DIGITAL OBJECT INTERFACE PROTOCOL SPECIFICATION

——VERSION 2.1

June 20, 2021

Task Group 2: Digital Object Interface Protocol Standard Development & Application

ATSD

CONTENTS

1. Introduction

This document is a specification for the Digital Object Interface Protocol (DOIP), a core protocol of the Digital Object Architecture (DO Architecture; or DOA). The DO Architecture is a logical extension of the Internet architecture that addresses the need to support information management more generally than just conveying information in digital form from one location on the Internet to another. It is a non-proprietary architecture and is publicly available without charge. It is an outgrowth of early work on mobile programs[1], and security for packet radio systems[2]. The DOIP is intended to enable interoperability across heterogeneous information systems.

This specification is an extension of the reference document of Digital Object Interface Protocol [3](DOIP) version 2.0. In DOIP version 2.0, the DOIP protocol is based on TLS. This document proposes an extension to eliminate the dependence of DOIP on TLS, and a new message format to support pluggable communication protocols. To enhance the security of the DOIP message, this specification proposes the encryption/decryption schema based on Identifier/Resolution System. Both DOIP client and DOIP service can use this schema to encrypt/decrypt their message to protect sensitive information.

The following is a list of the major functional differences between this specification and DOIP version 2.0 specification.

1. The underlying communication protocol is pluggable. There is a minimum requirement of DOIP 2.0 that TLS should be supported for secure network communication. To expand the working scope of DOIP, this specification eliminates the dependency of TLS, so that most communication protocols can be supported for network communication. Meanwhile, the Identifier/Resolution System based security enhancement can be used to protect sensitive information when tunneled through common communication protocols other than secure protocol.
2. DOIP can be tunneled through unreliable communication protocols. DOIP 2.0 transmits message based on a reliable communication protocol, and has no mechanism to deal with the loss of messages. This specification proposes a new message format that supports message lost detection and message retransmission. This new message format also supports integrity protection based on digital signature.
3. Support Identifier/Resolution System based security enhancement to protect sensitive information. DOIP 2.0 mainly supports security assurance relies on the secure communication protocol. It is very important to support a native security mechanism. This mechanism is supposed to give security enhancement in two aspects. Firstly, it can be used to support integrity protection based on digital signature. Secondly, it can be used to encrypt the DOIP messages.
4. Specified mechanism to support streaming data. With the great increment production and usage of streaming data, it is important and useful to support this new data type. DOIP 2.0 did not give native support to streaming data. This specification proposes some guides to support streaming data, including defining a new data type and the corresponding

operation.

2. Digital Object Architecture

The DO Architecture introduces the concept of a digital object, which forms the basis for the architecture[4]. A digital object (DO) is a sequence of bits, or a set of sequences of bits, incorporating a work or portion of a work or other information in which a party has rights or interests, or in which there is value, each of the sequences being structured in a way that is interpretable by one or more computational facilities[5]. Each DO, as an essential element, has an associated unique persistent identifier, known as digital object identifier (referred to informally as a handle). Also, each DO has a type, which determines the operations the DO support. DO can be serialized when it needs to be transferred between client and service.

For all practical purposes, the concept of a digital object is substantially similar to the notion of "digital entity" as defined in ITU-T Recommendation X.1255[6] that is based largely on the Digital Object Architecture. The ITU-T Recommendation is available in other languages. An "entity" in that recommendation is defined as anything that can be separately and uniquely identified. It also describes a "digital entity" (DE) as an entity that is represented as, or converted to, a machine-independent data structure consisting of one or more elements that may be parsed by different information systems. In this specification, the terms digital object and digital entity are used interchangeably. A detailed description of DOs, the DO Data Model, DO interface protocol, and federated registries are presented in X.1255.

Furthermore, DOA has also published Y.4459[17] as a recommendation to achieve the interoperability for Internet of Things.

The DO Architecture specifies two core protocols and three basic components. As described briefly below, the three components are the identifier/resolution system, the repository system, and the registry system. In practice, the repository and registry components are modular and may be combined, as needed.

The first protocol, the Identifier/Resolution Protocol (IRP), also known as the Handle System Protocol in an earlier version, is used for creating, updating, deleting, and resolving digital object identifiers. As specified in the IRP, each identifier is associated with an identifier record containing relevant "state information" that clients can resolve to; and all identifiers are of the form prefix/suffix where, by default, the prefix may first be resolved to locate the specific identifier/ resolution service to be used and the suffix may be any bit sequence. An organization may run a resolution system for its own set of identifiers by having a prefix allotted to it, and any existing identifier may be converted to a digital object identifier by treating it as a suffix and prepending its allotted prefix. A system implementation based on an earlier version of the protocol was described in three RFCs[19][20][21]; the document specifying the Identifier/Resolution Protocol will be available shortly.

The second protocol, the Digital Object Interface Protocol (DOIP), is defined for use by digital object services more generally, of which the repository and registry systems are specific instances. Digital object services are intended to implement the DOIP and its basic required features, as specified in this document. Another earlier version of this protocol, based on the Repository Access Protocol (RAP) originally described by R. E. Kahn and R. Wilensky[4], was made publicly available in 2009[7].

There are three basic components in DOA: The Identifier/Resolution System for DO identification and analysis, the Repository System for DO storage and access, and the Registry System for DO metadata registration and DO search.

2.1 The Identifier/Resolution System

The identifier/resolution system is one of the three components comprising the Digital Object Architecture. This system enables several digital object services, including:

1. allotment of unique identifiers to information in digital form structured as digital objects regardless of the location of such information or the technology used to serve such information;
2. rapid resolution of the identifier to current state information about the corresponding digital object, e.g., its location(s), access & usage policies, timestamps, and/or public keys; and
3. administration of the identifier records that contain the state information.

2.2 The Repository System

The repository system is a digital object service that provides the necessary functionality to manage digital objects including the provision of access to such objects based on the use of identifiers, and with integrated security. Through the use of identifiers in the access protocol, the repository system abstracts away the details of the storage technologies from the clients, thus enabling a long-lived mechanism for depositing and accessing digital objects. Access to this system is enabled using the DOIP described below.

