies the following:

$K_S = KEY_{I/R} \oplus ENC[K_{I/R}](MAC_{I/R})$

It then recreates the MAC based on its input parameters and the shared parameters of the other node and validates that the recreated MAC and the MAC from the ticket match.

The ticket hash must be recreated and validated before the key can be used.

`dof-2008-16-spec-38`

Not validating the hash is a serious security failure.

It is the job of the responder to communicate with the authentication server. Because of this, it obtains its ticket first. If the validation of the last step passes, then the responder sends a confirmation block along with the Ticket for the initiator. This block is used to prove that the responder correctly decoded his Ticket, and is defined as follows:

$TC_R = MAC[K_S](PAD[16](N_I) || PAD[16](N_R) || S || ID_R)$

Note that the protocol may dictate modifications to the confirmation. This is acceptable as long as both sides know that the modification has occurred. If the AS ticket and TC_R is valid as checked by the initiator then it sends a final confirmation block defined as:

$TC_I = MAC[K_S](PAD[16](N_I) || PAD[16](N_R))$

Again, the protocol may dictate modifications to this confirmation. This is acceptable as long as both sides know that the modification has occurred.

8.8. Two-party Cryptography

The discussion so far has dealt with three-party authentication. DOF security can also be used with two-party authentication (for example, peer-to-group authentication). Because the "group" is not able to participate in this communication, the cryptography must change. Two-party authentication is defined between an "initiator" and an "authenticator".

This uses exactly the same cryptography as three-party, but the responder blocks are not present and there are no direct confirmations sent. Protocols that use this method must assume the responsibility of confirmation.

In this method, there is a shared key required to begin the process, and the result is a distributed key and optionally some authenticated, shared data. The initiator protects the transaction from replay attack by using a non-repeating Nonce, just as in three-party authentication.

Commands that use this cryptographic method must define the following blocks of information, and must ensure that the blocks meet the specified requirements.

Table 8.2. General Two-Party Block Names

Parameter Name	Parameter Requirements
D	The security domain identifier of the domain providing the authentication. This is not included directly in any of the other blocks, but is always present in the commands or understood by context.
I	This block must uniquely identify the initiator and contain the nonce.
A	This block must contain all authenticated information that is distributed from the authenticator.
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_D	The 256-bit distributed key, assigned by the authenticator.

Two party authentication parameters must not alias when concatenated with other security blocks. `dof-2008-16-spec-39`

This usually means that the blocks must be self-describing and that they have a defined ending point.

The initiator obtains a ticket from the authenticator. The contents of the ticket are described here.

The Ticket contains two 32-byte blocks. The MAC is defined as:

$MAC_I = MAC[K_I](D \| I \| A \| K_D)$

The second block, or KEY, is defined as follows:

$KEY_I = ENC[K_I](MAC_I) \oplus K_D$

The receiving node applies $ENC[K_I](MAC_I) \oplus KEY_I$ to extract K_D. Once K_D is known, the initiator then recreates MAC_I to verify that the information has been distributed without tampering. K_D represents a shared secret that the node now shares with the authenticator and potentially with other nodes. It can be used as defined in the protocols that use this method.

9. Service Security

In DOF architecture, "service" is a synonym of "provider". Much of what has been presented applies directly to services just as it does to all other providers, objects, and interfaces. What is more critical is how services are provided in DOF implementations.

There are two fundamental approaches to services: the *universal* approach, and the *extended* approach. To compare these two consider a hypothetical service that returns an object's location.

In the *universal* approach, a single binding would be created in the system. The interface would contain a function that takes an object and returns a location. A requestor passes the identity of the object into the function and retrieves the location. This works well until you realize that no single provider could understand the location of *all objects*, and so the requestor may have to iterate over several bindings in order to find a particular object's location. He knows to do this because he finds bindings using the interface described provided by different providers.

Permissions in this case can only be granted on the binding itself. There is no way to further restrict access based on the object (used in the request), because there is no way to enforce restrictions on a particular method parameter value based on a permission set.

In the *extended* approach, a target object is required. The interface still contains a function to obtain the location, but there is no parameter required because the interface assumes the requestor wants the location of the object that the interface is being provided on. This does not preclude a single (or multiple) services from managing the information, it only has to do with how the binding is presented to the system.

