Workgroup: Network Working Group
Internet-Draft: draft-hallambaker-mesh-udf
Published: 20 April 2022
Intended Status: Informational
Expires: 22 October 2022
Authors: P. M. Hallam-Baker
ThresholdSecrets.com

Mathematical Mesh 3.0 Part II: Uniform Data Fingerprint.

Abstract

This document describes the underlying naming and addressing schemes used in the Mathematical Mesh. The means of generating Uniform Data Fingerprint (UDF) values and their presentation as text sequences and as URIs are described.

A UDF consists of a binary sequence, the initial eight bits of which specify a type identifier code. For convenience, UDFs are typically presented to the user in the form of a Base32 encoded string. Type identifier codes have been selected so as to provide a useful mnemonic indicating their purpose when presented in Base32 encoding.

Two categories of UDF are described. Data UDFs provide a compact presentation of a fixed length binary data value in a format that is convenient for data entry. A Data UDF may represent a cryptographic key, a nonce value or a share of a secret. Fingerprint UDFs provide a compact presentation of a Message Digest or Message Authentication Code value.

A Strong Internet Name (SIN) consists of a DNS name which contains at least one label that is a UDF fingerprint of a policy document controlling interpretation of the name. SINs allow a direct trust model to be applied to achieve end-to-end security in existing Internet applications without the need for trusted third parties.

UDFs may be presented as URIs to form either names or locators for use with the UDF location service. An Encrypted Authenticated Resource Locator (EARL) is a UDF locator URI presenting a service from which an encrypted resource may be obtained and a symmetric key that may be used to decrypt the content. EARLs may be presented on paper correspondence as a QR code to securely provide a machine-readable version of the same content. This may be applied to automate processes such as invoicing or to provide accessibility services for the partially sighted.

[Note to Readers]

Discussion of this draft takes place on the MATHMESH mailing list (mathmesh@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=mathmesh.

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 22 October 2022.

Copyright Notice

Table of Contents

1. Introduction

A Uniform Data Fingerprint (UDF) is a generalized format for presenting and interpreting short binary sequences representing cryptographic keys or fingerprints of data of any specified type. The UDF format provides a superset of the OpenPGP [RFC4880] fingerprint encoding capability with greater encoding density and readability.

This document describes the syntax and encoding of UDFs, the means of constructing and comparing them and their use in other Internet addressing schemes.

1.1. UDF Types

Two categories of UDF are described. Data UDFs provide a compact presentation of a fixed length binary data value in a format that is convenient for data entry. A Data UDF may represent a cryptographic

key or nonce value or a part share of a key generated using a secret sharing mechanism. Fingerprint UDFs provide a compact presentation of a Message Digest or Message Authentication Code value.

Both categories of UDF are encoded as a UDF binary sequence, the first octet of which is a Type Identifier and the remaining octets specify the binary value according to the type identifier and data referenced.

UDFs are typically presented to the user as a Base32 encoded sequence in groups of four characters separated by dashes. This format provides a useful balance between compactness and readability. The type identifier codes have been selected so as to provide a useful mnemonic when presented in Base32 encoding.

The following are examples of UDF values:

```
Nonce:      ND5L-BTJE-W372-4TUS-E23N-FIXU-6ARA
Secret:     UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ
SHA-2 Digest:MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
SHA-3 Digest:KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN
Public Key: OAYC-4MAH-AYBS-WZLQ-AUAA-GIYA-AQQK-XW5Y-YO6L-TIJU-43NU-2
        2J5-VP23-IJ7A-4JCH-LBCC-LFAZ-6QHO-GLUY-FEQ
```

Like email addresses, UDFs are not a Uniform Resource Identifier (URI) but may be expressed in URI form by adding the scheme identifier (UDF) for use in contexts where an identifier in URI syntax is required. A UDF URI **MAY** contain a domain name component allowing it to be used as a locator

1.1.1. Cryptographic Keys and Nonces

A Nonce (N) UDF represents a short, fixed length randomly chosen binary value.

Nonce UDFs are used within many Mesh protocols and data formats where it is necessary to represent a nonce value in text form.

```
Nonce UDF:
  ND5L-BTJE-W372-4TUS-E23N-FIXU-6ARA
```

An Encryption/Authentication (E) UDF has the same format as a Random UDF but is identified as being intended to be used as a symmetric key for encryption and/or authentication.

```
KeyValue:
  DE BB 13 26  28 7A 83 5B  58 65 38 B1  48 8C 7C

Encryption/Authenticator UDF:
  UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ
```

A Share (S) UDF also represents a short, fixed length binary value but only provides one share in secret sharing scheme. Recovery of the binary value requires a sufficient number of shares.

