Network Working Group
Internet-Draft
Intended status: Informational
Expires: 12 October 2025

P. M. Hallam-Baker
ThresholdSecrets.com

10 April 2025

Encrypted Authenticated Resource Locator
draft-hallambaker-earl-00

Abstract

This document describes the Encrypted Authenticated Resource Locator (EARL) URI scheme and the encoding and decoding of the associated content data. An EARL is a bearer token that allows an encrypted data object to be located, decrypted and authenticated using a compact URI form designed for human readability. A range of work factors is supported from 2^112 to 2^252.

The plaintext data format consists of an initial header section containing metadata describing the payload, the payload itself and an optional section containing signature data.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-mesh-udf.html.

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 12 October 2025.

Copyright Notice

Table of Contents

1. Introduction

This document describes the Encrypted Authenticated Resource Locator (EARL) URI scheme. An EARL is a bearer token that allows an encrypted data object to be located, decrypted and authenticated using a compact URI form designed for human readability.

This document specifies two URI schemes using EARL construction and resolution:

earl: A generic URI scheme for exchange of any type of data.

jscontact: An application specific URI scheme for exchange of contact data encoded in JSContact format.

The processing and resolution schemes for generic and application specific schemes are identical. Use of an application specific form in a QR code allows the URI to indicate which type of application should be used to process the associated data.

Additional application specific schemes MAY be defined in the future.

The following are examples of EARL URIs:

earl://example.com/eluv-woab-g7ih-onix-ybns-qdxk-rzqs
earl:eluv-woab-g7ih-onix-ybns-qdxk-rzqs
jscontact://example.com/eluv-woab-g7ih-onix-ybns-qdxk-rzqs

All three forms refer to the exact same data object. The first two forms are URLs that indicate that the corresponding ciphertext MAY be retreived via HTTPS at:

https://example.com/.well-known/earl/-utAO8IYsdcqmVGk2W15PCLDAFT1HL7MfWCWQ-s9qYU

1.1. Construction

Construction of an EARL for and publication of the associated ciphertext is a four-step process:

Envelope The enveloped plaintext data consists of the payload combined with optional metadata and signature data sections.

Derive multi-purpose key The multi-purpose key is calculated over the entire enveloped plaintext data and the result presented as a Base32-dash encoded string.

The EARL is a URI with the multi-purpose key as the path section.

Encryption The enveloped plaintext data is encrypted under a key derived from the multi-purpose key, the result being the ciphertext data.

Publish The ciphertext data is published as an HTTPS well-known service with a path formed from the earl well-known service prefix followed by a locator derived by the result of passing the content digest string through a digest function twice.

This construction establishes the EARL as a ’bearer token’ that may be used to locate, decrypt and authenticate the associated payload and metadata.

Since the recovered payload is authenticated by means of the multi-purpose key, the source from which the ciphertext is obtained is immaterial. Ciphertexts MAY be cached for later retrieval. The double digest construction of the locator allows the result of the first pass to be used to authenticate retrieval requests by a party that does not have the ability to decrypt.

1.2. Resolution

Resolution of an EARL follows the same pattern as construction except that the ciphertext data is decrypted to recover the enveloped data and the resolver MUST verify that the computed value of the digest value is consistent with the value of the key.

1.3. Applications

The EARL scheme is designed to support a wide range of applications including:

A laboratory provides pathology results to a doctor in paper form which are in turn passed to a consultant who requires the results in electronic form for further analysis. Attaching a QR code containing an EARL allows the consultant to obtain the electronic form from the printed document without compromise to patient confidentiality. The paper document is a bearer token that can be exchanged for a paper form of itself

A device carries a QR code containing an EARL linking to a description of the device for use in onboarding.

A protocol requires that a trust anchor be passed as a compact URI without reliance on a Trusted Third Party. This is of particular concern when Post Quantum Cryptographic algorithms requiring very large public keys are involved.

1.4. JSContact

One important application of EARLs is to support the exchange of JSContact documents so that the trust context in which the contact is exchanged is preserved. Thus, ensuring that any cryptographic keys or trust anchors have the benefit of that context.

For example, Alice might print a QR code containing an EARL linking to her JSContact data on her business card. When she hands the card to Bob, he can scan the card to obtain Alice's OpenPGP, SSH, S/MIME etc. credentials.