The permissions for this approach are granted based on the object in question. This allows finer granularity, and allows for services to be added to the system without the need to modify permissions if a particular requestor has "global" access to a particular object. In this case, any new service merely provides a new interface on that object, and the requestor already has permission to use it.

9.1. Security in the Universal Approach

There are situations where the universal service approach is appropriate. For example, an interface that provides the current universal time. Access would be granted to use the interface, independent of the provider. Permission to provide that interface is used to control which providers are authorized to provide the time.



In general, if an argument to a function includes a reference to an object, then it is not a good fit for a global service.

9.2. Security in the Extended Approach

In the extended approach, access control is handled as discussed in the previous sections. There is no argument that identifies the object, because the object is inherent in the binding of the requested operation. This is one reason that extended services are preferred to universal services.

The usual way to ensure appropriate access control in the extended approach is to utilize an attribute to identify the provider (the service). This allows requestors to be granted appropriate access to *all* providers very easily (using a wildcard on the attribute), and allows the providers to be correctly restricted (by using a wildcard on the object, but only with a specific attribute value).

9.3. Requestor-Specific Access

Note that in the two methods discussed the identity of the requestor is not known. This can be a problem if the permission is dependent on the requestor.

For example, assume that a manager (requestor) has access to bindings used to manage the account settings of another node. This example probably has the following requirements:

- Permission depends on the requestor and on the identity of the managed node.
- The service must create a unique instance for each management session, so that multiple managers do not conflict in their actions.
- The presence of proxy nodes should not cause problems. Note, however, that the presence of proxies should be carefully considered, as they will have access to the unencrypted contents of the management session.

Multiple possible solutions meet these requirements, but a single solution is presented here. Further, the specifics of how to implement this solution are not discussed, just the impact of security for the service.

First, it is clear that there are three objects that are critical to the security: the manager, the provider (service) and the target. All of these must be secure and authenticated. In addition, the binding created should be unique between the service and the manager so that no conflicts will occur.

A good match for this is a binding S.T.U:I. This means that the primary object identifies the service, with a provider attribute that identifies the target and a session attribute that provides uniqueness. The interface I identifies the particular type of management required.

In this solution, the manager is not explicitly identified in the binding, but must have permission to access the binding. A global manager can easily be granted such permission, as can the service that provides the binding, using a wildcard for the provider AVP (used to identify the target).

However, if the manager needs access to a list of target nodes then the permission becomes more difficult to manage. In the case of a manager of 1,000 targets it would need 1,000 permission grants!

In this case it is possible to incorporate the permission request and grant logic directly into the AS logic itself. This would create a type of "request-based permission checking". In this case, the AS may have a sense of a "manage" permission that is not shared with other nodes except as a result of a specific permission request.

For example, a node requesting access to $S_{MGR}.O_1:I_{PW}$ would trigger a check for "requestor has manage permission for O_1" instead of a check for the specific binding requested. This kind of check would be triggered by the presence of S_{MGR} (the management service) or I_{PW} (the management interface).

10. Distributed Security

While this topic may have been more appropriate at the beginning of this document, it covers issues that are difficult to understand without having understood all of the preceding information.

Like everything else in the DOF specifications, DOF security is distributed. Traditional systems usually use a single point-to-point session, for example between a web browser and a web server. Security in these systems is relatively simple, because only a single link (between the client and the server) must be secured. Authentication is also relatively simple in these systems because the client and server communicate directly, and the server is easily able to hold repositories of authentication information (like usernames or passwords).

DOF specifications differs substantially from these typical systems in the following ways:

- There are usually many sessions between the requestor and provider. Intermediate nodes (like OASes) cause these additional sessions, but the goal of the DOF specificications is to make them transparent.
- The peer-to-peer nature of DOF systems makes it difficult to centralize authentication information in a single server (like a web server). The solution is to create a new type of server (the Authentication Server).
- It is possible for many DOF nodes (OASes in particular) to not have full knowledge of the real identities of "distant" requestors and providers. This could be, for example, because they are connected "indirectly" through another OAS. This situation is discussed more later.
- This same "indirect session" problem affects providers in the same way as it does requestors.
- Last, DOF security is complete and all-encompassing. To compare DOF security to something like an SSL session between a web server and a browser is ignoring significant aspects of security (like authentication and access control). In the web server situation, each of these aspects of security is handled by a different application. For example, the web server may have a database of usernames and passwords, and it may protect them on the wire with SSL. The web server may further have access controls for each URL it provides. This works well (and does for many thousands of web servers!), but it does nothing to solve a peer-to-peer topology.

