draft G. Monroe
JanRain
August 9, 2006

OpenID DTP Version 1.0 Envelopes - Draft 02

Abstract

OpenID DTP is a protocol for sending, receiving, and relaying an arbitrary signed and encrypted payload between two endpoints. DTP Version 1.0 Part 1: Envelopes defines the envelope formats along with message generation, validation, and error handling.

Table of Contents

1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, B., “Key words for use in RFCs to Indicate Requirement Levels,” .).

2. Terminology

Identifier:	An Identifier is a URL or XRI.
User:	Either a sender, recipient, or relayer of messages. Users are represented within this protocol by Identifiers.
Payload:	An octet string of any length to be exchanged between Users.
Reciever Endpoint:	The software processing a recieved message.

3. XML Namespaces

The default namespace for XML fragments in this document without a namespace prefix is xmlns="http://www.example.com/2006/06/dtp#". Other namespace prefixes used include xmlns:xrd="xri://$xrd*($v*2.0)".

4. Messages

A Message consists of a set of nested envelopes. The outermost envelope is always an XML document with a root node of <OuterEnvelope>. The innermost envelope is always an XML document with a root node of <InnerEnvelope>.

It is possible to nest the innermost envelope in one or more relay documents before it is encrypted and put in the outermost envelope. This allows for relaying messages while maintaining the routing metadata, including signatures, with each relay.

4.1. The Inner Envelope

An Inner Envelope is an XML document looking something like this:

```
<?xml version="1.0"?>
<InnerEnvelope xmlns="http://www.example.com/2006/06/dtp#">
  <Recipient ...>
    .
    .
    .
  <Sender ...>
  <Data ...>
</InnerEnvelope>
```

The InnerEnvelope encapsulates the Payload along with information about the User sending the Payload and the one or more Users who should recieve it.

4.1.1. The Recipient Elements

Each Recipient element contains a <Identifier> element. The Identifier element contains the Identifier of the recipient, e.g.,

```
<Recipient>
  <Identifier>http://joe.example.com/</Identifier>
</Recipient>
```

4.1.2. The Sender Element

The Sender element has two children: a <Identifier> element and a <Fingerprint> element, e.g.,

```
<Sender>
  <Identifier>http://bob.example.com/</Identifier>
  <Fingerprint>
    h9LtI0I0ccY7JxcmEP0eFlGZV6Q=
  </Fingerprint>
</Sender>
```

The Identifier element contains the Identifier of the User sending the Payload. The contents of the Fingerprint element are described in Section 7.4 (Fingerprints).

4.1.3. The Data Element

The Data element encapsulates the Payload. Since the Payload is an octet string, it is first base64 encoded, and then set as the inner text of the Data element. The Type attribute of the Data element MUST be set to a URI denoting the type of the Payload.

Note: The Data element may have additional namespaced attributes for application-specific purposes.

4.2. The Outer Envelope

An Outer Envelope is an XML document looking something like this:

```
<?xml version="1.0"?>
<OuterEnvelope xmlns="http://www.example.com/2006/06/dtp#">
  <Recipient ...>
    .
    .
    .
  <Signature ...>
  <Data ...>
</OuterEnvelope>
```

An Outer Envelope contains an encrypted blob along with the information necessary to decrypt it and a signature.

4.2.1. The Recipient Elements

Each Recipient element contains two subelements: a <Fingerprint> element, and a <EncryptedCipherKey> element. The contents of the Fingerprint element are described in Section 7.4 (Fingerprints). The EncryptedCipherKey element contains the symmetric cipher key octet string encrypted for the recipient and base64 encoded, e.g.,

```
<Recipient>
  <Fingerprint>
    W3et8wAsnqP2CZjsELkn2nVyx5c=
  </Fingerprint>
  <EncryptedCipherKey
EncryptionAlgorithm="http://www.example.com/2006/06/dtp#rsa-oaep">
    R2EXPGIuZBNuoGoCM79uJG7nCzgeFq1b3VoZ4SdWcZiOGIjBdBt5N
    DyQWN3RvfohYna4NsZN6vUzSzL7S8ojCj7Ny1IlS4HYcwkOY7l6eK
    WM+B/tP40bVmqu8RDoVnziVuJOCO5eM/P1pnA4KTwntoz3SEqiZa8
    RebIhlRyUScE=
  </EncryptedCipherKey>
</Recipient>
```

A random octet string must be generated to use as the symmetric cipher key for encryption. The length of the octet string will vary depending on the symmetric cipher algorithm used.