Use of the jscontact URI scheme for this application allows a camera application reading the QR code to hand off processing of the URI to a contacts application registered as the handler for the jscontact scheme.

Alternatively, if Alice and Bob are unable to meet in person, Alice might publish the jscontact URI as a prefixed TXT entry in her personal DNS Handle @alice.example.com. If Alice's DNS domain is secured by means of DNSSEC, Bob has a degree of third-party attestation to the binding of the contact data to the domain.

In either case, the JSContact data MAY contain information specifying how updates MAY be received and validated by means of a digital signature contained within the enveloped plaintext data. Once established, the trust relationship is maintained end-to-end between Alice and Bob without the need to rely on any third party.

2. Definitions

This section presents the related specifications and standards, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. 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].

2.2. Defined Terms

Cryptographic Digest Function A hash function that has the properties required for use as a cryptographic hash function. These include collision resistance, first pre-image resistance and second pre-image resistance.

Media Type An identifier indicating how a Data Value is to be interpreted as specified in the IANA registry Media Types.

Data Value The binary octet stream that is the input to the digest function used to calculate a digest value.

Data Object A Data Value and its associated Content Type

Digest Algorithm A synonym for Cryptographic Digest Function

Digest Value The output of a Cryptographic Digest Function

Data Digest Value The output of a Cryptographic Digest Function for a given Data Value input.

Work Factor A measure of computational effort required to perform an attack against some security property.

2.3. Related Specifications

This specification makes use of Base32 [RFC4648] encoding, the SHA-3 [SHA-3] digest function and AES-GCM encryption mode [RFC5288].

The URI scheme follows the approach described in [RFC3986].

The resource location scheme makes use of the Well-Known Resource scheme described in [RFC8615].

This work is based on the work originally presented in the Mathematical Mesh UDF [draft-hallambaker-mesh-udf] and DARE Envelope [draft-hallambaker-mesh-dare] schemes.

2.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

3. Architecture

3.1. URI Syntax

Two URI forms are defined: Name Form and Locator Form.

In each case the scheme name MUST be specified. This MAY be the base scheme name ’earl’ or a separately registered sub-scheme name used to specify a particular disposition for the referenced content.

For example, a QR code containing a contact record might specify a scheme name indicating that the linked data is contact data so that the content can be passed to the user’s contacts application.

3.1.1. Name Form

The name form of the EARL URI consists of the scheme name (e.g. earl) and a path value constructed from the Base32-D encoding of the key. For example:<include=..\Examples\ EarlExamplesName.md>

The name form of the EARL allows a single URI to be used to decrypt the associated ciphertext and authenticate the resulting payload and metadata. It does not specify the means for retrieving the ciphertext.

3.1.2. Locator Form

The locator form of the EARL URI consists of the scheme name (e.g. earl), an authority section specifying a host name and a path value constructed from the Base32-D encoding of the key. For example:<include=..\Examples\ EarlExamplesLocator.md>

The locator form of the EARL allows a single URI to be used to retrieve the associated ciphertext, decrypt it and authenticate the resulting payload and metadata.

3.1.3. Algorithm Suites

Currently, only one algorithm suite is defined: SHA3-AES-GCM

Type Identifier: 34

Content Digest: SHAKE-256

Key Derivation Function: SHAKE-256

Payload Encryption Algorithm AES-GCM 256 bit with 96 bit nonce

Locator Digest Function: SHA-3-512

The first byte of the binary key is always 34. Thus, the Base32 encoded key will always begin with the letter E, providing an easy means of distinguishing EARL identifier from other UDF identifiers [draft-hallambaker-mesh-udf].

3.2. Enveloped Data

The enveloped data format uses the varint scheme specified in QUIC [RFC9000] to establish a simple and space efficient envelope with similar capabilities to that of Cryptographic Message Syntax [RFC5652]. Unlike the JWS format [RFC7515], neither the payload nor the associated metadata is Base64 encoded.

The envelope scheme is based on the one presented in Data At Rest Encryption (DARE) [draft-hallambaker-mesh-dare] but using QUIC varints for framing in place of JSON-B tags, and does not specify a means of encrypting the envelope payload or constructing sequences of envelopes.

3.2.1. Type 0

The Type 0 envelope supports a fixed length payload with optional metadata.

