Network Working Group
Internet-Draft
Intended status: Informational
Expires: 9 April 2026

P. M. Hallam-Baker
ThresholdSecrets.com

6 October 2025

Data At Rest Envelope (DARE)
draft-hallambaker-dare-00

Abstract

This document describes the Data At Rest Envelope (DARE) Envelope and Sequence. The DARE Envelope syntax is used to package plaintext or encrypted content payload and metadata with JOSE decryption and signature data authenticating the payload and metadata. The DARE Sequence format comprises a sequence of DARE Envelopes.

The binary encoding of DARE envelopes allows compact encoding of encrypted and signed payloads of unknown length in a single pass with minimal overhead. The binary encoding of DARE sequences is designed to support an append only write mode and efficient reading in either the forwards or backwards directions. The JSON encoding of DARE Envelopes and sequences allows envelopes and sequences to be incorporated in other JSON objects.

This document does not cover construction of indexes or signatures over multiple entries in a sequence.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

This document describes the Data At Rest Envelope (DARE) Envelope and Sequence. The DARE Envelope syntax is used to package plaintext or encrypted content payload and metadata with JOSE decryption and signature data authenticating the payload and metadata. The DARE Sequence format comprises a sequence of DARE Envelopes.

Two serializations are defined, a JSON serialization allowing envelopes and sequences to be included in other JSON objects and a binary serialization employing a simple {length, data} approach. This avoids the overhead of Base64 encoding signed and encrypted components.

The envelope and sequence serializations are optimized for different purposes:

* The DARE envelope binary serialization allows payloads of unknown length to be efficiently encoded in a single pass with bounded memory overhead.

* The DARE sequence binary serialization is designed to support append only writes and sequential reads in either the forward or the reverse direction.

The envelope and sequence approaches are complimentary in that enveloped data MAY be efficiently added to a sequence and an individual entry in a sequence MAY be extracted as an envelope.

A DARE Envelope is a sequence of exactly four fields:

* Unsigned header

* Signed header

* Encrypted or plaintext payload

* Trailer

The unsigned header contains all the information required to process the signed header and payload. This includes the decryption information (if required) and the digest algorithm used to process the payload (if required). The signature value itself MAY be specified in either the unsigned header or the trailer.

A DARE Sequence is a sequence of DARE envelope entries in which each entry contains exactly three fields, the contents of the unsigned header and trailer being combined:

* Unsigned header

* Signed header

* Encrypted or plaintext payload

1.1. Serialization

Two serialization formats are supported, a JSON serialization to allow DARE data to be exchanged within JSON objects and a binary serialization that avoids the need for Base64 encoding of cryptographic inputs and outputs. Unlike CBOR and the numerous flavors of 'Binary JSON' encodings, the DARE binary serialization is limited to specifying opaque binary blobs containing either JSON objects (headers, trailers) or payload data.

1.1.1. JSON Serialization

The JSON serialization of a DARE envelope consists of an array of four entries in which the unsigned header and trailer are represented as a JSON object and the signed header and payload are represented as a string containing the Base64 encoding of the field.

```
Minimal DARE Envelope: Envelope
[null,
  "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
  "VGhpcyBpcyBhIHRlc3QgZm9yIERhdGEgQXQgUmVzdCBFbnZlbG9wZQ",
  null
  ]
```

The envelope serialization is designed to allow efficient encoding of streamed data of unknown length without the need for buffering with the signature information being split between the unsigned header and the trailer.

The JSON serialization of a DARE sequence consists of a sequence of envelope entries. Each envelope containing three entries, an unsigned header, a signed header and a payload.

```
Minimal DARE Sequence: Sequence
[[null,
    "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
    "VGhpcyBpcyBhIHRlc3QgZm9yIERhdGEgQXQgUmVzdCBFbnZlbG9wZQ",
    null
    ]
  ]
```

The sequence serialization does not support separation of the signature data

1.1.2. Binary Serialization

The binary serialization of the envelope consists of a Type Identifier sequence specifying a DARE Envelope followed by the same four entries from the JSON encoding encoded as a series of {length, data} items in which the length entries are encoded as variable length integers.