10.1. Security through Distributed Sessions

DOF security is integrally related to a concept of a permission set that is associated with a session key. This session key and associated permission set is used in several ways:

- TCP or other point-to-point connections. In this case the session key is distributed using TEP, and is only known by the two endpoints (and by the AS while it is distributed, although it is not stored for long).
- Unicast UDP sessions. In this case a point-to-point session key can be obtained using TEP, and again is only known by the two endpoints.
- UDP or other multicast sessions. In this case there are two session keys. The first is obtained using TRP and is called the KEK. This same key is shared with all members of the group. The second is the actual session key used for encryption. The KEK is used to join the group and is associated with permissions. The session key is used to communicate with the group.

In these situations, the following point is critical: **there can be no change in the permission set without a change in the session key**. Let us illustrate this with an example. Any two nodes that are known by the Authentication Server can open a session to each other as long as they request no permissions and are in compatible security domains. This *empty* permission set is associated with a session key between the two

nodes. Assume that one of the nodes now wants to access a particular binding – one that it does *not* have permission to access (as stored in the Authentication Server). This request is not allowed by the current shared key (associated with the *empty* permission set).

What is maybe not so clear is that there is no way to change this permission set without changing the session key. Assuming that you could, this would mean that a message encrypted with the same key would at one point in time be rejected, while later it would be accepted. What is it that would indicate that point in time? Any solution (apart from a synchronous change of the session key) would open the system to different types of timing attacks.

This means that the node that wants to access the binding must renegotiate his permission set, and this is done (again using TEP) by requesting a new session key.

Let's now focus on the next major point about distributed security and DOF specifications: **a requestor does not always have a direct session established with the provider, and in fact doesn't know whether he does or doesn't**. It is possible that a client (requestor) has opened a session to an OAS, and that OAS is in fact connected to a device (provider). How does the security work in this situation?

Remember that each session has an associated permission set and shared key. In this case there are two sessions, and therefore two permission sets, each with an associated (and different) shared key. Assume that the requestor *R* has permission to read binding *B*. The provider *P* has permission to provide binding *B*. The OAS *O* has full permissions (he can read, write, execute, and provide everything).

As each request is made, the requestor and provider negotiate a new permission set. Since *O* has all permissions, the end result will depend on the permissions granted to *R* and *P*.

Why is this system secure? Remember that each session is validated by the Authentication Server according the requested permissions and the permissions that the AS has stored for *both* of the nodes that are connecting. When the requestor *R* connects to the OAS *O* and requests to read binding *B*, the AS validates that:

- *R* is known.
- *O* is known.
- *R* has permission to read *B*.
- *O* has permission to provide B (note that this is always validated internally by the AS, even if the permission is not explicitly requested).

Since *O* has full permissions, the checks work. *R* can be assured that whatever he is connected to ultimately has permission to provide *B*. This "trust" is established through the Authentication Server. On the other end, *P* trusts that *O* has permission to read *B*, and *O* knows that *P* can provide *B*. This trust is also established through the Authentication Server.

Think about the situation where there is a large cluster of OASes, O_1 through O_{10}. These OASes are connected in an arbitrary way. Each of these sessions requested *full permissions*, which was granted by the Authentication Server. The trust of *R* to the OAS that it connected to (any one of O_1 through O_{10}), extends through any of the arbitrary sessions until finally it reaches *P*. *R* does not care, and is not aware, of the actual sessions being used. In fact his request may pass over many sessions, and be encrypted with many different shared keys, and evaluated against many permission sets. As far as *R* knows, however, it is connected directly to *P* and trusts the information that it receives.