2.3 The Registry System

The registry system is a specialized repository system intended to store metadata about digital objects rather than the digital information itself. When used as a standalone component, it typically stores metadata of digital objects managed by one or more repository systems. Access to this system is enabled using the DOIP as well.

3. Digital Object Interface Protocol

The Digital Object Interface Protocol (DOIP) ver. 2.1 specifies a standard way for clients to interact with digital objects (DOs). It is assumed that such DOs are managed by DO Services, which we often refer to as DOIP services in this document, and that the protocol implementation is part of those services. In this context, a DOIP service itself is considered a digital object. By its very nature, a protocol is intended for enabling interaction between one or more other entities running the protocol and thus, in general, to support a specific form of process-to-process interaction in a network environment.

The DOIP makes use of the IRP for associating identifiers with different elements of the protocol. The maximum size of an identifier will vary over time, but, initially, the maximum size of identifiers as specified in the DOIP is 4096 bits.

The DOIP enables the provision of security using PKI[8] to validate digital objects, including for service/client authentication as well as for ensuring integrity via signatures. The inherent PKI support will also help clients and services leverage encryption. Access control of DOs using identifiers to designate an approved access control list and a PKI challenge response test is assumed by the protocol. Basic operations that clients may invoke on the services are defined; and the protocol inherently supports the addition of operations.

DOIP can be tunneled through most of the communication protocols and the DOIP itself can be used to determine the choice of such protocol. In this specification, we define a message format with the built-in integrity check and security features. Messages can be transferred through unreliable communication protocol such as UDP. Client can choose the suitable communication protocol of the target service according to some specific fields in DOIPServiceInfo. See section 5.2.1 for details.

In addition to transport security, several other specifications are also leveraged by the protocol: one is how the serialization is achieved, for which JSON[1] can be used. The rest of this document assumes that JSON is used for serialization unless otherwise stated. The other, as indicated above, is the use of PKI for encryption and decryption of DOs including the authentication of other system resources. But this capability, although relying on techniques external to the protocol, is enabled with the use of digital object identifiers. Other external specifications include Unicode[11] (specifically UTF-8 encoding[12]), TCP[13], MIME[14], X509[15], JWS and JWK[16].

Each DO must specify its type. Core types are defined for this purpose; and types are extensible to allow for the creation of new types. One important function of types is to enable a DOIP service to identify allowable operations. Types are allotted identifiers, and each type is therefore associated with an identifier record that can be accessed by use of the IRP. The semantic and other structuring specifics of type records are not specified in the DOIP. It is assumed that groups or organizations with domain expertise will take responsibility for creating types in their domain and for specifying the semantic and serialization of type records, for instance: Streaming data.

Streaming data is data that is continuously generated by different sources. Because streaming data is continuously generated, it brings some problems when processing this data. For instance, it cannot be retrieved through a single retrieve operation. To support streaming data, additional information may be included in the request. For instance, when a client tries to retrieve a Streaming DO, it may include additional information in the request (e.g. add attributes). When the DOIP service received the request, it would check the additional information and decide how to deal with the request (e.g. start to push data or stop pushing data).

Because streaming data is continuously generated and usually incurs lots of overhead, it is recommended that users consider performance problems when processing streaming data (e.g. using differential coding to reduce video processing overhead).

4. Identifiers

The DOIP defines four forms of identifiers: one for basic operations, one for types, one for status information and one for DOs. It is not necessarily that every identifier must be resolvable by IRP if there is a consensus on the meaning of the identifier, such as basic operation identifiers and status information.

Implementations can define their own set of identifiers, as appropriate, as long as they are resolvable as specified in the IRP. That said, whenever DOIP is used in specific environments where external resolutions of identifiers become unnecessary, either because the anticipated clients are already aware of the information in the identifier records, or the exposure of such information in the identifier records poses security risks or other concerns, such identifiers may not be resolvable via the IRP. However, it is notable that when DOIP Services are made available in the Internet, it is anticipated that such identifiers shall resolve to records that contain minimum information as specified in this document.

The convention used for the basic set is to use the prefix 0.DOIP for operations as well as status information, and the prefix 0.TYPE for types. In this version of the protocol, the identifiers have semantics; in subsequent versions, it is intended that such identifiers will have non-semantic representations as well.

5. Types

Types, in this context, are intended for DOIP services and related clients to learn of operations that are appropriate to be invoked against a DO. In particular, each DO specifies its type, and that type shall inform DOIP services what operations to perform. Clients shall learn of those applicable operations from DOIP services. Some of these types are intended to be available to all DOIP services, while others may not. In particular, types that are part of encrypted digital information would not be

available for general use by DOIP services in the clear.

Types are associated with unique identifiers. The identifier record associated with any type, at a minimum, will specify its parent type. In particular, a type indicates in its identifier record, using one value whose data field is associated with 0.TYPE/Type shall be the type of its parent.

5.1 Core types

The necessary core types for DOIP operations are defined here. 0.TYPE/Type is the root of the types. All other core types extend from 0.TYPE/Type. Types that DOIP implementations may create shall extend either from 0.TYPE/Type or its extensions as defined here.

Core types and their intrinsic relationships (presented as a hierarchy) are defined below:

1. 0.TYPE/Type: the root of the types.
    1) 0.TYPE/DO: This is a generalized DO type. DOs that correspond to this type shall include, in their identifier record, one value whose data field associated with "0.TYPE/DOIPServiceInfo" shall be the service identifier of the DOIP service that manages the DO. DOIP services may use types extended from this type to convey DO specializations.
        – 0.TYPE/DOIPServiceInfo: the type that shall be used to convey the DO service information.
        – 0.TYPE/DOIPOperation: the type that shall be used to designate that a DO represents an extended operation.

5.2 Extended types

5.2.1 0.TYPE/DOIPServiceInfo

The DOIP service information of every repository and registry must be managed as a special DO type: 0.TYPE/DOIPServiceInfo.