Minimal DARE Envelope: Envelope

```
F8 00 18 7B  0A 20 20 22  63 74 79 22  3A 20 22 74
65 78 74 2F  70 6C 61 69  6E 22 7D 28  54 68 69 73
20 69 73 20  61 20 74 65  73 74 20 66  6F 72 20 44
61 74 61 20  41 74 20 52  65 73 74 20  45 6E 76 65
6C 6F 70 65  00 00
```

The binary serialization of a DARE Sequence follows the same pattern. The sequence begins with a Type Identifier sequence specifying a DARE Sequence followed by a sequence of wrapped frames, each consisting of three parts:

* Forward length indicator

* Envelope entry

* Reverse length indicator

The forward and reverse length indicators are both encoded as variable length integers but with the bytes of the reverse length indicator written backwards to enable the sequence to be read efficiently in either the forward or reverse direction.

Minimal DARE Sequence: Sequence

```
F9 00 40 43  00 18 7B 0A  20 20 22 63  74 79 22 3A
20 22 74 65  78 74 2F 70  6C 61 69 6E  22 7D 28 54
68 69 73 20  69 73 20 61  20 74 65 73  74 20 66 6F
72 20 44 61  74 61 20 41  74 20 52 65  73 74 20 45
6E 76 65 6C  6F 70 65 43  40
```

1.2. JOSE Signature and Encryption

The DARE Envelope and Sequence constructions make use of JOSE signature and encryption but with modifications designed to support efficient processing of very large payloads and to facilitate use in incremental encryption of sequences.

In the DARE model, an envelope presents all the information necessary to process the payload. In a typical workflow a processor needs to know what type of content is to be processed and which party vouches for it before processing the payload itself.

The security enhancements supported are correspondingly limited, the only field that can be encrypted is the payload field and the signature value is always computed over a manifest consisting of the digest of the signed header and the digest of the envelope payload

value. If a different processing order is required, this is achieved by nesting. While nested constructions result in a very serious space overhead when using JSON encoding (33% increase in encoding size per pass), this is not a concern when the binary encoding is used.

1.3. Extensions

The DARE Sequence Format is intended to be used as the basis for append only data structures offering a wide range of index and signature capabilities. These capabilities are outside the scope of this document as the range of approaches is very large.

For example, a simple file archive format based on DARE Sequence might consist of an initial entry describing the collection as a whole, a sequence of entries containing data from a single file and a final entry containing an index of the entire collection and a signature over the list of digest values of each entry. This approach is optimal for read access but incurs a penalty on writes.

Applications involving frequent write operations are likely to be better served by some form of Haber-Stornetta chained digest construction. Merkel Tree offers an acceptable balance of space efficiency, read performance and write performance for most purposes but is not necessarily ideal for all. A system logging application in which a server writes out reports on each transaction as it is performed, might be better served by a construction in which large numbers of entries are authenticated in a chained digest construction, the result of which is periodically signed by some relevant authority.

2. Definitions

2.1. Related Specifications

The DARE Envelope and Sequence formats are based on the following existing standards and specifications.

Message syntax The cryptographic processing model is based on JSON Web Signature (JWS) [RFC7515], JSON Web Encryption (JWE) [RFC7516] and JSON Web Key (JWK) [RFC7517].

Cryptographic primitives. The HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] and Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] are used.

Cryptographic algorithms The cryptographic algorithms and

identifiers described in JSON Web Algorithms (JWA) [RFC7518] are used together with additional algorithms as defined in the JSON Object Signing and Encryption IANA registry [IANAJOSE].

2.2. Requirements Language

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].

2.3. Defined terms

The terms "Authentication Tag", "Content Encryption Key", "Key Management Mode", "Key Encryption", "Direct Key Agreement", "Key Agreement with Key Wrapping" and "Direct Encryption" are defined in the JWE specification [RFC7516].

The terms "Authentication", "Ciphertext", "Digital Signature", "Encryption", "Initialization Vector (IV)", "Message Authentication Code (MAC)", "Plaintext" and "Salt" are defined by the Internet Security Glossary, Version 2 [RFC4949].

Annotated Envelope A DARE Envelope that contains an Annotations field with at least one entry.

Authentication Data A Message Authentication Code or authentication tag.

Complete Envelope A DARE envelope that contains the key exchange information necessary for the intended recipient(s) to decrypt it.

Detached Envelope A DARE envelope that does not contain the key exchange information necessary for the intended recipient(s) to decrypt it.

Encryption Context The master key, encryption algorithms and associated parameters used to generate a set of one or more enhanced data sequences.

Enhancement Applying a cryptographic operation to a data sequence. This includes encryption, authentication and both at the same time.

Generator The party that generates a DARE envelope.

Group Encryption Key A key used to encrypt data to be read by a group of users. This is typically achieved by means of some form of proxy re-encryption or distributed key generation.

Group Encryption Key Identifier A key identifier for a group encryption key.