For example, the payload:

"This is a test" =

```
  54 68 69 73  20 69 73 20  61 20 74 65  73 74
```

Is encoded as a Type 0 envelope without metadata by concatenating

* The envelope type identifier (0),

* a zero length byte to indicate an empty metadata field,

* a varint specifying the number of payload bytes (14) followed by the payload bytes.

```
  00 00 0E 54  68 69 73 20  69 73 20 61  20 74 65 73
  74
```

The same payload as before MAY be enveloped with the following metadata:

```
{
  "Nonce": "NBIH-H7MK-AI4M-ZWCV-RSCC-E57K-UPCA",
  "cty": "text/plain"}
```

The enveloped content is as before except that the metadata field is now specified:

```
00 40 49 7B  0A 20 20 22  4E 6F 6E 63  65 22 3A 20
22 4E 42 49  48 2D 48 37  4D 4B 2D 41  49 34 4D 2D
5A 57 43 56  2D 52 53 43  43 2D 45 35  37 4B 2D 55
50 43 41 22  2C 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
```

3.2.2. Type 1

The Type 1 envelope additionally supports signatures and encoding of payloads of unknown length without buffering.

"First Chunk"
"Second Chunk"

Is encoded as a Type 1 envelope by concatenating:

* The envelope type identifier (1),
* a zero length byte to indicate an empty unprotected header field,
* a varint specifying the number of metadata bytes followed by the metadata,
* a varint specifying the number of bytes in the first chunk followed by the first chunk,
* a varint specifying the number of bytes in the second chunk followed by the second chunk,
* a zero length byte to indicate the end of the payload section,
* a zero length byte to indicate an empty trailer field.

```
01 00 40 49  7B 0A 20 20  22 4E 6F 6E  63 65 22 3A
20 22 4E 42  49 48 2D 48  37 4D 4B 2D  41 49 34 4D
2D 5A 57 43  56 2D 52 53  43 43 2D 45  35 37 4B 2D
55 50 43 41  22 2C 0A 20  20 22 63 74  79 22 3A 20
22 74 65 78  74 2F 70 6C  61 69 6E 22  7D 0B 46 69
72 73 74 20  43 68 75 6E  6B 0C 53 65  63 6F 6E 64
20 43 68 75  6E 6B 00 00
```

Since the multi-purpose key is calculated over the entire envelope, changing the envelope type or the division of the payload into chunks will change the value of the multi-purpose key. It is therefore necessary to preserve the original ciphertext package or maintain the encoding of the recovered plaintext if ’round trip’ processing is to be attempted.

Changing the division of the payload into chunks does not change the value of the signature however since this is computed over a manifest constructed from the digest algorithm identifier, metadata digest and payload digest.

3.2.3. Word alignment

In some circumstances, it is desirable to align the payload portion of an envelope on a 4-byte or 8-byte word. This MAY be achieved by padding the metadata section of the envelope with the necessary quantity of white space.

3.3. Multipurpose Key Derivation

The enveloped plaintext data is processed with the Content Digest. The first byte is replaced by the Type identifier, and the result is presented as a Base32-dash string with enough 4-character segments to reach the desired work factor.

The Base32-dash encoded form is only used to create the path segment of the EARL URI. The locator and encryption key are derived from the binary form of the multi-purpose key obtained by decoding the key.

The content digest of the envelope shown in the first example is computed using SHAK256:

```
81 E9 5B 38  01 37 D0 77  35 17 C0 5B  28 0E EA 8E
61 23 F5 B5  D1 B1 54 91  BC 88 C4 18  29 71 AD 07
```

The first byte of the result is set to the type identifier 34, truncated to the desired precision (140 bits):

```
22 E9 5B 38  01 37 D0 77  35 17 C0 5B  28 0E EA 8E
61 23
```

The multipurpose key presentation is the result presented in Base32 with the characters grouped into sets of four with dashes:

```
eluv-woab-g7ih-onix-ybns-qdxk-rzqs
```

Note that since there is an odd number of 20-bit segments in this example, only the upper half of the last byte is recorded in the key:

```
22 E9 5B 38  01 37 D0 77  35 17 C0 5B  28 0E EA 8E
61 20
```

The EARL forms consist of the scheme name, authority section (if specified) and the presentation form of the multi-purpose key:

earl://example.com/eluv-woab-g7ih-onix-ybns-qdxk-rzqs
earl:eluv-woab-g7ih-onix-ybns-qdxk-rzqs
jscontact://example.com/eluv-woab-g7ih-onix-ybns-qdxk-rzqs

3.3.1. Nonces

If a low entropy payload is encoded, an attacker might be able to recover the payload content through a brute force attack. To prevent this attack, the metadata segment SHOULD include a nonce property containing a string containing a sufficient degree of unguessable information.