The symmetric cipher key octet string is encrypted with the Recipient's Public Key using one of the algorithms described in Section 7.3 (Key Transport). The result of encrypting the symmetric cipher key is another octet string that must then be base64 encoded before being inserted in the <EncryptedCipherKey> element.

4.2.2. The Signature Element

The Signature element contains a signature of the plain text octet string resulting from decrypting the contents of the Data element. The signature is generated as described in Section 7.1.1 (Signature Generation). The result of generating the signature is a new octet string that must be base64 encoded before it is inserted into the <Signature> element.

The Algorithm attribute of the Signature element MUST be set to the identifier of the signature algorithm used.

4.2.3. The Data Element

The Data element encapsulates either an encrypted Inner Envelope XML Document or an encrypted Relay Envelope XML Document. Either way, the XML Document is treated as an octet string.

The XML Document to be encrypted is passed to a symmetric cipher algorithm (see Section 7.2 (Symmetric Cipher Algorithms)) along with the symmetric cipher key. The result of encryption is an octet string of cipher text.

The cipher text is then prefixed with the initial vector used by the symmetric cipher and base64 encoded. The CipherAlgorithm attribute of the Data element MUST be set to the identifier of the cipher algorithm used.

The Contents attribute of the Data element is a string, and MUST be set to a either inner or relay, denoting the type of its contents.

4.3. The Relay Envelope

A Relay Envelope is an XML document looking something like this:

```
<?xml version="1.0"?>
<RelayEnvelope xmlns="http://www.example.com/2006/06/dtp#">
  <Recipient ...>
    .
    .
    .
  <Relayer ...>
  <Signature ...>
  <Data ...>
</RelayEnvelope>
```

The Relay Envelope encapsulates either an Inner Envelope or another Relay Envelope. The nesting of Relay Envelopes represents the number of times the Inner Envelope has been relayed.

4.3.1. The Recipient Elements

The format of the Recipient elements in a Relay Envelope are identical to those of the Inner Envelope described in Section 4.1.1 (The Recipient Elements). The Recipients, however, will more than likely not be the same.

4.3.2. The Relayer Element

The Relayer element has two children: a <Identifier> element and a <Fingerprint> element, e.g.,

```
<Relayer>
  <Identifier>http://joe.example.com/</Identifier>
  <Fingerprint>
    W3et8wAsnqP2CZjsELkn2nVyx5c=
  </Fingerprint>
</Relayer>
```

The Identifier element contains the Identifier of the User relaying the Inner Envelope. The contents of the Fingerprint element are described in Section 7.4 (Fingerprints).

4.3.3. The Data Element

The Data element encapsulates either Inner Envelope XML Document or another Relay Envelope XML Document. Either way, the XML Document is treated as an octet string. It is first base64 encoded, and then set as the inner text of the Data element. The Contents attribute of the Data element is a string, and MUST be set to a either inner or relay, denoting the type of its contents.

4.3.4. The Signature Element

The Signature element contains a signature of the plain text octet string resulting from decoding the base64 encoded contents of the Data element. The signature is generated as described in Section 7.1.1 (Signature Generation). The result of generating the signature is a new octet string that must be base64 encoded before it is inserted into the <Signature> element.

The contents of the Signature element along with its attributes MAY have been extracted from the Signature element of the Outer Envelope XML Document that originally wrapped the XML Document enclosed by the Data element of this Relay Envelope. In other words, the signature may have been generated by someone other than the current Relayer.