The identifier of a DOIP service information must be resolvable through IRP and its identifier record must contain the following value at least:

1. listeners (required): DOIP service can have more than one listener, each of which has different URLs and accept different message formats of different protocol versions. Each listener consists of:
    1) url (required): the URL of the service in the regular format: "protocol://ip:port".
    2) protocolVersion (required): the highest version of the DOIP protocol supported.
2. publicKey (required): the public key expressed in JWK format by default.
3. serviceName (optional): the name of the DOIP service.
4. serviceDescription (optional): the description of the service.

In some cases, DOIP service information need to be serialized as a DO and transferred, such as basic operation “0.DOIP/Op.Hello”. DOIP service information is serialized based on the basic DO serialization. The content of the DO should include the following information:

1. id: the identifier of the DOIP service.
2. type: must be 0.TYPE/DOIPServiceInfo.
3. attributes:
    1) listeners (required): DOIP service can have more than one listener, each of which has different URLs and accept different message formats of different protocol versions. Each listener consists of:
        – url (required): the URL of the service in the regular format: “protocol://ip:port”.
        – protocolVersion (required): the highest version of the DOIP protocol supported.
    2) publicKey (required): the public key expressed in JWK format by default.
    3) serviceName (optional): the name of the DOIP service.
    4) serviceDescription (optional): the description of the service.
    5) any number of other fields (optional).

5.2.2 0.TYPE/DOIPOperation

When the extended operation is used, the operation details shall be managed as a DO. Not all identifiers of extended operations must be registered and managed as a DO if it is for internal use. However, we suggest doing so for better interoperability.

The identifier of an extended operation is suggested to be human-readable and its identifier record should contain these values at least:

1. repository: the identifier of the repository where this DO is managed.
2. operationName (optional): the name of the DOIP service.
3. operationDescription (optional): the description of the service.

A client may get the detailed information of an extended operation by sending a “0.DOIP/Op.Retrieve” request, and the response body should be the extended operation as a serialized DO. The content of the DO should include the following information:

1. id: the identifier of the operation.
2. type: must be 0.TYPE/DOIPOperation.
3. attributes:
    1) DOIP/Request: One or more key-value pairs in JSON format are used for describing the expected values like basic operations in section 6.1. One key must be ‘human-readable’ to suggest that the description of the DOIP request is useful for humans. Other forms of descriptions that simply automation may additionally be used.
    2) DOIP/Response: One or more key-value pairs are used for describing the expected values like basic operations in section 6.1. One key must be ‘human-readable’ to suggest that the description of the DOIP request is useful for humans.
    3) DOIP/OperationReference (optional): An optional field to reference another operation identifier to establish similarity of operation implementations.

6. Operations

The DOIP operations are categorized as Basic and Extended. Basic operations must be properly interpreted by every DOIP service and shall be built into those services a priori. Extended operations may be implemented by DOIP services as they choose, provided that adequate security is enforced in retrieving, validating, and executing such operations.

All DOIP operations, whether basic or extended, must have unique resolvable identifiers as specified in the IRP. The user could select the granularity of the encapsulation according to the application requirements, which makes the DOIP operation mode more reasonable.

6.1 Basic Operations

6.1.1 0.DOIP/Op.Hello

An operation to allow a client to get information about the DOIP service.

1. Request
    1) Message header parameters:
        – identifier: identifier of target DOIP service.
        – operation: “0.DOIP/Op.Hello”.
        – response: none.
        – attributes: none.
    2) Message body: none.
2. Response
    1) Message header parameters:
        – identifier: identifier of target DOIP service.
        – operation: “0.DOIP/Op.Hello”.
        – response: status code, see section 6.3 for detail.
        – attributes: none.
    2) Message body: the default serialization of the DOIP Service Information as a DO.

6.1.2 0.DOIP/Op.Retrieve

An operation to retrieve (some parts of the) information represented by the target DO.

1. Request
    1) Message header parameters:
        – identifier: identifier of target DO.
        – operation: “0.DOIP/Op.Retrieve”.
        – response: none.
        – attributes:
            ■ element: if specified, retrieves the data of that element.

        - includeElementData: if present and the element is absent, returns the serialization of the DO including all element data.
    2) Message body: none.
2. Response
    1) Message header parameters:
        - identifier: identifier of target DO.
        - operation: “0.DOIP/Op.Retrieve”.
        - response: status code, see section 6.3 for detail.
        - attributes: same as the corresponding request.
    2) Message Body: the default output is the serialization of the object using the default serialization without element data. If "element" was specified, the output is a single byte segment with the bytes of the specified element. If "includeElementData" was specified and the element is absent, the output is the full serialized DO.

6.1.3 0.DOIP/Op.Create

An operation to create a digital object within the DOIP service. The target of a creation operation is the DOIP service itself.

1. Request
    1) Message header parameters:
        - identifier: identifier of target DOIP service.
        - operation: “0.DOIP/Op.Create”.
        - response: none.
        - attributes: none.
    2) Message Body: a serialized DO including the identifier of DO. Client can register DO in Identifier/Resolution System and get DO identifier first, then send a Create message to target repository. The "id" can also be omitted to ask the DOIP service to automatically choose the identifier and register in Identifier/Resolution System.
2. Response
    1) Message header parameters:
        - identifier: identifier of target DOIP service.
        - operation: “0.DOIP/Op.Create”.
        - response: status code, see section 6.3 for detail.
        - attributes: none.
    2) Message Body: the default serialization of the object without element data.

6.1.4 0.DOIP/Op.Delete

An operation to remove the target DO from the hosted DOIP service.

1. Request
    1) Message header parameters:
        - identifier: identifier of target DO.
        - operation: “0.DOIP/Op.Delete”.

- response: none.
- attributes: none.

2) Message Body: none.

2. Response
   1) Message header parameters:
      - identifier: identifier of target DO.
      - operation: “0.DOIP/Op. Delete”.
      - response: status code, see section 6.3 for detail.
      - attributes: none.
   2) Message Body (optional): arbitrary response description string.