10.2. The Effects of Key Duration

Each shared key is valid for a given amount of time. The duration is requested during authentication. The AS has ultimate authority over the granted duration. There are two reasons for the duration:

- To ensure that no single key is used for "too long". This is an encryption issue, but because of the strength of our encryption methods this is *almost never a concern*. For example, running at full capacity on a fast machine an OAS would have to run continuously for several decades in order to put the shared key at risk. However slight, this risk is avoided by forcing period rekeys.

- To ensure that no single permission set is used for "too long". This is a much bigger concern than the use of the key for too long. For example, assume that a user requests admin permission, which is granted. This permission is then revoked on the AS. The session has already been established, however, and the permission set includes admin. That permission set will remain valid until the key expires or the session is closed. If the duration is set too long (like 1 week), then the user could use the admin permission for up to a week after the permission was revoked on the AS. This is a security hole, and is solved by setting reasonable maximums on the duration.

Keep in mind that many clients will request permission set changes much more frequently than their requested duration, and that no key extends beyond a single session. These two issues mean that the effects of key duration are felt mostly by sessions that do not frequently request new permissions and leave sessions open for long periods of time. This is a typical case for connected OASes. It is up to the system designers to balance the effects of key duration as part of the AS security policy.

Also keep in mind that the duration of the key has an affect on a session, or communication between nodes. If a key is allowed to expire then a session will close and state will be destroyed.

11. Security Examples

The following sections discuss detailed examples of DOF security. These examples will start with a configuration, and then present a sequence, and finally will show a sequence diagram of the resulting commands and responses.

These examples are extremely technical. While they are based on the specific DOF security requirements, the exact sequences of commands are under control of implementations and may differ slightly from the sequences presented here.

In addition, rather than limit the examples to a specific protocol version these examples use a pseudo-protocol. The pseudo-protocol would be mapped to a specific protocol to determine exact exchanges.

The details of this pseudo-protocol are the topic of the first section. Once the pseudo-protocol is defined then it will be used in the following examples.

11.1. DOF Security Pseudo-Protocol

DOF security is always based either between two nodes or between a node and a group. Further, there is always an initiator and responder whenever security is involved.

The pseudo-protocol defined here deals with sessions between two authenticated nodes or an authenticated node and a group. Each authenticated node is fully identified as

$N(I_N,K_N)$, where

I_N is the identity of the node, and

K_N is the associated node secret.

However, since the I_N and K_N are always required, they are typically not shown in order to make the descriptions clearer (unless the identities or keys are integral to the exchange).

Each two-way session between nodes is indicated as

A $\leftrightarrow$ B as S_n, where

A, B represent nodes, and

S_n indicates the session identifier.

The session identifier S_n can be used in place of the session. The initiator is always shown on the left, the responder on the right. Multipoint sessions are indicated as

(A), (B) $\rightarrow$ G as S_n, where

A, B represent nodes, and

S_n indicates the session identifier.

In this case G represents the group, which is *not* an authenticated node. Either a single node or multiple nodes may be shown on the left, with multiple nodes being a shortcut to showing multiple individual requests for each node. A further difference is that all nodes in the group use the same session identifier.

The letters (shown above as A, B, N, and G) can be replaced by any other letter to indicate different nodes or groups.

The process of establishing a session always results in additional information being known by both nodes. This information is always associated with the session identifier, and is either associated with the session itself (no subscript) or a particular node in the session (subscript of the node name). Session data is flexible and identified as shown later.

Permission sets are important, and are represented as sets of individual permissions. A permission is always identified by the letter 'P' with a numeric subscript, as P_1. The complement of permission P_1 is indicated as P_1^{-1}. Each permission is associated with a security scope, shown as $P_n(\{X, Y\})$. If the context does not require specifying the scope then it may be left off, but it is always assumed to be associated with any permission. As a special case, $P_{n(\varnothing)}$ indicates a permission with an empty security scope.

Permission sets are shown as

$\{P\}$, or $\{P\}_A$, or $\{P_1, P_2\}_A$, or $\{P_1(\{A\}), P_{2(\varnothing)}\}$, where

P indicates the name of a permission set, P_n indicates an individual permission, $\{\ldots\}_X$ indicates an identified permission set,

$\{A\}$ indicates a security scope, and

$P_n(\{A\})$ indicates a permission with associated security scope.