Master Key (MK) The master secret from which keys are derived for authenticating enhanced data sequences.

Recipient Any party that receives and processes at least some part of a DARE envelope.

Related Envelope A set of DARE envelopes that share the same key exchange information and hence the same Master Key.

Uniform Data Fingerprint (UDF) The means of presenting the result of a cryptographic digest function over a data sequence and content type identifier specified in the Uniform Data Fingerprint specification [draft-hallambaker-mesh-udf]

3. Architecture

DARE Envelope and DARE Sequence provide a means of efficiently serializing cryptographic payloads without the complexity of a full binary serialization format such as ASN.1 or CBOR.

The cryptographic processing model follows the same pattern as JSON Web Encryption and JSON Web Signature but only supporting modern cryptographic algorithms and cipher modes:

* Key Exchange mechanisms MUST support use of context information.

* Signature mechanisms are used in pure mode.

* Bulk encryption algorithms MUST support Authenticated Encryption with Associated Data (AEAD).

3.1. DARE Envelope

A DARE envelope is a sequence of exactly four fields:

Unsigned header MUST provide all the information required to begin processing the security enhancements to the payload. This includes the digest algorithm (if used) and the information required to decrypt the payload. MAY provide the signature and/or digest values.

Signed header MAY be used to specify content metadata and/or any other information to be included within the scope of the signature.

Payload Contains the content data in the case of an unencrypted envelope and the encrypted content data and associated integrity tag (if required) in the case of encrypted data.

Trailer MUST be present, even if empty. MAY be used to specify the signature value(s).

Allowing signature data to be split between the unsigned header and the trailer allows implementations to support efficient single pass processing of streamed data of indeterminate length without the need to buffer large quantities of data.

3.1.1. Security Enhancements

A core principle of the DARE architecture is that the security enhancements all pertain to the sequence of octets specified in the envelope payload.

* If the envelope is signed, the signature is computed over the digest of the payload octets. That is, the ciphertext if the envelope is encrypted and the content data itself otherwise.

* The headers and trailer are always cleartext.

If an application requires the combination of encryption and a signature over the plaintext, this MAY be achieved through nesting with an outer encrypted envelope containing an inner envelope with the content plaintext and signature data. The same approach MAY be applied to encrypt content metadata.

3.2. DARE Sequence

A DARE Sequence is a sequence of DARE Envelope entries consisting of three fields:

Unsigned header MUST provide all the information required to process the security enhancements to the payload including the signature values.

Signed header As for envelope Signed Header.

Payload As for envelope payload except that the payload is encoded as a single chunk.

Splitting the signature data into two parts does not provide a useful advantage when processing sequence entries as each envelope entry is wrapped in a frame beginning and ending with fixed length specifiers to allow efficient read operations in either the forwards or reverse directions.

4. Serialization

Two serialization formats are defined

JSON Serialization Yada, 33% payload expansion

Binary Serialization Very small overhead, minimum of 6 bytes, maximum of 40 bytes + 8 bytes per chunk.

4.1. JSON Serialization

DARE Envelopes and Sequences are encoded as JSON sequences.

4.1.1. Dare Envelope

A DARE Envelope in JSON serialization consists of a sequence with four entries:

Unsigned header The unsigned header as a JSON object.

Signed header A string containing the base64url encoded value of the JSON serialization of the signed header.

Encrypted or plaintext payload A string containing the base64url encoded value of the JSON serialization of the payload.

Trailer The trailer as a JSON object or the value null.

For example, the following envelope contains a payload containing the text ’This is a test’ and a header specifying that the content type is text/plain:

```
Envelope
[null,
  "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
  "VGhpcyBpcyBhIHRlc3Q",
  null
  ]
Signed Header:
{
  "cty": "text/plain"}
Payload:
This is a test
```

4.1.2. Dare Sequence

A DARE Sequence in JSON serialization consists of a sequence of sequences. Each entry in the outer sequence containing a single envelope entry with exactly three fields:

Unsigned header The unsigned header as a JSON object.

Signed header A string containing the base64url encoded value of the JSON serialization of the signed header.

Encrypted or plaintext payload A string containing the base64url encoded value of the JSON serialization of the payload.

For example, the following sequence contains the envelopes presented in the previous two examples:

```
Sequence
[[null,
    "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
    "VGhpcyBpcyBhIHRlc3QgZm9yIERhdGEgQXQgUmVzdCBFbnZlbG9wZQ",
    null
    ],
  [null,
    "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
    "VGhpcyBpcyBhIHRlc3Q",
    null
    ]
  ]
```

4.2. Binary Serialization

The binary serialization is the preferred serialization for DARE Envelopes and Sequences allowing use of cryptographic enhancements without the need for Base64 encoding of cryptographic inputs or outputs.