Share UDFs are used in the Mesh to support key and data escrow operations without the need to rely on trusted hardware. A share UDF can be copied by hand or printed in human or machine-readable form (e.g. QR code).

```
Key:     UXPL-WEZG-FB5I-GW2Y-MU4L-CSEM-PQ
Share 0: SAQF-K5LV-2HDO-PSY5-YQBN-CPMS-7MXM-K
Share 1: SAQQ-KDBQ-SBT2-OG5Y-FSWT-2QTU-VXIQ-4
Share 2: SARL-JIXL-J4EG-M3CS-SVL2-SR2W-MBZY-U
```

1.1.2. Fingerprint type UDFS

Fingerprint type UDFs contains a fingerprint value calculated over a content data item and an IANA media type.

A Content Digest type UDF is a fingerprint type UDF in which the fingerprint is formed using a cryptographic algorithm. Two digest algorithms are currently supported, SHA-2-512 (M, for Merkle Damgard) and SHA-3-512 (K, for Keccak).

The inclusion of the media type in the calculation of the UDF value provides protection against semantic substitution attacks in which content that has been found to be trustworthy when interpreted as one content type is presented in a context in which it is interpreted as a different content type in which it is unsafe.

```
SHA-2-512: MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4
SHA-3-512: KCM5-7VB6-IJXJ-WKHX-NZQF-OKGZ-EWVN
```

An Authentication UDF (A) is formed in the same manner as a fingerprint but using a Message Authentication Code algorithm and a symmetric key.

Authentication UDFs are used to express commitments and to provide a means of blinding fingerprint values within a protocol by means of a nonce.

SHA-2-512: ABIR-Y75D-WV63-P5WC-KR3F-VA57-YZ6I

1.2. Using UDFs in URIs

The UDF URI scheme allows use of a UDF in contexts where a URF is expected. The UDF URI scheme has two forms, name and locator.

1.2.1. Name Form

Name form UDF URIs identify a data resource but do not provide a means of discovery. The URI is simply the scheme (udf) followed by the UDF value:

udf:MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4

1.2.2. Locator Form

Locator form UDF URIs identify a data resource and provide a hint that **MAY** provide a means of discovery. If the content is not available from the location indicated, content obtained from a different source that matches the fingerprint **MAY** be used instead.

udf://example.com/MB5S-R4AJ-3FBT-7NHO-T26Z-2E6Y-WFH4

UDF locator form URIs presenting a fingerprint type UDF provide a tight binding of the content to the locator. This allows the resolved content to be verified and rejected if it has been modified.

UDF locator form URIs presenting an Encryptor/Authenticator type UDF provide a mechanism for identification, discovery and decryption of encrypted content. UDF locators of this type are known as Encrypted/Authenticated Resource Locators (EARLs).

Regardless of the type of the embedded UDF, UDF locator form URIs are resolved by first performing DNS Web Service Discovery to identify the Web Service Endpoint for the mmm-udf service at the specified domain.

Resolution is completed by presenting the Content Digest Fingerprint of the UDF value specified in the URI to the specified Web Service Endpoint and performing a GET method request on the result.

For example, Alice subscribes to Example.com, a purveyor of cat and kitten images. The company generates paper and electronic invoices on a monthly basis.

To generate the paper invoice, Example.com first creates a new encryption key:

EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z

One or more electronic forms of the invoice are encrypted under the key EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z and placed on the Example.com Web site so that the appropriate version is returned if Alice scans the QR code.

The key is then converted to form an EARL for the example.com UDF resolution service:

udf://example.com/EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z

The EARL is then rendered as a QR code:

Figure 1: QR Code with embedded decryption and location key

A printable invoice containing the QR code is now generated and sent to Alice.

When Alice receives the invoice, she can pay it by simply scanning the invoice with a device that recognizes at least one of the invoice formats supported by Example.com.

The UDF EARL locator shown above is resolved by first determining the Web Service Endpoint for the mmm-udf service for the domain example.com.

```
Discover ("example.com", "mmm-udf") =
https://example.com/.well-known/mmm-udf/
```

Next the fingerprint of the source UDF is obtained.

```
UDF (EA2L-PABQ-ZUIO-UR2U-KLMP-YFYK-XI5U-6Z) =
MBUC-DFHR-NRPA-ZLHR-CZRT-75CV-VWL2-OSIC-2YUH-QJ7C-3WWA-7CLC-W4X7-YNBO
```

Combining the Web Service Endpoint and the fingerprint of the source UDF provides the URI from which the content is obtained using the normal HTTP GET method:

https://example.com/.well-known/mmm-udf/MBUC-DFHR-NRPA-ZLHR-CZRT-75CV-VWL2-OSIC-2YUH-QJ7C-3WWA-7CLC-W4X7-YNBO

Having established that Alice can read postal mail sent to a physical address and having delivered a secret to that address, this process might be extended to provide a means of automating the process of enrolment in electronic delivery of future invoices.