If a different nonce is specified in the metadata:

```
{
  "Nonce": "NCZ2-C6BE-IFM3-GHDN-LFDD-XZZ4-76KP",
  "cty": "text/plain"}
```

A different multi-purpose key is derived:

earl://example.com/eknc-x6cs-de3a-25vb-73si-2k6x-ni4a

3.4. Encryption

The enveloped payload is encrypted under a key and nonce derived from the multi-purpose key using the key derivation function.

Since the encryption algorithm is AES-GCM with a 96 nonce and a 256 bit key, it is necessary to generate 44 bytes to encrypt the enveloped data.

The key derrivation function, SHAKE-256 is applied to the multi-purpose key of the first example to obtain 44 bytes of output:

```
  3C BA 88 A8  5B AC 99 E1  E5 D2 A3 28  40 9E F2 67
  A9 D3 C2 0B  24 04 7B B0  C4 30 63 EA  23 45 B4 F2
  DF D7 06 31  50 4D 2D EF  E8 82 79 93
```

The first 32 bytes of the result are the AES-GCM encryption key:

encryption key =

```
  3C BA 88 A8  5B AC 99 E1  E5 D2 A3 28  40 9E F2 67
  A9 D3 C2 0B  24 04 7B B0  C4 30 63 EA  23 45 B4 F2
```

The final 12 bytes of the result are the AES-GCM nonce:

encryption nonce =

```
DF D7 06 31  50 4D 2D EF  E8 82 79 93
```

The enveloped plaintext data is encrypted under AES-GCM with the specified key and nonce to produce the ciphertext:

ciphertext =

```
BC 46 D1 67  A2 08 D2 4B  2D F7 27 18  3B 1B 27 EF
88 23 EF E3  68 8C EA 53  A8 FB 0C CB  72 EB D3 11
74
```

3.5. Publication

The ciphertext MAY be published through the HTTPS well-known service earl.

The host for the HTTP service is specified by the authority section of the EARL URI. The final path section is the base64url encoding of the result of passing the multi-purpose key through the locator digest function twice:

The first locator value is computed from the multi-purpose key using the locator digest function, in this case SHA-3-256:

locator1 =

```
59 34 18 96  A1 60 AD 9C  13 DF A4 33  8A 2A 72 DD
E7 EB DD 3F  6A DC 8D 0D  18 39 EB 51  2F 22 8C 03
```

The locator value is computed from the first locator value using the locator digest function, in this case SHA-3-256:

locator =

```
59 34 18 96  A1 60 AD 9C  13 DF A4 33  8A 2A 72 DD
E7 EB DD 3F  6A DC 8D 0D  18 39 EB 51  2F 22 8C 03
```

The locator URI is a HTTPS well-known service in which the final element of the path is the base64url encoded locator value.

EARL =
https://example.com/.well-known/earl/-utAO8IYsdcqmVGk2W15PCLDAFT1HL7M
    fWCWQ-s9qYU

Note that publication using HTTPS is only one possible method of retrieval. In certain circumstances

3.5.1. Access Authenticator

The first locator value MAY be used to authenticate access to the ciphertext. The first locator value is Base32 encoded without dashes to derive an access authenticator that can be used for password type authentication.

The same string is used as the username if the authentication mechanism requires one to be specified.

access authenticator =
LE2BRFVBMCWZYE67UQZYUKTS3XT6XXJ7NLOI2DIYHHVVCLZCRQBQ

Use of the first locator value is preferred over the multi-purpose key because the first locator value cannot be used to decrypt the ciphertext. Thus a repository of ciphertext values can be provided with the access authenticator to implement access control without granting the ability to decrypt.

3.5.2. Additional Derived Properties

Use of the multi-purpose key is not limited to the applications specified in this document. The multi-purpose key MAY be used to derive any additional information that is needed in an application.