5. Message Validation

Message validation always begins with an Outer Envelope. From there each message in the chain of nested envelopes must be validated, completing with validation of the Inner Envelope.

At any point during processing, if a document expected to be well-formed XML does no parse as such, this corresponds to the MALFORMED_XML error code listed in Section 6.1 (Error Codes).

Implementions SHOULD validate each envelope against the XML Schema listed in Appendix A (Message Schema) before any further processing. During processing, if a message does not validate against the XML Schema, does not contain elements as expected, or contains data that is not properly encoded, this corresponds to the XML_SCHEMA_MISMATCH error code listed in Section 6.1 (Error Codes).

5.1. Outer Envelope Validation

The Reciever Endpoint MUST know about at least one of the Recipients listed unless the CipherAlgorithm attribute of the Data element is set to http://www.example.com/2006/06/dtp#null. If the Reciever Endpoint does not have a the corresponding RSA Private Key associated with any of the listed fingerprints, this corresponds to the NO_KNOWN_RECIPIENTS error code listed in Section 6.1 (Error Codes).

For each Recipient element containing a Fingerprint for which the Reciever Endpoint has the corresponding private key, the encrypted symmetric cipher key should be decrypted using the private key, and the algorithm identified by the EncryptionAlgorithm attribute of the EncryptedCipherKey element. If the encryption algorithm identified by the EncryptionAlgorithm attribute is not known, this corresponds to the UNKNOWN_ALGORITHM error code listed in Section 6.1 (Error Codes).

The symmetric cipher key octet string resulting from decryption MUST be the same for all known recipients. If decryption results in more than one unique octet string, this corresponds to the CIPHER_KEY_MISMATCH error code listed in Section 6.1 (Error Codes).

Once the symmetric cipher key has been decrypted, it is used to decrypt the encapsulated XML Document. The ciphertext is extracted by decoding the base64 encoded contents of the Data element. The ciphertext is then decrypted according to the algorithm specified in the CipherAlgorithm attribute of the Data element using the symmetric cipher key resulting in an XML Document. If the symmetric cipher algorithm identified by the CipherAlgorithm attribute is not known, this corresponds to the UNKNOWN_ALGORITHM error code listed in Section 6.1 (Error Codes). The resulting XML Document can be identified as either a Relay or Inner Envelope by looking at the Contents attribute of the Outer envelope's Data element.

The signature octet string is extracted by decoding the base64 encoded contents of the Signature element. The Algorithm attribute of the Signature element indicates the algorithm used to generate the signature. If the signature algorithm identified by the Algorithm attribute is not known, this corresponds to the UNKNOWN_ALGORITHM error code listed in Section 6.1 (Error Codes).

To verify the signature over the decrypted XML Document, the document must first be parsed to extract the Identifier and Fingerprint from the Sender/Relayer element. If no public key is found for the (Identifier, Fingerprint) pair, this corresponds to the UNKNOWN_OUTER_SIGNER error code listed in Section 6.1 (Error Codes).

Using the public key associated with the (Identifier, Fingerprint) pair, the signature octet string, and the unmodified plain text resulting from decrypting the contents of the Data element, signature verification must be performed as described in Section 7.1.2 (Signature Verification). If signature verification fails, this corresponds to the BAD_OUTER_SIGNATURE error code listed in Section 6.1 (Error Codes).

5.2. Relay Envelope Validation

The base64 encoded contents of the Data element can be decoded to produce the relayed XML Document. The resulting XML Document can be identified as either a Relay or Inner Envelope by looking at the Contents attribute of this envelope's Data element.

The signature octet string is extracted by decoding the base64 encoded contents of the Signature element. The Algorithm attribute of the Signature element indicates the algorithm used to generate the signature. If the signature algorithm identified by the Algorithm attribute is not known, this corresponds to the UNKNOWN_ALGORITHM error code listed in Section 6.1 (Error Codes).

To verify the signature over the relayed XML Document, the document must first be parsed to extract the Identifier and Fingerprint from the Sender/Relayer element. If no public key is found for the (Identifier, Fingerprint) pair, this corresponds to the UNKNOWN_RELAY_SIGNER error code listed in Section 6.1 (Error Codes).