6.1.5 0.DOIP/Op.Update

An operation to update (some parts of the) information represented by the target DO.

1. Request
   1) Message header parameters:
      - identifier: identifier of target DO.
      - operation: “0.DOIP/Op.Update”.
      - response: none.
      - attributes: none.
   2) Message Body: a serialized digital object. The default serialization may be used if the object lacks element data (or if no element data is to be changed). Elements that are not intended to be changed can be omitted from the input.
2. Response
   1) Message header parameters:
      - identifier: identifier of target DO.
      - operation: “0.DOIP/Op.Update”.
      - response: status code, see section 6.3 for detail.
      - attributes: none.
   2) Message Body: the default serialization of the digital object without element data.

6.1.6 0.DOIP/Op.Search

An operation to discover digital objects by searching metadata contained in the set of digital objects managed by the DOIP service.

1. Request
   1) Message header parameters:
      - identifier: identifier of target DOIP service.
      - operation: “0.DOIP/Op.Search”.
      - response: none.
      - attributes:
        - "query": the search query to be performed, in a textual representation.
        - "pageNum": the page number to be returned, starting with 0.

- "pageSize": the page size to be returned; if missing or negative, all results will be returned; if zero, no results are returned, but the "size" is still returned.
- "type": either "id", to return just object ids, or "full", to return full object data (omitting element data); defaults to "full".

2) Message Body: none.

2. Response
   1) Message header parameters:
      – identifier: identifier of target DOIP service.
      – operation: “0.DOIP/Op.Search”.
      – response: status code, see section 6.3 for detail.
      – attributes: same with the corresponding request.
   2) Message Body: an object based on the JSON serialization with top-level properties:
      – "size": the number of results across all pages.
      – "results": a list of DOs, each of which is a string (the object id) or the default serialization of a DO omitting element data.

6.1.7 0.DOIP/Op.ListOperations

An operation to request the list of operations that can be invoked on the target DO.

1. Request
   1) Message header parameters:
      – identifier: identifier of target DO.
      – operation: “0.DOIP/Op.ListOperations”.
      – response: none.
      – attributes: none.
   2) Message Body: None
2. Response
   1) Message header parameters:
      – identifier: identifier of target DO.
      – operation: “0.DOIP/Op.ListOperations”.
      – response: status code, see section 6.3 for detail.
      – attributes: none.
   2) Message Body: a serialized list of strings based on the default serialization, each of which is an operation id that the target DO supports.

6.2 Extended Operations

The DOIP services may support operations beyond the basic ones, and identifiers of such operations shall be resolvable as specified in the IRP and can be retrieved as a DO through DOIP. By retrieving the extended operation DO, client can get the input/output and other attributes of such operation. See section 5.2.2 for details.

Those operations are not part of the basic DOIP operations, but the way they are carried out is no different from those of the basic operations. The attributes and input/output of those operations should be clearly defined in the corresponding digital object.

Extended operation specific functionality may be built into the service implementation, if desired. Alternatively, a DOIP service may provide runtime environments that retrieve, validate, and execute code managed in special DOs that pertain to extended operations. In either case, the operation details shall be managed as a DO. The purpose of representing the operation as a DO is to disseminate information about how to invoke the extended operation.

Extended operations can be developed to add specific ways to access digital information or to leverage different security mechanisms such as encryption, role-based access control, or proof of work techniques.

6.3 Status Code

Status codes shall have associated unique identifiers resolvable as specified in the IRP. The following basic status codes are applicable. Additional status codes may be used by implementations and be supplied within attributes, but a basic code must be supplied in the status property of any DOIP response.

1. 0.DOIP/Status.001: The operation was successfully processed.
2. 0.DOIP/Status.101: The request was invalid in some way.
3. 0.DOIP/Status.102: The client did not successfully authenticate.
4. 0.DOIP/Status.103: The client successfully authenticated, but is unauthorized to invoke the operation.
5. 0.DOIP/Status.104: The digital object is not known to the service to exist.
6. 0.DOIP/Status.105: The client tried to create a new digital object with an identifier already in use by an existing digital object.
7. 0.DOIP/Status.200: The service declined to execute the extended operation.
8. 0.DOIP/Status.201: Invalid communication protocol.
9. 0.DOIP/Status.500: Error other than the ones stated above occurred.

6.4 Other Identifiers

The following identifiers designate parameters that are useful and/or necessary for DOIP Operations.

1. 0.DOIP/Request: This identifier shall be used to describe the specifics of the DOIP request for extended operation.
2. 0.DOIP/Response: This identifier shall be used to describe the specifics of the DOIP response for extended operation.
3. 0.DOIP/OperationReference: This identifier shall be used to designate one DOIP operation being similar to another DOIP operation.
4. 0.DOIP/Transport: This identifier may be used to specify the DOIP transport protocol

used by the DO Service; it resolves to an extended DOIP type. If no transport is specified, then TCP/IP is assumed. If the DOIP uses TLS for instance, it may also be specified in this field.

5. 0.DOIP/Encoding: This identifier may be used to provide information that is used by the DOIP Service to specify the encoding used by the DOIP; it resolves to an extended DOIP type.
6. 0.DOIP/AccessControl: This identifier may be used to provide information that specifies the access control operation.

7. Communication

Clients interact with any DOs by establishing DOIP connections to each DO's respective DO Service. To do so, clients will need to acquire that DO Service's information to establish a network connection. This information is called the DO Service Information. The specific values encoded in the Service Information are described in the types section of this document as the 0.TYPE/DOIPServiceInfo type.

The client shall resolve the DO identifier using the IRP. The responding information shall contain either the Service Information associated with the 0.TYPE/DOIPServiceInfo type or receive a redirection to another identifier. In the case the client receives a redirection, it will resolve the new identifier using the IRP and any additional redirection into a new Service Information record. Clients may cache any Service Information for expediting future interactions with the DOIP service. Details of the identifier records are stated in the discussion on Core Types in section 5.1 of this specification.