For example, a wireless device using a QR encoded EARL to provide a device description to be used for onboarding might support use of the multi-purpose key to derive a wireless network identifier and access credentials to enable initial access to the device.

The means by which these parameters are derived is outside the scope of this document but any such applications MUST ensure that the mechanism employed does not disclose the multi-purpose key or access authenticator values.

3.6. Recovery

If the ciphertext is published as a HTTPS well-known service, recovery of the ciphertext is achieved by performing a HTTPS GET method on the specified host and path.

3.6.1. Decryption and Authentication

To recover the enveloped plaintext data from the ciphertext, the encryption key and nonce are derived from the EARL from the multi-purpose key in the same fashion as before and the content digest of the result verified against the original multi-purpose key.

Applications MUST authenticate the recovered plaintext against the multi-purpose key. This step is necessary even in the case that an authenticated encryption scheme such as AES GCM is used as anyone with knowledge of the multi-purpose key can create a ciphertext with a valid GCM tag.

3.7. Signed Envelopes

The payload and metadata MAY be signed with one or more signatures.

The signature value is computed as a JOSE signature over a manifest consisting of an ASN.1 OID identifying the digest algorithm, the digest of the metadata and the digest of the payload. The values of the unprotected header and payload are not part of the signature scope, this allows additional signatures to be added to an envelope without changing the validity of the existing signatures.

When calculating the payload digest, only the payload bytes are included, the length bytes of the payload chunks are not. Thus, an envelope in which the payload is encoded in multiple chunks can be re-encoded in a single chunk without changing the validity of the the signature.

Signature values MAY be specified in either the unprotected header or the trailer but not both.

If the signature values are specified in the trailer, the unprotected header MUST contain a Digest property specify the digest used to compute the metadata and payload digest and a Signers property identifying the algorithms and signature keys.

3.7.1. Unprotected Header

If the envelope is signed, the unprotected header MUST contain a digest property and either a Signers property or a Signatures property but not both.

```
Unprotected Header =
{
  "dig": "SHA3256",
  "signs": [{
      "alg": "ED25519",
      "kid": "MAHF-HMNL-GYII-ANCK-P2ZY-GXMN-GUIC"}]}
```

It is not necessary for the header to fully specify the public signature key(s).

The digest algorithm SHOULD provide at least as many output bits as the signature algorithm secures.

3.7.2. Manifest

The manifest consists of an ASN.1 OID specifying the digest algorithm, the metadata digest and the payload digest in that order. If the Signature value specified a Protected property, the binary protected data is appended.

Since the length of an ASN.1 digest algorithm OID is self-describing and determines the lengths of the digest values, it is not necessary for the manifest contain any length specifiers.

Metadata digest =

```
C2 21 EC 1C  5B 98 B7 AD  35 D4 1A 10  5C 3E 66 86
8C 73 6F 1B  7C AB AF C6  9C E7 A2 A1  DB 6C 24 D5
```

Payload digest =

```
BC 83 A2 BD  6C 81 D1 89  A5 C3 58 E2  2A 39 8D 07
FC 43 CE 99  74 78 FA 27  FC AC 9B 5B  55 04 AC 86
```

Manifest =

```
60 48 86 01  65 03 04 02  08 C2 21 EC  1C 5B 98 B7
AD 35 D4 1A  10 5C 3E 66  86 8C 73 6F  1B 7C AB AF
C6 9C E7 A2  A1 DB 6C 24  D5 BC 83 A2  BD 6C 81 D1
89 A5 C3 58  E2 2A 39 8D  07 FC 43 CE  99 74 78 FA
27 FC AC 9B  5B 55 04 AC  86
```

3.7.3. Signatures

The signature values are calculated over the binary form of the manifest.