Using the public key associated with the (Identifier, Fingerprint) pair, the signature octet string, and the unmodified plain text resulting from decoding the contents of the Data element, signature verification must be performed as described in Section 7.1.2 (Signature Verification). If signature verification fails, this corresponds to th BAD_RELAY_SIGNATURE error code listed in Section 6.1 (Error Codes).

5.3. Inner Envelope Validation

The base64 encoded contents of the Data element can be decoded to produce the Payload. If the Payload type identified by the Type attribute is not known, this corresponds to the UNKNOWN_PAYLOAD error code listed in Section 6.1 (Error Codes).

6. Errors

In the event that validation of a message fails, in some instances, it MAY be possible to report the issue to the Sender/Relayer. For this we define a simple XML document and a set of error codes corresponding to the way in which validation failed.

An Error document has a root element of Error, e.g.,

```
  <?xml version="1.0"?>
  <Error Code="MALFORMED_XML"
           xmlns="http://www.example.com/2006/06/dtp#">
```

```
    h9LtI0I0ccY7JxcmEP0eFlGZV6Q=
  </Error>
```

The Error element has an attribute, Code, indicating the error that occured. The Error element contains a base64 encoded [SHA-1] (Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” .) digest of the bytes that validation was attempted on.

The Error XML document should be wrapped in an Outer envelope and signed with the private key associated with the Reciever Endpoint. The NULL encryption type should be used, and no Recipient elements are necessary.

6.1. Error Codes

MALFORMED_XML
: At any point, a document expected to be well-formed XML is not.

XML_SCHEMA_MISMATCH
: At any point, a document expected to be well-formed XML does not match the XML Schema listed in Appendix A (Message Schema).

NO_KNOWN_RECIPIENTS
: The Reciever Endpoint does not have the corresponding RSA Private Key for any of the fingerprints listed in the Outer Envelope's Recipient elements.

CIPHER_KEY_MISMATCH
: The result of decrypting the cipher key for each known recipient results in more than one unique octet string.

UNKNOWN_OUTER_SIGNER
: The Reciever Endpoint is unable to acquire the RSA Public Key identified by the (Identifier, Fingerprint) pair in the Sender/Relayer element of the envelope enclosed in the OuterEnvelope.

UNKNOWN_RELAY_SIGNER
: The Reciever Endpoint is unable to acquire the RSA Public Key identified by the (Identifier, Fingerprint) pair in the Sender/Relayer element of an envelope enclosed in a RelayEnvelope.

BAD_OUTER_SIGNATURE
: The result of running the signature verification algorithm on the enclosed envelope does not match the signature in the OuterEnvelope.

BAD_RELAY_SIGNATURE
: The result of running the signature verification algorithm on an enclosed envelope does not match the signature in a RelayEnvelope.

UNKNOWN_ALGORITHM
: An algorithm specified on an EncryptedCipherKey element, a Data element, or a Signature element is not supported by the Reciever Endpoint.

UNKNOWN_PAYLOAD
: The Type attribute of the Data element of the Inner envelope describing the Payload is not recognized.

7. Algorithms

This section describes the cryptographic algorithms used to digest, sign, and encrypt various portions of data used to construct messages.

7.1. Signatures

All implementations MUST support the RSA-SHA1 signature algorithms identified by http://www.example.com/2006/06/dtp#rsa-sha1. The RSA-SHA1 signature algorithms are described in RFC 2473 [PKCS1] (Kaliski, B. and J. Staddon, “PKCS #1: RSA Cryptography Specifications Version 2.0,” .), section 8.1.

7.1.1. Signature Generation

Section 8.1.1 of RFC 2437 [PKCS1] (Kaliski, B. and J. Staddon, “PKCS #1: RSA Cryptography Specifications Version 2.0,” .) describes a function RSASSA-PKCS1-V1_5-SIGN with two inputs, K and M. K is the Sender's or Relayer's RSA Private Key and M is the XML Document, treated as an octet string to be signed. The output of RSASSA-PKCS1-V1_5-SIGN is an octet string.