1.3. Secure Internet Names

A SIN is an Internet Identifier that contains a UDF fingerprint of a security policy document that may be used to verify the interpretation of the identifier. This permits traditional forms of Internet address such as URIs and RFC822 email addresses to be used to express a trusted address that is independent of any trusted third party.

This document only describes the syntax and interpretation of the identifiers themselves. The means by which the security policy documents bound to an address govern interpretation of the name is discussed separately in [draft-hallambaker-mesh-trust].

For example, Example Inc holds the domain name example.com and has deployed a private CA whose root of trust is a PKIX certificate with the UDF fingerprint MB2GK-6DUF5-YGYYL-JNY5E-RWSHZ.

Alice is an employee of Example Inc., she uses three email addresses:

alice@example.com A regular email address (not a SIN).

alice@mm--mb2gk-6duf5-ygyyl-jny5e-rwshz.example.com A strong email address that is backwards compatible.

alice@example.com.mm--mb2gk-6duf5-ygyyl-jny5e-rwshz A strong email address that is backwards incompatible.

All three forms of the address are valid RFC822 addresses and may be used in a legacy email client, stored in an address book application, etc. But the ability of a legacy client to make use of the address differs. Addresses of the first type may always be used. Addresses of the second type may only be used if an appropriate MX record is provisioned. Addresses of the third type will always fail unless the resolver understands that it is a SIN requiring special processing.

These rules allow Bob to send email to Alice with either 'best effort' security or mandatory security as the circumstances demand.

2. Definitions

This section presents the related specifications and standard, 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.

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

Commitment A cryptographic primitive that allows one to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later.

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.

Fingerprint A presentation of the digest value of a data value or data object.

Fingerprint Presentation The representation of at least some part of a fingerprint value in human or machine-readable form.

Fingerprint Improvement The practice of recording a higher precision presentation of a fingerprint on successful validation.

Fingerprint Work Hardening The practice of generating a sequence of fingerprints until one is found that matches criteria that permit a compressed presentation form to be used. The compressed fingerprint thus being shorter than but presenting the same work factor as an uncompressed one.

Hash A function which takes an input and returns a fixed-size output. Ideally, the output of a hash function is unbiased and not correlated to the outputs returned to similar inputs in any predictable fashion.

Precision The number of significant bits provided by a Fingerprint Presentation.

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

2.3. Related Specifications

This specification makes use of Base32 [RFC4648] encoding, SHA-2 [SHA-2] and SHA-3 [SHA-3] digest functions in the derivation of basic fingerprints. The derivation of keyed fingerprints additionally requires the use of the HMAC [RFC2014] and HKDF [RFC5869] functions.

Resolution of UDF URI Locators makes use of DNS Web Service Discovery [draft-hallambaker-web-service-discovery].

2.4. Implementation Status

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

3. Architecture

A Uniform Data Fingerprint (UDF) is a presentation of a UDF Binary Data Sequence.

This document specifies seven UDF Binary Data Sequence types and one presentation.

The first octet of a UDF Binary Data Sequence identifies the UDF type and is referred to as the Type identifier.

UDF Binary Data Sequence types are either fixed length or variable length. A variable length Binary Data Sequence **MUST** be truncated for presentation. Fixed length Binary Data Sequences **MUST** not be truncated.

3.1. Base32 Presentation

The default UDF presentation is Base32 Presentation.

Variable length Binary Data Sequences are truncated to an integer multiple of 20 bits that provides the desired precision before conversion to Base32 form.

Fixed length Binary Data Sequences are converted to Base32 form without truncation.

After conversion to Base32 form, dash '-' characters are inserted between groups of 4 characters to aid reading. This representation improves the accuracy of both data entry and verification.

3.1.1. Precision Improvement

Precision improvement is the practice of using a high precision UDF (e.g. 260 bits) calculated from content data that has been validated according to a lower precision UDF (e.g. 120 bits).

This allows a lower precision UDF to be used in a medium such as a business card where space is constrained without compromising subsequent uses.

Applications **SHOULD** make use of precision improvement wherever possible.

3.2. Type Identifier

A Version Identifier consists of a single byte.