4.2.1. Type Identifier

Type Identifiers are a sequence of zero or more odd octets (bit 0 is set) terminated by a single even octet (bit 0 is clear). This definition allows for an unlimited number of type identifiers to be defined.

This document specifies the following Type Identifiers:

Type	Purpose
[248]	DARE Envelope with JSON metadata
[249][00]	DARE Sequence with JSON metadata

Table 1

The assignment of a single byte Type Identifier for DARE Envelope is justified by the need for compactness when used in nested encodings and to allow an entire envelope to specified in highly constrained messaging contexts (e.g. NFC tag, QR Code).

Since it is the nature of sequences to contain large numbers of items, this use does not justify assignment of a single byte identifier.

Applications MAY specify additional Type Identifiers to be used in place of the defaults above to specify particular data formats using DARE Encodings. For example, a content archive format based on DARE Sequence might use its own Type Identifier so that files can be identified as being in that particular format.

4.2.2. Varint

It is intended that the variable length integer encoding used in DARE be the same as the encoding used in MOQ. Currently, MOQ uses the same encoding as QUIC which allows one byte encoding of the integers 0-63, but there is an outstanding proposal to switch to an encoding that would allow single byte encoding of the integers 0-127.

When used to specify the lengths of DARE sequence frames, the octets of the final, varint are written in reverse order to allow the value to be read backwards. This encoding is referred to in this document as a _ravint_.

For convenience of readers, the QUIC definition, as used in the examples is presented here:

The QUIC variable-length integer encoding reserves the two most significant bits of the first byte to encode the base-2 logarithm of the integer encoding length in bytes. The integer value is encoded on the remaining bits, in network byte order.

This means that integers are encoded on 1, 2, 4, or 8 bytes and can encode 6-, 14-, 30-, or 62-bit values, respectively. The following summarizes the encoding properties.

2MSB	Length	Usable Bits	Range
00	1	6	0-63
01	2	14	0-16383
10	4	30	0-1073741823
11	8	62	0-4611686018427387903

Table 2

4.2.3. Known Length Field

All fields in a DARE Envelope or DARE Sequence envelope entry except for the payload are serialized as known length fields. This comprises a Length: Data form as follows:

Length A varint specifying the number of data octets that follow.

Data A sequence of [length] octets.

4.2.4. Variable Length Field

The payload field in a DARE Envelope or DARE Sequence envelope entry is serialized as a variable length field. This consists of zero or more known length fields of length greater than zero followed by the varint zero.

The payload data consists of the concatenation of the data sections of the fixed length fields.

4.2.5. Frame wrapper

A frame wrapper consists of a sequence of three entries:

Forward length The number of octets of frame data

Frame data entry A sequence of [Forward Length] octets.

Reverse length number of octets of frame data written as a ravint.

The value of the forward and reverse length indicators MUST be equal and MUST use the same number of encoding bytes.

4.2.6. Envelope Serialization

For example, the binary encoding of the envelope shown in the previous example is:

Envelope

```
F8 00 18 7B  0A 20 20 22  63 74 79 22  3A 20 22 74
65 78 74 2F  70 6C 61 69  6E 22 7D 0E  54 68 69 73
20 69 73 20  61 20 74 65  73 74 00 00
```

4.2.7. Sequence Serialization

For example, the binary encoding of the sequence shown in the previous example is: ~~~~ DARE Sequence in Binary Serialization: Sequence

F9 00 40 43 00 18 7B 0A 20 20 22 63 74 79 22 3A

0. 69 73 20 69 73 20 61 20 74 65 73 74 20 66 6F

1. 20 44 61 74 61 20 41 74 20 52 65 73 74 20 45

2. E 76 65 6C 6F 70 65 43 40 29 00 18 7B 0A 20 20

3. 63 74 79 22 3A 20 22 74 65 78 74 2F 70 6C 61

4. 6E 22 7D 0E 54 68 69 73 20 69 73 20 61 20 74

5. 73 74 29 ~~~~

5. Encryption

DARE Encryption is designed to allow the use of a single key exchange to encrypt multiple envelope entries in a sequence. This avoids the computation and data overhead of repeated public key operations.