7.1.2. Signature Verification

Section 8.1.2 of RFC 2473 [PKCS1] (Kaliski, B. and J. Staddon, “PKCS #1: RSA Cryptography Specifications Version 2.0,” .) describes a function RSASSA-PKCS1-V1_5-VERIFY with three inputs, (n, e), M, and S. The pair, (n, e), represents the Sender's or Relayer's RSA Public Key, M is the octet string representing the signed envelope that was signed by the Sender/Relayer, and S is the signature to be verified.

7.2. Symmetric Cipher Algorithms

These ciphers are used with a symmetric cipher key to encrypt either an Inner Envelope or Relay Envelope XML Document before being included in the Data element of an Outer Envelope. Each of these algorithms is identified by a URI. All implementations MUST support the following ciphers.

7.2.1. AES 192 CBC

Identifier
: http://www.example.com/2006/06/dtp#aes192-cbc

AES is used in the Cipher Block Chaining (CBC) mode with a 192 bit key. This algorithm uses a 128 bit initialization vector.

7.2.2. AES 256 CBC

Identifier

    http://www.example.com/2006/06/dtp#aes256-cbc

AES is used in the Cipher Block Chaining (CBC) mode with a 256 bit key. This algorithm uses a 128 bit initialization vector.

7.2.3. NULL Encryption

Identifier

    http://www.example.com/2006/06/dtp#null

If the CipherAlgorithm attribute of the Data element of the Outer envelope is set to this identifier, the data is not encrypted. All that is needed to retrieve the encapsulated envelope is to decode the base64 encoded inner text of the Data element. No Recipient elements are needed in the outer envelope when this algorithm is specified.

7.3. Key Transport

The symmetric cipher key used during symmetric encryption (Symmetric Cipher Algorithms) must be encrypted for each recipient. To do this, a key transport algorithm using the recipients public key must be used.

7.3.1. RSA-OAEP

Identifier

    http://www.example.com/2006/06/dtp#rsa-oaep

The RSAES-OAEP algorithm, specified in section 7.1. of RFC 2437 [PKCS1] (Kaliski, B. and J. Staddon, “PKCS #1: RSA Cryptography Specifications Version 2.0,” .), is used to encrypt and decrypt the symmetric cipher key for each Recipient.

The value input to the key transport function SHOULD be calculated using [SHA-1] (Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” .) and the empty octet string, and MGF1 (with SHA1) SHOULD be used for mask generation during RSAES-OAEP-ENCRYPT and RSAES-OAEP-DECRYPT.

7.4. Fingerprints

A Fingerprint is a digest of a DER-encoded RSA Public Key. These fingerprints can be used in combination with an Identifier to aid in caching of Public Keys. The <Fingerprint> element contains a base64 encoded digest.

There is only one type of signature defined. The fingerprint is a [SHA-1] (Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” .) digest of the DER-encoded public key and is identified by the URI, http://www.example.com/2006/06/dtp#der-sha1.

8. Normative References

[PKCS1]	Kaliski, B. and J. Staddon, “PKCS #1: RSA Cryptography Specifications Version 2.0,” RFC 2437.
[RFC2119]	Bradner, B., “Key words for use in RFCs to Indicate Requirement Levels.”
[SHA-1]	Eastlake, D. and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” RFC 3174.

Appendix A. Message Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<schema targetNamespace="http://www.example.com/2006/06/dtp#"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:dtp="http://www.example.com/2006/06/dtp#"
        blockDefault="#all"
        elementFormDefault="qualified"
        version="1.0"
        xml:lang="EN" >

  <complexType name="SignatureType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute name="Algorithm" type="anyURI" use="required"/>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="EncryptedCipherKeyType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute name="EncryptionAlgorithm" type="anyURI"
                   use="required"/>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="SimpleAddressType">
    <sequence>
      <element name="Identifier" type="anyURI"/>
    </sequence>
  </complexType>