For example, the syntax

$A \leftrightarrow B$ as S_1 with perms:$\{P\}_A$, $\{P\}_B$; key: K

indicates that A and B create a session S_1, that A (the initiator) has permissions $\{P\}_A$ while B (the responder) has permissions $\{P\}_B$, and that the session has a key K. Associated with each permission is a security scope, although not shown in that example.

Existing sessions can be modified, shown as

$A \leftrightarrow B$ continue S_1 adding perms:$\{P\}_A$, $\{P\}_B$.

In this case how the session uses the additional information must be specified (as in "adding", "removing", or "ignoring").

Individual permissions can be compared, as described in the section on permissions. The statement

$P_1 \leq P_2$

indicates that P_1 is allowed by P_2.

Permissions can also be compared against permission sets. The statement

$P_1(\{S_1\}) \in \{P\}_A$

indicates that permission P_1 is allowed by permission set $\{P\}_A$ with a security scope S_1. This means that $\#P_i$: $P_i(\{S_1\})$ is a member of $\{P\}_A$ and $P_1 \leq P_i$. Further, the statement

$P_1(\{S_1\}) \in \{P\}_A,\{P\}_B$ (where S_1 may not be shown)

indicates that $P_1(\{S_A\}) \in \{P\}_A$, $P_1^{-1}(\{S_B\}) \in \{P\}_B$, and $S_A \cap S_B = S_1$. In this case the single permission P_1 is being tested against the combination of $\{P\}_A$ and $\{P\}_B$. Comparing a permission in this way requires testing the permission against the first permission set, and then comparing its *complement* against the second permission set. Also, the resulting security scope is the intersection of the security scopes resulting from the two permission sets.

Communication with the authentication server is shown as a logical assertion, for example

(A, …), (B, …) $\Rightarrow$ A $\leftrightarrow$ B with permissions: $\{P\}_A$, $\{P\}_B$; key: K, where

(*X*, …) indicates a node and any associated data.

A group request to the authentication server is similar:

(A, …), (G) $\Rightarrow$ A $\rightarrow$ G with perms:$\{P\}_A$, $\{P\}_G$; session S_2; kek: K.

There are two classes of failure associated with the authentication server. The first is a form of complete failure, indicated as

(A, …), (B, …) $\Rightarrow \varnothing$.

The second is not necessarily a failure, but rather that some individual permission was requested and not granted. Given the request (A, …), (B, …) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\{P\}_A$, $\{P\}_B$, this kind of failure indicates that

#P_i: $P_{i(\{\varnothing\}) \in} \{P\}_N$, where

#A: B indicates "there exists some A such that B".

Lastly, an operation is indicated as numeric subscripts of the letters 'OP', as OP_1. An operation that requires a permission is listed as OP_1[P]. A response to a given operation is italicized OP_1. Operations that are merely side effects are not shown as responses.

Requests are always indicated as questions, with the target of the question being shown first, followed by a colon.

11.2. General Example Formatting

The following examples show a form of sequence diagram. The left column is reserved for a sequence number, and the right column is reserved for comments. Other columns refer to named entities that take part in the example.

In general, each row consists of a request and a response. If the request requires additional requests before an answer can be specified then the sequence number of the row containing the response is indicated.

11.3. Basic Session

The following cases present basic session scenarios.

11.3.1. Success

In this example, two nodes will establish a session. Both request empty permission sets.

#	A	B	AS	Comment
1	B?: A $\leftrightarrow$ B with perms: $\varnothing$, $\{P\}_B$	(see #3)		A initiates request.
2		AS?: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\varnothing$, $\varnothing$	B: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\varnothing$, $\varnothing$; key: K	B queries the AS and gets a valid response.
3		A: A $\leftrightarrow$ B as S_1 with perms: $\varnothing$, $\varnothing$; key: K		B responds to A, the session is established.

11.3.2. Failure

In this example, two nodes will establish a session. Both request empty permission sets. For some reason the AS rejects the session. In this case the rejection cannot be on the basis of permissions, as none are requested.