A DOIP service can also act as a proxy and allow operations to be invoked on DOs managed by other DOIP services. The proxy service invokes the client-specified operation on the service that manages the identified DO and responds the results back to the client. The proxy service may also cache the response for expeditiously responding to such future requests.

7.1 Conventions

The DOIP protocols should follow the following conventions to ensure interoperability among different implementations.

7.1.1 Data Transmission Order

The order of transmission of data packets follows the network byte order (also called the Big-Endian[22]). That is, when a data-gram consists of a group of octets, the order of transmission of those octets follows their natural order from left to right and from top to bottom, as they are read in

English.

7.1.2 Standard String Type: UTF8-String

DOIP messages are transmitted as UTF8-Strings under the DOIP protocol. Throughout this document, UTF8-String stands for the data type that consists of a 4-byte unsigned integer followed by a character string in UTF-8 encoding. The leading integer specifies the number of octets of the character string.

7.2 Serialization of Digital Object

A digital object (DO) as communicated between digital object services and clients must conform to the agreed form of serialization. Client software that incorporates DOIP software may be part of another DOIP service. The minimum required serialization of a DO is specified below. At a high level, a DO consists of an identifier, a type, optional and open-ended attributes, plus optional elements. The identifier of the DO must be unique and resolvable as specified in the IRP.

The serialized DO in DOIP message consists of three segments: JSON segment length, JSON segment, and element data. JSON segment length specifies the length of JSON segment, and the JSON segment follows in JSON format. The last segment is a concatenation of the element data, each of which must be introduced in elements field in JSON segment. The data segment contains the data of all elements and is arranged and assembled in the same order with their appearances in elements array in JSON segment.

7.2.1 <JSON Segment Length>

A 4-byte unsigned integer indicates the length of JSON Segment.

7.2.2 <JSON Segment>

One or more fields (key-value pairs) serialized as a JSON object, includes:

1. id: the identifier of the DO.
2. type: the DO type. Must be 0.TYPE/DO or its extension. See Types section.
3. attributes (optional): one or more fields serialized as a JSON object.
4. elements (optional): one or more elements serialized as an array in JSON, with each element consisting of:
    1) id: identifier of the element; must be unique within a DO.
    2) length: length of the data portion.
    3) type: shall be a type as defined in this spec or a MIME type.
    4) attributes (optional): one or more fields serialized as a JSON object.

7.2.3 <Elements data>

This part contains the data of all elements which should be in the same order with the elements array in the JSON segment. The receiver can deserialize elements data according to the length filed in each element.

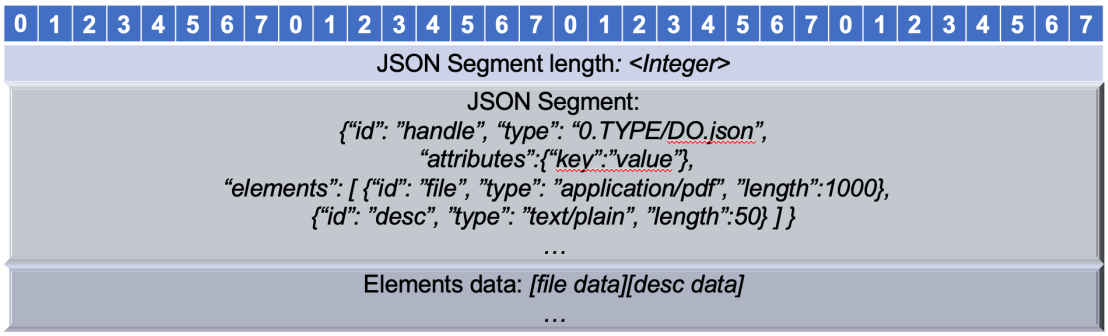


Figure 7.1 Example of serialization of Digital Object in DOIP message

7.3 Message Format

To support the unreliable/connectionless transport protocol, this specification defines a new message format. The message of DOIP consists of 4 parts: Message Envelope, Message Header, Message Body, and Message Credential (optional). Figure 7.2 shows the structure of DOIP message.

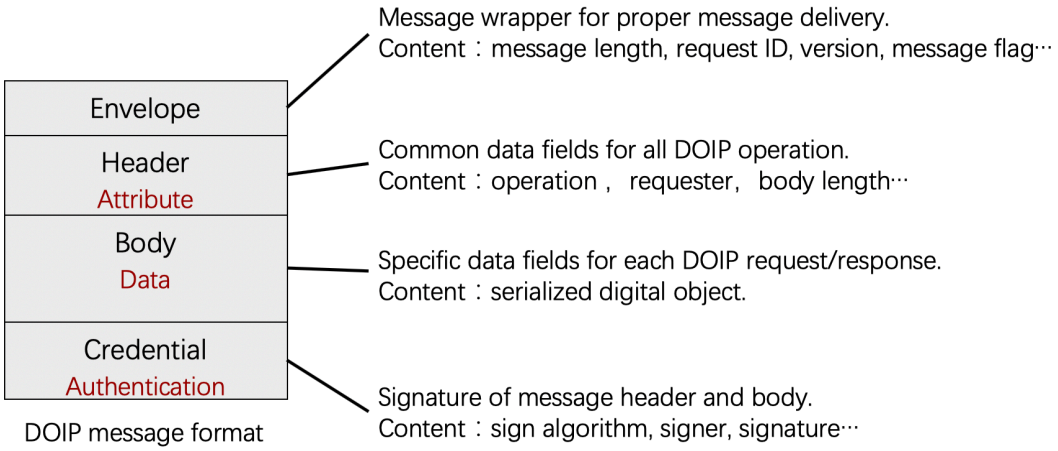
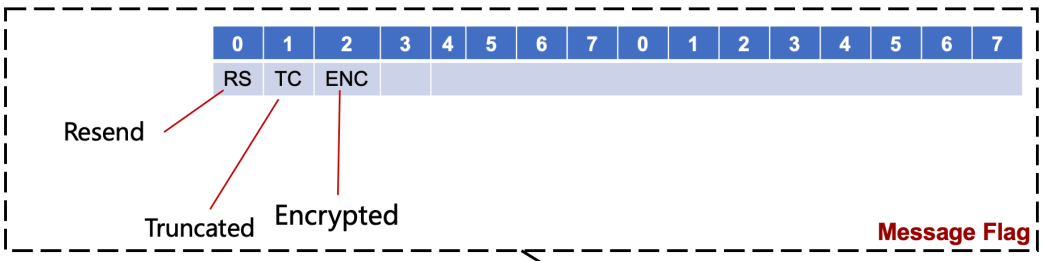


Figure 7.2 Format of DOIP message

7.3.1 Message Envelope

Message Envelope is the message wrapper for proper message delivery with a fixed length of 24 Bytes. Message Envelope is not protected by the digital signature in the Message Credential.