JOSE Web Encryption supports use of the industry standard algorithms and approaches current at the time development began. DARE Encryption narrows the scope of the supported encryption algorithms and cipher modes to those considered state of the art in 2025. Instead of supporting many approaches to key exchange to avoid unnecessary processing, the same key exchange construction is used regardless of the number of recipients or the number of items to be encrypted:

* The symmetric cipher mode used MUST provide Authenticated Encryption with Associated Data.

* Support for the X25519 and X448 algorithms is REQUIRED

The DARE encryption process has two stages:

Key Exchange A random value that will become the exchanged key is generated using a sufficiently random technique and assigned a unique key identifier.

An ephemeral public key pair is generated for each of the intended public encryption keys, the ephemeral key and encryption key is then used to establish a shared secret which is in turn used as an encryption key in a key wrapping operation on the exchanged key.

Encryption Parameters Derivation A salt value is computed for each plaintext to be enciphered. It is sufficient that the salt value be guaranteed to be unique for each plaintext.

A Key Derivation Function is used to combine the exchanged key and the salt to derive the encryption key and initialization vector to be used to encrypt the plaintext to obtain the ciphertext.

5.1. Algorithms Suites

The algorithms used in these examples are sub-optimal. The following table shows the algorithms currently used and the proposed replacements.

Purpose	Current	Proposed
Key Identifier	UDF of PKIX subjectKeyInfo	UDF of JOSE Thumbprint
Key Wrap	A256KW	A256GCMKW
Key Derivation Function	SHAKE256	SHAKE256

Table 3

The rationale for these changes is that the SHAKE functions defined in SHA3 offer a considerably simpler approach than the HKDF approach used in the earlier version of DARE.

5.2. Key Exchange

The key exchange process comprises the following steps:

* A random value that will become the exchanged key is generated using a sufficiently random technique and assigned a unique key identifier.

* An ephemeral public key pair is generated for each of the intended public encryption keys, the ephemeral key and encryption key is then used to establish a shared secret which is in turn used as an encryption key in a key wrapping operation on the exchanged key.

5.2.1. Exchanged Key Generation

The exchanged key is a random sequence of octets containing at least as many bits as the symmetric cipher encryption key. Each exchanged key is assigned a unique identifier.

The generation technique MUST ensure that the exchanged key is:

* Sufficiently random

* Does not leak information that might be useful to an attacker.

One means of satisfying these constraints is to use a strong Key Derivation Function to combine a random value provided by the platform with a secret value obtained from an independent source. This ensures that the exchanged key cannot be used as a side channel. [Move this to an appendix and reference each time]

The Key Exchange information comprises:

* The Exchanged Key Identifier [Optional].

* The list of Recipient Key Information entries.

In the case that the exchanged key is only used to encrypt the payload of the envelope in which the Key Exchange Information is specified, the Exchanged Key Identifier MAY be omitted.

Exchanged Key
14 C3 88 28 3F 62 FC 2D 09 77 5D 02 BD B3 79 8C
F0 AF 8A 8B 4F 73 F0 2C CB ED D3 24 C6 E2 EF 80
Exchanged Key Identifier
MCFX-KXN4-4Z6Q-D2OO-AJW2-FRLH-PDIV

5.2.2. Recipient Data Generation

Recipient key information is calculated for each decryption key as follows:

* An ephemeral key is generated with the same algorithm parameters as the decryption key.

* The private component of the ephemeral key and the decryption key are used to compute the shared secret for the recipient.

* A KDF is used to calculate the key wrap parameters for the recipient.

* The key wrap algorithm and parameters are applied to the Exchanged Key to create the wrapped key data for the recipient.

The recipient key information comprises:

* The Recipient Key Identifier.

* The public component of the ephemeral key.

* The wrapped exchanged key data.

* The key wrap algorithm identifier.

```
Recipient Public Key
MAY4-Y4CP-ZNS5-XUIB-2ZYL-QVRI-UTC3
Ephemeral Key Pair: Envelope
{
  "crv": "X25519",
  "Public": "HNDDtrjgh7VqFhMD2zMmxeoN3dan1Us-KWVyxLHcODE"}
SharedSecret
  19 6A 8E FF  00 81 D8 F8  CD E6 3C 1D  40 29 7D 4E
  5D 4E F4 19  4D C6 CF CD  0E E9 7A 80  49 C6 0B 14
Wrapped Exchanged Key
  E8 BC CC 09  B1 AB 26 86  EB CF 90 F4  C7 3A AF D7
  AE 73 B3 9C  87 B1 15 68  B9 58 BB D1  13 BD 82 B4
  59 5D 77 FF  BE 17 86 44
Recipient Information: Envelope
{
  "kid": "MAY4-Y4CP-ZNS5-XUIB-2ZYL-QVRI-UTC3",
  "epk": {
    "PublicKeyECDH": {
      "crv": "X25519",
      "Public": "HNDDtrjgh7VqFhMD2zMmxeoN3dan1Us-KWVyxLHcODE"}},
  "wmk": "6LzMCbGrJobrz5D0xzqv165zs5yHsRVouVi70RO9grRZXXf_vheGRA"}
```