#	A	B	AS	Comment
1	B?: A $\leftrightarrow$ B with perms: $\varnothing$, $\{P\}_B$	(see #3)		A initiates request.
2		AS?: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\varnothing$, $\varnothing$	B: (A), (B) $\Rightarrow \varnothing$	B queries the AS and gets a failure. B specifies no permissions in the request.
3		A: A $\leftrightarrow$ B as $\varnothing$		B responds to A, the session is not allowed.

11.4. Group Join

The following cases present basic group scenarios.

11.4.1. Success

In this example, two nodes each join the same group.

#	A	B	AS	Comment
1	AS?: (A), (G) $\Rightarrow$ A $\rightarrow$ G with perms: $\{P\}_A$, $\{P\}_{GA}$		A: (A), (G) $\Rightarrow$ A $\rightarrow$ G as S_1 with $\{P\}_A$, $\{P\}_{GA}$; kek: K	A initiates request to join group G.
2		AS?: (B), (G) $\Rightarrow$ B $\rightarrow$ G with perms: $\{P\}_B$, $\{P\}_{GB}$	B: (B), (G) $\Rightarrow$ B $\rightarrow$ G as S_1 with $\{P\}_B$, $\{P\}_{GB}$; kek K	B initiates request to join group G.
A and B separately join the actual group G.	3	B?: $OP_1[P_1]$	A: OP_1 $[P_2]$	

11.4.2. Failure of One Node

Here, node A cannot join the group while node B can.

#	A	B	AS	Comment
1	AS?: (A), (G) $\Rightarrow$ A $\rightarrow$ G with perms: $\{P\}_A$, $\{P\}_{GA}$		A: (A), (G) $\Rightarrow \varnothing$	A initiates request to join group G and is rejected.
2		AS?: (B), (G) $\Rightarrow$ B $\rightarrow$ G with perms: $\{P\}_B$, $\{P\}_{GB}$	B: (B), (G) $\Rightarrow$ B $\rightarrow$ G with $\{P\}_B$, $\{P\}_{GB}$; kek K	B initiates request to join group G.

11.5. Proxy

The following examples show two proxy situations. In the first permissions between two nodes have been negotiated beforehand, in the second they are extended over time.

11.5.1. Proxy with Pre-Authorized Session

The following assertions are made at the beginning of this sequence:

$A \leftrightarrow B$ as S_1 with perms: $\varnothing, \varnothing$

$B \leftrightarrow C$ as S_2 with perms: P_B, P_C, $\#P_i : P_i \in \{P\}_B, \{P\}_C$.

$C \leftrightarrow D$ as S_3 with perms: $\varnothing, \varnothing$

#	A	B	C	D	AS	Comment
1	B?: $OP_1[P_1]$ (queued)					$P_1 \notin \varnothing$, so A must augment permissions.
2	B?: $A \leftrightarrow B$ with perms: $\{P_1\}, \{P\}_B$	(see #4)				A asks for augmented permissions.
3		AS?: (A), (B) $\Rightarrow A \leftrightarrow B$ with perms: $\{P_1\}, \{P_1^{-1}\}$			B: (A), (B) $\Rightarrow A \leftrightarrow B$ with perms: $\{P_1\}, \{P_1^{-1}\}$, key: K_2	B asks the AS, which approves.
4		A: $A \leftrightarrow B$ continues S_1 adding perms: $\{P_1\}, \{P_1^{-1}\}$; ignoring key				B confirms the new permissions.
5	B?: $OP_1[P_1]$	(see #13)				A requests from B.
6		C?: $OP_1[P_1]$	(see #12)			B forwards to C.
7			D?: $OP_1[P_1]$ (queued)			C queues.
8			D?: $C \leftrightarrow D$ with perms: $\{P_1\}$	(see #10)		C requests augmented permissions.
9				AS?: ©, (D) $\Rightarrow C \leftrightarrow D$ with perms: $\{P_1\}, \{P_1^{-1}\}$	D: ©, (D) $\Rightarrow C \leftrightarrow D$ with perms: $\{P_1\}, \{P_1^{-1}\}$, key: K_2	D goes to the AS, which approves.
10				C: $A \leftrightarrow B$ continue		D responds to C.