	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
0	Major Version	Minor Version	Message Flag	
1	Reserved			
2	Request Id			
3	Sequence Number			
4	Total Number			
5	Content Length			

Figure 7.3 Format of Message Envelope

7.3.1.1 <Major Version> and <Minor Version>

The <Major Version> and <Minor Version> are used to identify the version of the DOIP. Each of them is defined as a one-byte unsigned integer. This specification defines the protocol version whose <Major Version> is 2 and <Minor Version> is 1.

<Major Version> and <Minor Version> are designed to allow future backward compatibility. A difference in <Major Version> indicates major variation in the protocol format and the party with the lower <Major Version> will have to upgrade its software to ensure precise communication. An increment in <Minor Version> is made when additional capabilities are added to the protocol without any major change to the message format.

7.3.1.2 <Message Flag>

The <Message Flag> is used to define some properties of this envelope.

1. Bit 0 is the RS (ReSend) flag that indicates whether the message is a resend request from the receiver. If the RS bit is set (to 1), means that some packet was lost during transmission and the receiver asks the sender to resend the lost packet. Lost packets can be located by <Request Id> and <Sequence Number> fields. If RS is set to 1 while sequence number is set to -1 indicates the receiver has received all the packets.
2. Bit 1 is the TC (TrunCated) flag that indicates whether this is a truncated message. Message truncation happens most often when transmitting a large message over the UDP protocol.
3. Bit 2 is the ENCryption flag. A request with the ENC bit set (to 1) requires the server to encrypt its response using the public key of the target client. A response with the ENC bit set (to 1) indicates that the message is encrypted. See section 8 for details.

The rest bits are reserved for future use.

7.3.1.3 <Reserved field>

A 4-bytes field for future use.

7.3.1.4 <Request Id>

Each request from a client is identified by a <Request Id>, a 4-byte unsigned integer set by the client. Each <Request Id> must be unique from all other requests from the same client. The <Request Id> allows the client to keep track of its requests, and any response from the server must include the same <Request Id> with the corresponding request.

7.3.1.5 <Sequence Number>

Messages under the DOIP may be truncated during their transmission (e.g., under UDP). The <Sequence Number> is a 4-byte unsigned integer used as a counter to keep track of each truncated portion of the original message. The message recipient can reassemble the original message based on the <Sequence Number>. The <Sequence Number> must start with 0 for each message. Each truncated message must set its TC flag in the Message Envelope. Messages that are not truncated must set their <Sequence Number> to 0.

7.3.1.6 <Total Number>

A 4-byte unsigned integer that specifies the total number of the envelopes of the truncated messages. The length of any single message exchanged under the DOIP is limited by the range of a 4-byte unsigned integer. Longer data can be transmitted as multiple messages with a common <Request Id>.

7.3.1.7 <Content Length>

A 4-byte unsigned integer that specifies the length of the content in this envelope.

7.3.2 Message Header

Figure 7.4 shows the format of the Message Header. Message Header specifies the common fields of every DOIP message such as: operation, target DO ID, etc.

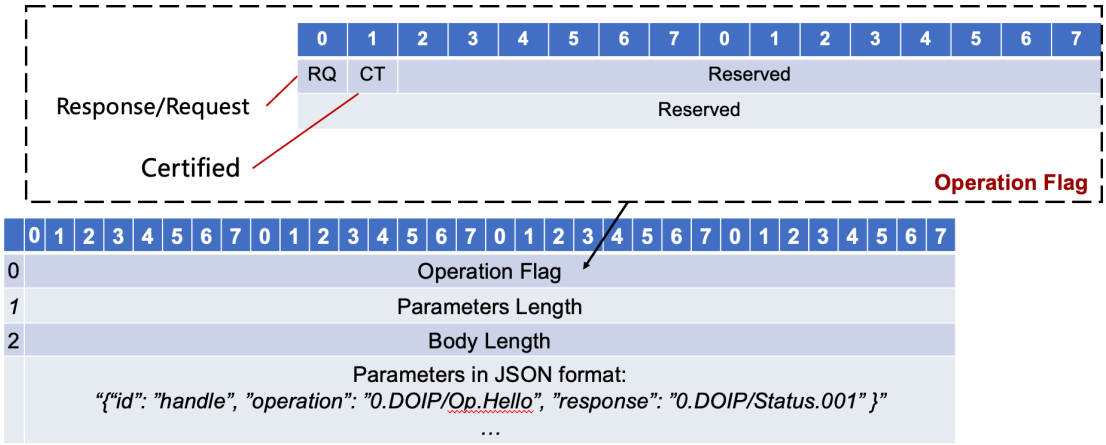


Figure 7.4 Format of Message Header

The Message Header contains following fields:

7.3.2.1 <Operation Flag>

<Operation Flag> is used to define some properties of the DOIP message. The first 2 bits are RQ, CT:

1. RQ: Request/Response bit. A request with the AT bit set (to 1) indicates that this is a request message. Otherwise, it is a response message to a request.
2. CT: CerTified bit. A request with the CT bit set (to 1) asks the server to sign its response with its private key. A response with the CT bit set (to 1) indicates that the message is signed. The server must sign its response if the request has its CT bit set (to 1). If the server fails to provide a valid signature in its response, the client should discard the response and treat the request as failed. If CT is set to 0, this message may not contain the Message Credential part.