5.3. Payload Encryption

5.3.1. Encryption Parameters Derivation

The encryption key and nonce are obtained by applying a Key Derivation Function to the concatenation of a salt value and the exchanged key.

The Associated Data used in the AEAD encryption algorithm is the Signed Header.

5.3.2. Secure salt construction

Implementations MUST ensure that each salt value used is unique.

```
Salt
  93 E5 A0 2B  93 93 A6 6B  8B BF B7 B0  28 DF 00 F1
  3E 69 47 6E  AD FB 31 3E  B2 C7 02 10  A4 84 2E 19
```

5.3.3. Salt erasure

If the salt value contains sufficient unpredictable information, erasure of the salt value in a data sequence renders decryption of the payload value infeasible.

5.4. Payload Format

The input to the KDF is the concatenation of the Salt value and the Exchanged Key:

```
KDF Input
  93 E5 A0 2B  93 93 A6 6B  8B BF B7 B0  28 DF 00 F1
  3E 69 47 6E  AD FB 31 3E  B2 C7 02 10  A4 84 2E 19
  14 C3 88 28  3F 62 FC 2D  09 77 5D 02  BD B3 79 8C
  F0 AF 8A 8B  4F 73 F0 2C  CB ED D3 24  C6 E2 EF 80
```

Since we require a 256 bit key and 96 bit nonce, 352 bits of KDF output are required:

```
KDF Output = SHAKE256(KDF Input,352)
  7A E2 8F C3  A4 FE 57 FC  CA C4 E5 D2  B3 4D CF F2
  3F DC 51 8A  FC D7 B7 D9  59 B3 29 37  CF 77 45 EA
  0A D8 DA 5C  34 A1 45 5C  EB D9 20 1E
```

The first 96 bits of the output provide the nonce, the following 256 bits provide the key:

```
Encryption Nonce
  7A E2 8F C3  A4 FE 57 FC  CA C4 E5 D2
Encryption Key
  B3 4D CF F2  3F DC 51 8A  FC D7 B7 D9  59 B3 29 37
  CF 77 45 EA  0A D8 DA 5C  34 A1 45 5C  EB D9 20 1E
```

The plaintext payload is the string ’This is a test’. The ciphertext is obtained by applying AES-GCM to the plaintext payload using the key and nonce derrived using the KDF and the Signed header as Associated Data.

```
Associated Data
  7B 0A 20 20  22 63 74 79  22 3A 20 22  74 65 78 74
  2F 70 6C 61  69 6E 22 7D
Ciphertext
  7F 34 BA 07  B7 41 83 62  4A 50 1A 8C  4E 12 0E 53
  FC 29 E6 5D  BE 8B D5 39  12 A9 A0 84  10 01 97 B1
  B0 43 F6 9A  8E 87 24 FB  D7 8E A8 B8  71 93 CA 8C
  4E 29 AA 23  3C 6C 33 01
```

The complete encryption envelope is:

```
DARE Envelope With Encrypted Payload: Envelope
[{
    "enc": "A256GCM",
    "Salt": "k-WgK5OTpmuLv7ewKN8A8T5pR26t-zE-sscCEKSELhk",
    "recipients": [{
        "kid": "MAY4-Y4CP-ZNS5-XUIB-2ZYL-QVRI-UTC3",
        "epk": {
          "PublicKeyECDH": {
            "crv": "X25519",
            "Public": "HNDDtrjgh7VqFhMD2zMmxeoN3dan1Us-KWVyxLHcOD
  E"}},
        "wmk": "6LzMCbGrJobrz5D0xzqv165zs5yHsRVouVi70RO9grRZXXf_v
  heGRA"}
      ]},
  "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
  "fzS6B7dBg2JKUBqMThIOU_wp5l2-i9U5EqmghBABl7GwQ_aajock-9eOqLhxk8
  qMTimqIzxsMwE",
  null
  ]
```

6. Signature

The DARE signature scheme is designed to provide a simple and efficient means of signing a DARE Envelope or Envelope Entry Payload and associated content metadata. As with the approach to encryption, the opportunity is taken to make use of the full capabilities of modern signature algorithms:

* The signature algorithm MUST support context strings.