Ed25519 private key =

```
3C AA 32 31  AE FC 0C EA  7D BE 1F CD  67 EC 6F 8A
D3 FA A9 8C  48 60 80 4B  27 44 6D D0  46 54 78 61
```

The trailer contains a Signatures property with a single entry specifying the key identifier of the signing key and the signature value:

```
Trailer =
{
  "sigs": [{
      "kid": "MAHF-HMNL-GYII-ANCK-P2ZY-GXMN-GUIC",
      "val": "ewoF3SnajtDoFuPY2j5QDg58r54ePnZeMnrYm0QAwqNOVPw-YgAnGnA
  ceDjUzb9NHVG-q29vKEprdPz9f1G_Dg"}]}
```

The process for signing with an Ed448 key is identical except that achieving the higher work factor requires use of a 512 bit digest:

```
Unprotected Header =
{
  "dig": "SHA3512",
  "signs": [{
      "alg": "ED448",
      "kid": "MCPK-6QF7-NABM-CRK5-D2OA-6JSP-ST2R"}]}
```

The private key used to generate the signature in this example is:

Ed448 private key =

```
  3A DB E3 59  3B 4B D7 02  F7 E7 E5 96  A0 66 71 12
  9E 50 B8 BF  C5 F1 26 04  A2 1D 7D 5F  E6 81 4E 36
  FA BB D9 8C  93 84 88 F3  E8 87 90 D2  A1 B7 4F A6
  9E 4E BE 40  3A 70 F2 09
```

The signature value is specified in the trailer as before:

```
Trailer =
{
  "sigs": [{
      "kid": "MCPK-6QF7-NABM-CRK5-D2OA-6JSP-ST2R",
      "val": "KFrCppvBXHDUwkb1UJof64dN7DTnLLUVqyBIejJVJxGDiSUJlrTRh5Q
  tASC4gEVIYEuXWzNJKEcAH3n8oDBKhCW9Jt3iqJQ-ksKai-jOdwLr1a1XgYqD5
  zoXjzBV8m1WS1e2kB-8ycRH8hF7UmCG_SsA"}]}
```

4. Envelope Format

Two envelope formats are defined:

Type 0: Known content length

Type 1: Indeterminate content length.

The known content length encoding is used in cases where the length of the content is known at the start of the envelope creation process, the type 1 encoding is used for all other cases.

4.1. Variable-Length Integer Encoding

The Variable-Length Integer Encoding (varint) scheme describe in [RFC9000] section 16 is used to specify the section length, presented here for the convenience of the reader:

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. Table 4 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 1

Values do not need to be encoded in the minimum number of bytes necessary.

4.2. Type Identifier

The type identifier consists of a varint specifying the envelope type and version. Applications MUST reject envelopes with an unknown type identifier. Future identifiers MAY be entirely incompatible, for example, specifying metadata as HTTP header fields or using XML syntax.

4.3. Envelope Type 0

The type 0 encoding has exactly three sections in the following order:

Type Identifier (0)

Metadata Header

Payload

Use of the encoding incurs an encoding overhead of between 3 and 17 bytes depending on the length of the data segments.

The chief limitation of the type 0 encoding is that it does not support use of signatures and the length of the payload must be known in advance.

4.3.1. Metadata Header

The Metadata section begins with a varint length specifier followed by the specified number of data bytes.

If the varint length specifier is zero, the metadata section is empty. Otherwise, the data bytes MUST contain a JSON object containing metadata parameters.

4.3.2. Payload

The payload section begins with a varint length specifier followed by the specified number of data bytes.

The payload content is interpreted according to the directions in the specified metadata.

4.4. Envelope Type 1

The type 1 encoding is designed to allow a stream of data to be enveloped and signed without buffering the payload using a chunked payload encoding. It contains an additional unprotected header and trailer section. in the following order:

Type Identifier (1)

Unprotected Header

Metadata Header

Payload

Trailer

The type 1 encoding incurs a typical encoding overhead of 5 bytes plus 1-2 bytes per payload chunk.

4.4.1. Unprotected Header

The Unprotected Header section begins with a varint length specifier followed by the specified number of data bytes.

If the varint length specifier is zero, the metadata section is empty. Otherwise, the data bytes MUST contain a JSON object which MAY contain signature parameters.

4.4.2. Metadata Header

The Metadata section begins with a varint length specifier followed by the specified number of data bytes.

If the varint length specifier is zero, the metadata section is empty. Otherwise, the data bytes MUST contain a JSON object containing metadata parameters.

4.4.3. Payload

The Payload section consists of a series of zero or more data chunks with a non-zero varint length specifier followed by the specified number of data bytes followed by a varint encoding of zero.

The payload value is the result of concatenating the payload data bytes in the order they are presented.

4.4.4. Trailer

The Trailer section begins with a varint length specifier followed by the specified number of data bytes.