  <simpleType name="FingerprintType">
    <restriction base="base64Binary"/>
  </simpleType>
  <complexType name="FingerprintedAddressType">
    <complexContent>
      <extension base="dtp:SimpleAddressType">
        <sequence>
          <element name="Fingerprint" type="dtp:FingerprintType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <simpleType name="DataType">
    <restriction base="base64Binary"/>
  </simpleType>

  <simpleType name="ContentsAttrType">
    <restriction base="string">
      <enumeration value="inner"/>
      <enumeration value="relay"/>
    </restriction>
  </simpleType>

  <complexType name="RelayDataType">
    <simpleContent>
      <extension base="dtp:DataType">
        <attribute name="Contents" type="dtp:ContentsAttrType"
                   use="required"/>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="AnnotatedDataType">
    <simpleContent>
      <extension base="dtp:DataType">
        <attribute name="Type" type="anyURI" use="required"/>
        <anyAttribute namespace="##other" processContents="lax"/>
      </extension>
    </simpleContent>
  </complexType>

  <attributeGroup name="EncryptedDataAttrGroup">
    <attribute name="Contents" type="dtp:ContentsType"
               use="required"/>
    <attribute name="CipherAlgorithm" type="anyURI"
               use="required"/>
  </attributeGroup>

  <complexType name="EncryptedDataType">
    <simpleContent>
      <extension base="dtp:DataType">
        <attributeGroup ref="dtp:EncryptedDataAttrGroup"/>
      </extension>
    </simpleContent>
  </complexType>

  <element name="InnerEnvelope">
```

```
    <complexType>
      <sequence>
        <element name="Recipient" type="dtp:SimpleAddressType"
                 maxOccurs="unbounded"/>
        <element name="Sender" type="dtp:FingerprintedAddressType"/>
        <element name="Data" type="dtp:AnnotatedDataType" />
      </sequence>
    </complexType>
  </element>

  <element name="RelayEnvelope">
    <complexType>
      <sequence>
        <element name="Recipient"
                 type="dtp:SimpleAddressType"
                 maxOccurs="unbounded"/>
        <element name="Relayer" type="dtp:FingerprintedAddressType"/>
        <element name="Signature" type="dtp:SignatureType"/>
        <element name="Data" type="dtp:RelayDataType"/>
      </sequence>
    </complexType>
  </element>

  <element name="OuterEnvelope">
    <complexType>
      <sequence>
        <element name="Recipient" minOccurs="0"
                 maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Fingerprint"
                       type="dtp:FingerprintType"/>
              <element name="EncryptedCipherKey"
                       type="dtp:EncryptedCipherKeyType" />
            </sequence>
          </complexType>
        </element>
        <element name="Signature" type="dtp:SignatureType"/>
        <element name="Data" type="dtp:EncryptedDataType"/>
      </sequence>
    </complexType>
  </element>

  <simpleType name="ErrorCodeType">
    <restriction base="string">
      <enumeration value="CIPHER_KEY_MISMATCH"/>
      <enumeration value="NO_KNOWN_RECIPIENTS"/>
      <enumeration value="UNKNOWN_OUTER_SIGNER"/>
      <enumeration value="UNKNOWN_RELAY_SIGNER"/>
      <enumeration value="BAD_OUTER_SIGNATURE"/>
      <enumeration value="BAD_RELAY_SIGNATURE"/>
      <enumeration value="MALFORMED_XML"/>
      <enumeration value="XML_SCHEMA_MISMATCH"/>
      <enumeration value="UNKNOWN_ALGORITHM"/>
      <enumeration value="UNKNOWN_PAYLOAD"/>
    </restriction>
  </simpleType>

  <element name="Error">
    <complexType>
      <simpleContent>
        <extension base="base64Binary">
          <attribute name="Code" type="dtp:ErroCodeType"
                     use="required"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

</schema>
```

Author's Address

Grant Monroe
JanRain, Inc.
5331 SW Macadam Avenue
Suite #375
Portland, OR 97239
USA
Email: grant@janrain.com