* The signature value is computed over a manifest consisting of a digest algorithm identifier, the digest of the signed header, and the digest of the payload bytes using the pure version of the signature algorithm (not pre-hashed).

The DARE signature approach intentionally removes much of the flexibility provided by the JWS signature approach.

6.1. Signature Preamble

Signed DARE Envelopes MAY separate signature data presenting only the information required to process the signature

* The digest algorithm used to digest the Signed Header and Payload.

* The Key Identifier of the Signature Key

* An application context specifier (optional)

```
Signature Preamble: Envelope
{
  "signatures": [{
      "dig": "SHA3512",
      "alg": "ED25519",
      "kid": "MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN"}]}
```

6.2. Calculating the Signature Value

The signature algorithm takes three inputs:

* The private signature key

* The signature manifest

* A Context String denoting the purpose for which the signature was created.

6.2.1. Signature Manifest

The signature manifest consists of the concatenation of the following octet sequences:

* The ASCII encoding of the digest algorithm identifier.

* A zero octet.

* The digest value of the unsigned header octets as specified in the envelope.

* The digest value of the payload octets as specified in the envelope.

Note that the signature value presented in DARE envelope is always a signature over the payload octets presented in that envelope. If the envelope is signed and encrypted, the signature value is calculated over the ciphertext, not the plaintext.

If the application requires a signature over a plaintext payload, the only way this can be achieved using DARE envelope is to nest an envelope containing the signature information inside an envelope containing the encryption information. This approach has the advantage of ensuring that the signature information does not leak any information regarding the plaintext. The construction of DARE Envelope permits this to be achieved in a single pass provided that the encryption mode supports single pass processing.

6.2.2. Context String

If the application context specifier is not specified, the context string is the string "DARE-Signature". Otherwise the context string is the concatenation of the string "DARE-Application:" and the application contact specifier.

6.2.3. Example:

```
Signed Header Digest
  C5 13 F0 84  4F 74 3D 9B  50 F8 1B EB  06 41 2D AA
  C6 58 3D D5  F5 83 40 FA  15 D4 69 12  09 EB 2A 13
  93 E2 F8 5B  A8 56 F3 13  60 A8 23 EE  58 68 62 B1
  14 BC 84 C9  58 1D 57 C5  FA 5D 71 D2  9F 9D B6 7E
Payload Digest
  95 94 53 1C  F4 B5 84 AB  AA CD 7C 93  BD 4E E1 00
  0C 86 A5 84  5F E5 F3 DD  99 C1 4B 96  97 F7 1E 22
  8B 1F 77 79  B9 C9 B9 C4  41 4C 05 F4  7D 29 1E 78
  8F 28 C9 ED  88 AF 06 D0  9A 37 83 E7  65 48 35 CB
Signature Manifest
  53 48 41 33  35 31 32 00  C5 13 F0 84  4F 74 3D 9B
  50 F8 1B EB  06 41 2D AA  C6 58 3D D5  F5 83 40 FA
  15 D4 69 12  09 EB 2A 13  93 E2 F8 5B  A8 56 F3 13
  60 A8 23 EE  58 68 62 B1  14 BC 84 C9  58 1D 57 C5
  FA 5D 71 D2  9F 9D B6 7E  C5 13 F0 84  4F 74 3D 9B
  50 F8 1B EB  06 41 2D AA  C6 58 3D D5  F5 83 40 FA
  15 D4 69 12  09 EB 2A 13  93 E2 F8 5B  A8 56 F3 13
  60 A8 23 EE  58 68 62 B1  14 BC 84 C9  58 1D 57 C5
  FA 5D 71 D2  9F 9D B6 7E
ContextString
  44 41 52 45  2D 53 69 67  6E 61 74 75  72 65
Signature Key Id
MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN
Signature Value
  F5 E6 FB 00  E1 46 C9 17  89 96 68 44  CC A1 3E 5F
  F6 DA 93 B8  B5 A6 A2 D0  6B 22 2B 96  FF E5 DA 90
  AD E6 3B 76  2A EB A1 0C  01 D7 D5 1F  35 A5 F0 5E
  C5 E4 CF B5  C8 A2 4C DC  BE 92 3D 18  9D 51 5D 0E
```

6.2.4. Signatures Header

When specified in an Unsigned Header, the Signatures Header contains all the information specified in the signature preamble plus the signature values. When specified in a Trailer, the Signatures Header contains only the signature key identifiers and the corresponding signature values.