If the varint length specifier is zero, the metadata section is empty. Otherwise, the data bytes MUST contain a JSON object which MAY contain signature parameters.

5. JSON Header Parameters

The Header and Trailer sections MAY contain any parameter specified in the IANA JSON Web Signature and Encryption Header Parameters registry with the meanings specified therein.

5.1. Signature Parameters

If the envelope is signed, the unprotected header MUST be present and MUST include either a Signer parameter or a Signatures parameter but not both.

If the unprotected header contains a Signer parameter, the trailer MUST be present and MUST contain a Signatures header.

5.1.1. Signers signs

An array of Signer objects. The presence of a Signer parameter notifies applications processing the envelope that the payload content is signed under the specified signature keys.

5.1.2. Signatures sigs

An array of Signature objects.

When specified in a trailer section, the Signatures parameter provides the signature values corresponding to the Signers specified in the unprotected header.

When specified in an unprotected header, the Signatures parameter provides all the signature parameters, including the signature value.

5.2. Metadata Parameters

The metadata parameters MAY be used to described the enveloped content. The interpretation of these parameters is the same as that specified by JOSE and the HTTP specification.

5.2.1. ContentType cty

The "cty" (content type) Header Parameter is used to declare the media type [IANA.MediaTypes] of the payload as describe in Section 4.1.10. [RFC7516]

5.2.2. Compression zip

Specifies a compression algorithm applied to the content to derive the payload as specified in [RFC7516] Section 4.1.3.

5.2.3. Language lang

The natural language(s) of the intended audience for the payload. The property contains an array of strings, each containing an language tag as specified in [RFC5645]. Use of this property is intended to be equivalent to the HTTP Content-Language header specified in [RFC9110] Section 8.5.

5.2.4. Filename file

Suggested filename for storing the payload content.

5.2.5. Digest dig

The default digest algorithm used to process the payload and protected headers to create signatures.

5.2.6. Nonce nonce

Optional unique value incorporated into the metadata.

The value of the "nonce" header parameter MUST be an octet string, encoded according to the base64url encoding as required by [RFC8555] Section 6.5.2.

Incorporation of a randomly chosen nonce value in the content metadata ensures the uniqueness of the envelope digest value and preventing disclosure of low entropy content data by brute force attack.

5.3. Signer

The signer object specifies an algorithm and key used to sign the payload and metadata.

5.3.1. Algorithm alg

The JOSE signature algorithm as specified in [RFC7515] Section 4.1.1

5.3.2. Digest dig

The digest algorithm used to process the payload and protected headers to create this signature if different to the default.

5.3.3. KeyId kid

The JOSE key identifier as specified in [RFC7515]

If separate Signatures and Signers parameters are specified, the kid fields of each are used to match signatures to signers. Therefore, each kid value MUST be unique within the scope of the envelope and there MUST be exactly one signature with a kid matching the value of each signer kid.

5.3.4. JWK Set Url jku

The JOSE Web Key Set as specified in [RFC7515] Section 4.1.2

5.3.5. JSON Web Key jwk

The JOSE key specifier as specified in [RFC7515] Section 4.1.3

5.4. Signature

The signature object specifies a signature value. When specified in a trailer, the signature object need only specify the Key Identifier and Value properties. When specified in an unprotected header, the signature object contains the full specification of the signature parameters.

Any Signer field plus:

5.4.1. Protected prot

A base64url string whose binary value is a JSON document containing additional headers to be incorporated into the signature manifest.

5.4.2. Value val

The signature value.

6. Security Considerations

6.1. Confidentiality

Encrypted locator is a bearer token

6.2. Availability

6.3. Integrity

6.4. Semantic Substitution

Many applications record the fact that a data item is trusted, rather fewer record the circumstances in which the data item is trusted. This results in a semantic substitution vulnerability which an attacker may exploit by presenting the trusted data item in the wrong context.

6.5. QR Code Scanning

The act of scanning a QR code SHOULD be considered equivalent to clicking on an unlabeled hypertext link. Since QR codes are scanned in many different contexts, the mere act of scanning a QR code MUST NOT be interpreted as constituting an affirmative acceptance of terms or conditions or as creating an electronic signature.

If such semantics are required in the context of an application, these MUST be established by secondary user actions made subsequent to the scanning of the QR code.