#	A	B	C	D	AS	Comment
				S_1 adding perms: $\{P_1\}$, $\{P_1^{-1}\}$; ignoring key		
11			D?: $OP_1[P_1]$	C: $OP_1[P_1^{-1}]$		C forwards the operation and D responds.
12			B: $OP_1[P_1^{-1}]$			C responds.
13		A: $OP_1[P_1^{-1}]$				B responds.

11.5.2. Proxy with Failed Permissions

The following assertions are made at the beginning of this sequence:

A $\leftrightarrow$ B as S_1 with perms: $\varnothing$, $\varnothing$

B $\leftrightarrow$ C as S_2 with perms: P_B, P_C, $\#P_i : P_i \in \{P\}_B, \{P\}_C$.

C $\leftrightarrow$ D as S_3 with perms: $\varnothing$, $\varnothing$

#	A	B	C	D	AS	Comment
1	B?: $OP_1[P_1]$ (queued)					$P_1 \notin \varnothing$, so A must augment permissions.
2	B?: A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P\}_B$	(see #4)				A asks for augmented permissions.
3		AS?: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P_1^{-1}\}$			B: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P_1^{-1}\}$, key: K_2	B asks the AS, which approves.
4		A: A $\leftrightarrow$ B continues S_1 adding $\{P_1\}$,$\{P_1^{-1}\}$, ignoring key				B confirms the new permissions.
5	B?: $OP_1[P_1]$	(see #12)				A requests from B.
6		C?: $OP_1[P_1]$	(see #11)			B forwards to C.
7			D?: $OP_1[P_1]$ (queued)			C queues.
8			D?: C $\leftrightarrow$ D with perms: $\{P_1\}$	(see #10)		C requests augmented permissions.

#	A	B	C	D	AS	Comment
9				AS?: ©, (D) $\Rightarrow$ C $\leftrightarrow$ D with perms: $\{P_1\}$, $\{P_1^{-1}\}$	D: ©, (D) $\Rightarrow$ $\varnothing$	D requests the AS, which fails.
10				C: C $\leftrightarrow$ D continue S_3		D indicates failure to C.
11			B: failure			C indicates failure of operation (lack of permission).
12		A: failure				B indicates failure.

11.6. Changed Permissions

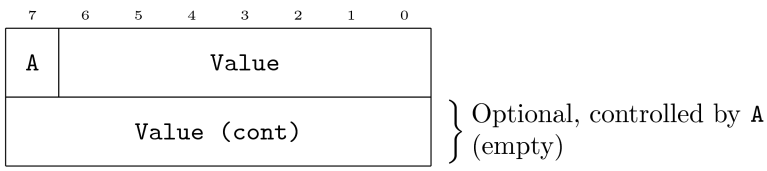
The following example shows nodes A and B that initially have no permissions, and then augment the session based on an operation.

#	A	B	AS	Comment
1	B?: A $\leftrightarrow$ B with perms: $\varnothing$, $\{P\}_B$	(see #3)		A initiates request.
2		AS?: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\varnothing$, $\varnothing$	B: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\varnothing$, $\varnothing$; key: K	B queries the AS and gets a valid response.
3		A: A $\leftrightarrow$ B as S_1 with perms: $\varnothing$, $\varnothing$; key: K		B responds to A, the session is established.
4	B?: $OP_1[P_1]$ (queued)			$P_1 \notin \varnothing$, so A must augment permissions.
5	B?: A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P\}_B$	(see #7)		A asks for augmented permissions.
6		AS?: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P_1^{-1}\}$	B: (A), (B) $\Rightarrow$ A $\leftrightarrow$ B with perms: $\{P_1\}$, $\{P_1^{-1}\}$, key: K_2	B asks the AS, which approves.
7		A: A $\leftrightarrow$ B continue S_1 adding perms: $\{P_1\}$,$\{P_1^{-1}\}$; ignoring key		B confirms the new permissions.
8	B?: $OP_1[P_1]$	A: $OP_1[P_1^{-1}]$		A requests and B responds. Permissions allow it.