Each signature entry consists of the following information:

* The Key Identifier of the Signature Key

* The Digest Algorithm

* The Signature Algorithm

* The Signature Value

Signature Trailer: Envelope

```
{
  "signatures": [{
      "dig": "SHA3512",
      "alg": "ED25519",
      "kid": "MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN",
      "signature": "9eb7AOFGyReJlmhEzKE-X_bak7i1pqLQayIrlv_l2pCt5jt2K
  uuhDAHX1R81pfBexeTPtciiTNy-kj0YnVFdDg"}]}
```

The complete envelope is thus:

Envelope

```
[{
    "signatures": [{
        "dig": "SHA3512",
        "alg": "ED25519",
        "kid": "MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN"}
      ]},
  "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
  "VGhpcyBpcyBhIHRlc3QgZm9yIERhdGEgQXQgUmVzdCBFbnZlbG9wZQ",
  {
    "signatures": [{
        "dig": "SHA3512",
        "alg": "ED25519",
        "kid": "MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN",
        "signature": "9eb7AOFGyReJlmhEzKE-X_bak7i1pqLQayIrlv_l2pC
  t5jt2KuuhDAHX1R81pfBexeTPtciiTNy-kj0YnVFdDg"}
      ]}
  ]
```

When a signed envelope is added to a sequence, the signature properties from the Header and Trailer are combined:

```
   Signature In Sequence: Sequence
   [[{
         "signatures": [{
             "dig": "SHA3512",
             "alg": "ED25519",
             "kid": "MBN5-OA3P-7DRU-FLK3-PTP2-OAUC-PXJN",
             "signature": "9eb7AOFGyReJlmhEzKE-X_bak7i1pqLQayIrlv_l2
     pCt5jt2KuuhDAHX1R81pfBexeTPtciiTNy-kj0YnVFdDg"}
           ]},
       "ewogICJjdHkiOiAidGV4dC9wbGFpbiJ9",
       "VGhpcyBpcyBhIHRlc3QgZm9yIERhdGEgQXQgUmVzdCBFbnZlbG9wZQ",
       null
       ]
     ]
```

7. Security Considerations

7.1. Confidentiality

7.1.1. Salt Reuse

7.1.2. Exchanged Key Construction

7.1.3. Side Channel

7.2. Integrity

7.2.1. Malicious Frame Length Indications

7.2.2. Semantic Substitution Attack

7.3. Availability

7.3.1. Resource Exhaustion

8. IANA Considerations

[Create registry of DARE encoded object types, prepopulate with code points for JOSE envelope and sequence, reserving code points for CBOR, XML and ASN.1]

[Add entries for any new JOSE headers]

[New JOSE Header - witness]

[Assign the Type Indicator registrations]

9. Acknowledgements

The name Data At Rest Encryption was proposed by Melhi Abdulhayo?lu.

10. Appendix A: Cryptographic Keys Used in Examples

Recipient Public Key : Envelope

```
{
  "crv": "X25519",
  "Public": "rlq57sHnbb_f8SoOid_YgYlqLA-F0U2jSeXG5AfNkWI"}
```

Signature Key: Envelope

11. Appendix B: DARE Envelope Examples and Test Vectors

11.1. Plaintext Unsigned

* Encryption Algorithm: none

11.2. Encrypted Unsigned

* Encryption Algorithm: X25519

* Encryption Algorithm: none

11.3. Plaintext Signed

* Encryption Algorithm: none

* Encryption Algorithm: Ed25519

11.4. Encrypted Signed

* Encryption Algorithm: X25519

* Encryption Algorithm: Ed25519

12. Appendix C: DARE Sequence Examples and Test Vectors

13. Appendix D: Outstanding Issues

The following issues need to be addressed.

Issue	Description
Algorithms	Need to audit the algorithm choices
Test Vectors	Produce test vectors for additional algorithms
Extensions	Need work
Security Considerations	Need work
IANA Considerations	Need work
Thumbprints	Use UDFs of thumbprints

Table 4

14. Normative References

[draft-hallambaker-mesh-udf]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.", Work in Progress, Internet-Draft, draft-hallambaker-mesh-udf-19, 14 October 2024, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-udf-19>.

[IANAJOSE] "[Reference Not Found!]".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/rfc/rfc3394>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/rfc/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/rfc7518>.

Author’s Address

Phillip Hallam-Baker
ThresholdSecrets.com
Email: phill@hallambaker.com