Rest bits are reserved for future use.

7.3.2.2 <Parameters Length>

Length of The Attributes JSON (in bytes), 4-byte unsigned integer, indicates the length of the attribute which is the last part of the Message Header..

7.3.2.3 <Body Length>

Length of body, 4-byte unsigned integer, indicates the length of message body which follows message header.

7.3.2.4 <Parameters>

The parameters of the DOIP request/response. Must contain the following elements:

1. id: identifier of target DO to operate.
2. operation: operation code is a handle that identifies the Operation to target DO such as "0.DOIP/Op.Retireve". For a detailed definition of Operation Code, see section 6 for details.
3. response(optional): operation code is a handle that identifies the Response status from a repository such as "0.DOIP/Status.001". For a detailed definition of the Response Code, see section 6.3. Required if it is a response message.
4. attributes(optional): additional attributes in JSON format. Varies according to different operations, see section 6 for details.

7.3.3 Message Body

The Message Body always follows the Message Header and refers to the input/output of the request/response. The Message Body may be empty. The exact format of the Message Body depends on the <Operation Code> and <Response Code> in the Message Header. For details of the Message Body, see section 6.

7.3.4 Message Credential(optional)

Message Credential contains signature information of header and body. Including: length, signature algorithm, signer handle, signature data, etc. The requester can verify the DOIP message through the information in Message Credential. It can be omitted if there is no credential, for example, when the CT in Message Header is set to 0.

Message Credential contains the signature information of DOIP Message when it is signed:

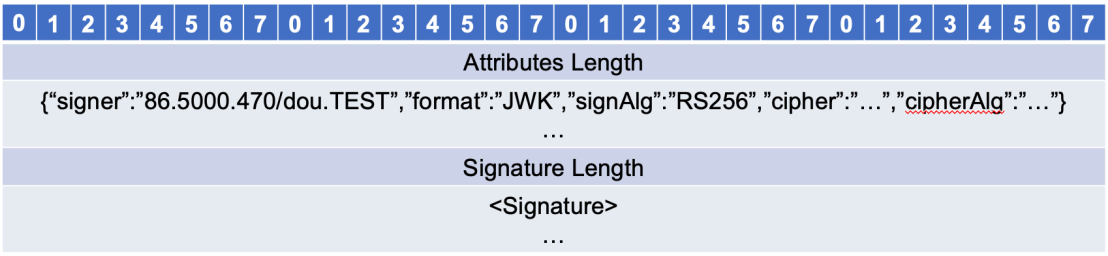


Figure 7.5 Message Credential

The Message Credential consists of following parts:

7.3.4.1 <Attributes Length>

A 4-bytes unsigned integer indicates the length of credential attributes.

7.3.4.2 <Attributes>

Necessary information to describe the signature, serialized in JSON format, including:

1. signer: identifier of the signer of the signature. The receiver can get the public key by resolving the identifier.
2. format(optional): the serialization format of signature, default is JWK.
3. signAlg(optional): the algorithm of the signature, also can be encapsulated in the signature part like JWK.
4. cipher(optional): symmetric key encrypted by recipient's public key, encoded by base64
5. cipherAlg(optional): symmetric encrypt algorithm used to encrypt message body (e.g. AES256)
6. recipient(optional): identifier of the recipient, the receiver can get recipient's public key by resolving this identifier
7. other fields help to verify the signature(optional).

7.3.4.3 <Signature Length>

A 4-bytes integer indicates the length of the signature.

7.3.4.4 <Signature>

Signature in bytes. Variable-length.

8. Security Consideration

DOIP can be tunneled through most of the communication protocols including secure protocols and others. When DOIP is tunneled through a secure protocol, the DOIP message do not need to be encrypted. Otherwise, the DOIP message can be encrypted according to this specification to protect sensitive information. To generate an encrypted DOIP message, follow these steps:

1. get recipient's identifier
2. resolve recipient's identifier through Identifier/Resolution System (e.g. Handle System or another implementation) to get recipient's public key (e.g. RSA2048)
3. generate random symmetric key locally (e.g. AES256)
4. encrypt message body using generated random symmetric key in step 3

5. encrypt generated random symmetric key using recipient's public key
6. put encrypted message body into message body
7. put encrypted symmetric key into message credential
8. set encrypt flag (EC) to 1 in envelope
9. sign message header and body using sender's private key
10. put signature into message credential

After receiving one encrypted DOIP message (EC flag is set to 1), follow these steps to process the message:

1. get sender's identifier from message credential
2. resolve sender's identifier to get sender's public key
3. verify the signature using sender's public key
4. decrypt symmetric key using receiver's private key
5. decrypt message body using decrypted symmetric key
6. process decrypted message body

For a client, it can get the identifier of DOIP service through a web page or other way. For a DOIP service, it can get a client's identifier from the request message.

9. Implementation Guidelines

To be compliant with DOIP 2.0, a complete implementation should support both this new specification and DOIP 2.0. To implement DOIP 2.0, check https://www.cordra.org/ for reference. For a complete implementation that supports both, check https://gitee.com/BDWare/doip-sdk.git for reference. It is a java reference implementation of DOIP based on Netty framework, including a command-line client that can parse handle to LHS and send DOIP message; a repository that uses RocksDB as persistent storage, which can serve based on TLS, TCP, or UDP transport protocol respectively; a simple registry, which can match and search DO identifier according to keywords.

Before a DOIP service is deployed, it should register an identifier record that includes supported protocol, address, and the services port. When a DOIP service starts up, it should open at least two ports, one for DOIP 2.0 and at least one for this specification.

10. Informative References

[1] R. E. Kahn and V. Cerf, “The Digital Library Project (Volume 1): The World of Knowbots [Draft],” CNRI (1988), at http://www.cnri.reston.va.us/kahn-cerf-88.pdf

[2] R. E. Kahn, "The organization of computer resources into a packet radio network," Managing Requirements Knowledge, International Workshop (1975), at https://www.computer.org/csdl/proceedings/afips/1975/5083/00/50830177.pdf

[3] DOIP v2.0 Specification https://www.dona.net/sites/default/files/2018-11/DOIPv2Spec_1.pdf

[4] R.E. Kahn and R. Wilensky, “A Framework for Distributed Digital Object Services,” International Journal on Digital Libraries (2006), at https://www.doi.org/topics/2006_05_02_Kahn_Framework.pdf.