12. External PDU Reference

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

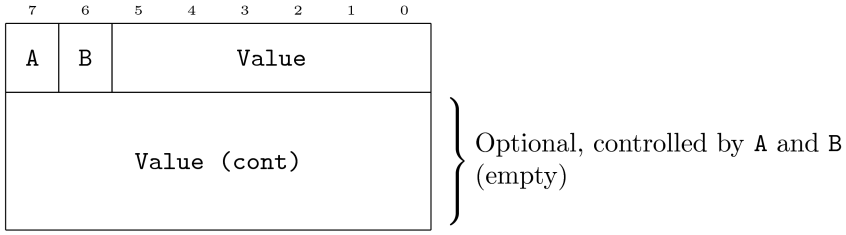
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`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 24-Bit `dof-2009-1-pdu-2`

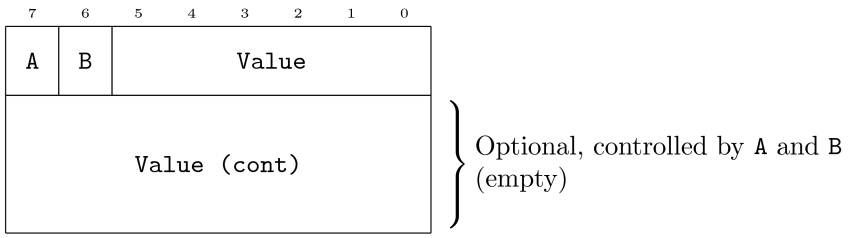


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

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

`Value` **Either seven, fourteen, or twenty-two 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 twenty-two bits (three bytes) are used.

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

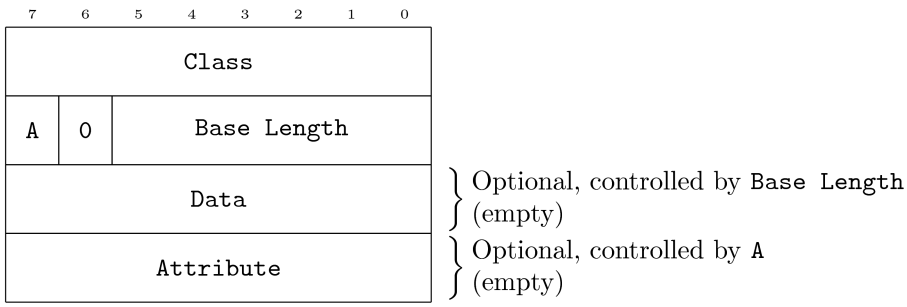


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`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.

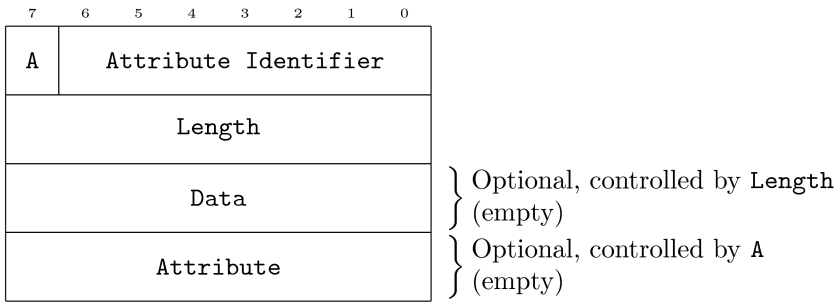
`A` **One bit.** Controls the presence of `Attribute`.

`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.

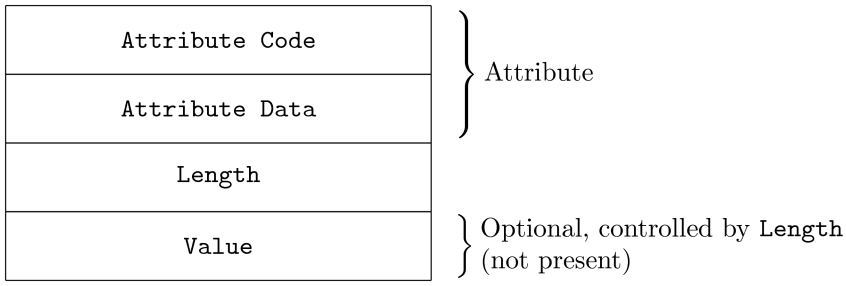
Object Identifier Attribute `dof-2009-1-pdu-5`



`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.

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

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.