There is a risk that use of QR codes to automate processes such as payment will lead to abusive practices such as presentation of fraudulent invoices for goods not ordered or delivered. It is therefore important to ensure that such requests are subject to adequate accountability controls.

7. IANA Considerations

Registrations are requested in the following registries:

* well-known URI registry

* Uniform Resource Identifier (URI) Schemes

In addition, the creation of the following registry is requested: Uniform Data Fingerprint Type Identifier Registry.

7.1. Well Known

The following registration is requested in the well-known URI registry in accordance with [RFC5785]

URI suffix earl

Change controller Phillip Hallam-Baker, phill@hallambaker.com

Specification document(s): [This document]

Related information

7.2. URI Registration

The following registration is requested in the Uniform Resource Identifier (URI) Schemes registry in accordance with [RFC7595]

Scheme name: earl

Status: Provisional

Applications/protocols that use this scheme name: TBS

Contact: Phillip Hallam-Baker mailto:phill@hallambaker.com

Change controller: Phillip Hallam-Baker

References: [This document]

7.3. URI Registration

The following registration is requested in the Uniform Resource Identifier (URI) Schemes registry in accordance with [RFC7595]

Scheme name: jscontact

Status: Provisional

Applications/protocols that use this scheme name: TBS

Contact: Phillip Hallam-Baker mailto:phill@hallambaker.com

Change controller: Phillip Hallam-Baker

References: [This document]

7.4. Uniform Data Fingerprint Type Identifier Registry

This document describes a new extensible data format employing fixed length version identifiers for UDF types.

7.4.1. The name of the registry

Uniform Data Fingerprint Type Identifier Registry

7.4.2. Required information for registrations

Registrants must specify the Type identifier code(s) requested, description and RFC number for the corresponding standards action document.

The standards document must specify the means of generating and interpreting the UDF Data Sequence Value and the purpose(s) for which it is proposed.

Since the initial letter of the Base32 presentation provides a mnemonic function in UDFs, the standards document must explain why the proposed Type Identifier and associated initial letter are appropriate. In cases where a new initial letter is to be created, there must be an explanation of why this is appropriate. If an existing initial letter is to be created, there must be an explanation of why this is appropriate and/or acceptable.

7.4.3. Applicable registration policy

Due to the intended field of use (human data entry), the code space is severely constrained. Accordingly, it is intended that code point registrations be as infrequent as possible.

Registration of new digest algorithms is strongly discouraged and should not occur unless, (1) there is a known security vulnerability in one of the two schemes specified in the original assignment and (2) the proposed algorithm has been subjected to rigorous peer review, preferably in the form of an open, international competition and (3) the proposed algorithm has been adopted as a preferred algorithm for use in IETF protocols.

Accordingly, the applicable registration policy is Standards Action.

7.4.4. Size, format, and syntax of registry entries

Each registry entry consists of an integer code.

7.4.5. Initial assignments and reservations

The following entries should be added to the registry as initial assignments:

Code	Description	Reference
34	EARL-AES-SHA3	[This document]
104	Nonce	[This document]

Table 2

8. Acknowledgements

9. Appendix A: Base32-D

Base32 in lower case with dashes between blocks of 4 characters, no padding.

The output is always an integer multiple of 20 bits.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	a	9	j	18	s	27	3
1	b	10	k	19	t	28	4
2	c	11	l	20	u	29	5
3	d	12	m	21	v	30	6
4	e	13	n	22	w	31	7
5	f	14	o	23	x		
6	g	15	p	24	y		
7	h	16	q	25	z		
8	i	17	r	26	2		

10. Normative References

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <https://www.rfc-editor.org/rfc/rfc5288>.

[RFC5645] "[Reference Not Found!]".

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/rfc/rfc5652>.

[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] "[Reference Not Found!]".

[RFC8555] "[Reference Not Found!]".

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC9000] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9110] "[Reference Not Found!]".

[SHA-3] Dworkin, M. J., "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", August 2015.

11. Informative References

[draft-hallambaker-mesh-dare]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part III : Data At Rest Encryption (DARE)", Work in Progress, Internet-Draft, draft-hallambaker-mesh-dare-18, 14 October 2024, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-dare-18>.

[draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-12, 14 October 2024, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-12>.

[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>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/rfc/rfc5785>.

[RFC7595] Thaler, D., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/rfc/rfc7595>.

Author’s Address

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