[5] See “System for uniquely and persistently identifying, managing and tracking digital objects,” U.S. Patent No. 6,135,646 (now expired).

[6] ITU-T Recommendation X.1255, “Framework for discovery of identity management information,” approved on September 4, 2013, at http://handle.itu.int/11.1002/1000/11951

[7] Digital Object Protocol Specification, version 1.0, CNRI (November 12, 2009), at http://dorepository.org/documentation/Protocol_Specification.pdf

[8] PKI-Public Key Infrastructure, see https://www.ssh.com/pki

[9] T. Dierks and C. Allen, “The Transport Layer Security (TLS) Protocol,” RFC 2246 (1999), at https://www.ietf.org/rfc/rfc2246.txt; T. Dierks and E. Rescorla, TLS, RFCs 4346 & 5246 (2006 & 2008), at https://www.ietf.org/rfc/rfc4346.txt and https://tools.ietf.org/html/rfc5246

[10] T. Bray, Ed., “The JavaScript Object Notation Format,” RFC 8259 (Dec. 2017), at https://tools.ietf.org/html/rfc8259

[11] The Unicode Standard, http://www.unicode.org/standard/standard.html

[12] F. Yergeau, “UTF-8, a transformation format of ISO 10646,” RFC 2279 (1998), at https://www.ietf.org/rfc/rfc2279.txt

[13] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, “A Roadmap for Transmission Control Protocol (TCP) Specification Documents,” RFC 7414 (2015), at http://tools.ietf.org/html/rfc7414

[14] N. Freed and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME),” RFCs 2045 & 2046 (1996), at https://tools.ietf.org/html/rfc2045 and https://tools.ietf.org/html/rfc2046; K. Moore, MIME, RFC 2047 (1996), at https://www.ietf.org/rfc/rfc2047.txt; N. Freed and J.Klensin, MIME, “Media Type Specifications and Registration Procedures,” IETF, RFCs 4288 & 4289 (2005), at https://tools.ietf.org/html/rfc4288 and https://tools.ietf.org/html/rfc4289

[15] D. Cooper, S. Santesson, S. Farrell, S. Boeyem, R. Housley and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280 (2008), at https://tools.ietf.org/html/rfc5280

[16] M. Jones, J. Bradley and N. Sakimura, “JSON Web Signature (JWS),” RFC 7515 (2015), at https://tools.ietf.org/html/rfc7515; M. Jones, JWS, “Unencoded Payload Option,” RFC 7787 (2016), at https://tools.ietf.org/html/rfc7797

[17] Y.4459: Digital entity architecture framework for Internet of things interoperability, https://www.itu.int/rec/T-REC-Y.4459-202001-I/en

[18] Kahn, Robert E., “The Role of Architecture in Internet Defense", America's Cyber Future:

Security and Prosperity in the Information Age, Center for a New American Security (CNAS), Volume II, Chapter XII, May 2011, http://www.cnri.reston.va.us/papers/CNAS_CyberSecurity_Kahn.pdf

[19] CNRI, RFC 3650: Handle System Overview, https://www.ietf.org/rfc/rfc3650.txt

[20] CNRI, RFC 3651: Handle System Namespace and Service Definition, https://www.ietf.org/rfc/rfc3651.txt

[21] CNRI, RFC 3652: Handle System Protocol (ver 2.1) Specification, https://www.ietf.org/rfc/rfc3652.txt

[22] Cohen, Danny. "On holy wars and a plea for peace." Computer 14.10 (1981): 48-54.

11. Authors' Addresses

Gang Huang
Key Lab of High-Confidence Software Technology (Peking University), MoE, PRC, Beijing, China, 100871
Email: hg@pku.edu.cn

Xuanzhe Liu
Key Lab of High-Confidence Software Technology (Peking University), MoE, PRC, Beijing, China, 100871
Email: xzl@pku.edu.cn

Yun Ma
Key Lab of High-Confidence Software Technology (Peking University), MoE, PRC, Beijing, China, 100871
Email: mayun@pku.edu.cn

Chaoran Luo
Key Lab of High-Confidence Software Technology (Peking University), MoE, PRC, Beijing, China, 100871
Email: luochaoran@pku.edu.cn

Xinjian Ma
Key Lab of High-Confidence Software Technology (Peking University), MoE, PRC, Beijing, China, 100871
Email: maxinjian@pku.edu.cn

12. COPYRIGHT LICENSE AGREEMENT

Copyright (C) ATSD (2021). All Rights Reserved.

This License Agreement (Agreement) is between the APPLICATION TECHNOLOGY & STANDARD DEVELOPMENT (ATSD), a non-profit, non-government international organization registered in Hong Kong, and the Individual or Organization (User) that has accessed, downloaded or implemented this Digital Object Interface Protocol Specification Version 2.1 (hereinafter called DOIP(v2.1)). ATSD is making DOIP(v2.1) available to User free of charge subject to the terms and conditions in the Agreement.

ATSD hereby grants User a non-exclusive, fully paid-up, irrevocable, world-wide license to reproduce, implement and further disseminate DOIP(v2.1) to the public, provided that the ATSD copyright notice and this Agreement are both retained in DOIP(v2.1), including any implementation made of it.

User hereby acknowledges that ATSD is making DOIP(v2.1) available to the public on an "AS IS" basis and ATSD MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, ATSD MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF FITNESS FOR ANY PARTICULAR PURPOSE, OR THAT THE USE OF DOIP(V2.1) WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

The Agreement will automatically terminate upon a material breach of its terms and conditions. Neither the names of the persons acknowledged as contributing to the preparation of DOIP(v2.1) nor the mark ATSD may be used in a trademark sense to endorse or promote products or services of User, or any third party.