The byte codes have been chosen so that the first character of the Base32 presentation of the UDF provides a mnemonic for its type. A SHA-2 fingerprint UDF will always have M (for Merkle Damgard) as the initial letter, a SHA-3 fingerprint UDF will always have K (for Keccak) as the initial letter, and so on.

The following version identifiers are specified in this document:

Type ID	Initial	Algorithm
0	A	HMAC_SHA_2_512
1	A	HMAC_SHA_3_512
32	E	HKDF_AES_512
33	E	HKDF_AES_512
80	K	SHA_3_512
81	K	SHA_3_512 (20 compressed)
82	K	SHA_3_512 (30 compressed)
83	K	SHA_3_512 (40 compressed)
84	K	SHA_3_512 (50 compressed)
96	M	SHA_2_512
97	M	SHA_2_512 (20 compressed)
98	M	SHA_2_512 (30 compressed)
99	M	SHA_2_512 (40 compressed)
100	M	SHA_2_512 (50 compressed)
104	N	Nonce Data
112	O	OID distinguished sequence (DER encoded)
144	S	Shamir Secret Share
200	Z	Secret seed

Table 1

3.3. Content Type Identifier

A secure cryptographic digest algorithm provides a unique digest value that is probabilistically unique for a particular byte sequence but does not fix the context in which a byte sequence is interpreted. While such ambiguity may be tolerated in a fingerprint format designed for a single specific field of use, it is not acceptable in a general-purpose format.

For example, the SSH and OpenPGP applications both make use of fingerprints as identifiers for the public keys used but using different digest algorithms and data formats for representing the public key data. While no such vulnerability has been demonstrated to date, it is certainly conceivable that a crafty attacker might construct an SSH key in such a fashion that OpenPGP interprets the data in an insecure fashion. If the number of applications making use of fingerprint format that permits such substitutions is sufficiently large, the probability of a semantic substitution vulnerability being possible becomes unacceptably large.

A simple control that defeats such attacks is to incorporate a content type identifier within the scope of the data input to the hash function.

3.4. Truncation

Different applications of fingerprints demand different tradeoffs between compactness of the representation and the number of significant bits. A larger the number of significant bits reduces the risk of collision but at a cost to convenience.

Modern cryptographic digest functions such as SHA-2 produce output values of at least 256 bits in length. This is considerably larger than most uses of fingerprints require and certainly greater than can be represented in human readable form on a business card.

Since a strong cryptographic digest function produces an output value in which every bit in the input value affects every bit in the output value with equal probability, it follows that truncating the digest value to produce a finger print is at least as strong as any other mechanism if digest algorithm used is strong.

Using truncation to reduce the precision of the digest function has the advantage that a lower precision fingerprint of some data content is always a prefix of a higher prefix of the same content. This allows higher precision fingerprints to be converted to a lower precision without the need for special tools.

3.4.1. Compressed presentation

The Content Digest UDF types make use of work factor compression. Additional type identifiers are used to indicate digest values with 20, 30, 40 or 50 trailing zero bits allowing a UDF fingerprint offering the equivalent of up to 150 bits of precision to be expressed in 20 characters instead of 30.

To use compressed UDF identifiers, it is necessary to search for content that can be compressed. If the digest algorithm used is secure, this means that by definition, the fastest means of search is brute force. Thus, the reduction in fingerprint size is achieved by transferring the work factor from the attacker to the defender. To maintain a work factor of 2^{120} with an identifier of 2^{80} bits, it is necessary for the content generator to perform a brute force search at a cost of the order of 2^{40} operations.

For example, the smallest allowable work factor for a UDF presentation of a public key fingerprint is 92 bits. This would normally require a presentation with 20 significant characters. Reducing this to 16 characters requires a brute force search of approximately 10^6 attempts. Reducing this to 12 characters would require 10^{12} attempts and to 10 characters, 10^{15} attempts.

Omission of support for higher levels of compression than 2^{50} is intentional.

In addition to allowing use of shorter presentations, work factor compression **MAY** be used as evidence of proof of work.

3.5. Presentation

The presentation of a fingerprint is the format in which it is presented to either an application or the user.

Base32 encoding is used to produce the preferred text representation of a UDF fingerprint. This encoding uses only the letters of the Latin alphabet with numbers chosen to minimize the risk of ambiguity between numbers and letters (2, 3, 4, 5, 6 and 7).

To enhance readability and improve data entry, characters are grouped into groups of four. This means that each block of four characters represents an increase in work factor of approximately one million times.

3.6. Alternative Presentations

Applications that support UDF **MUST** support use of the Base32 presentation. Applications **MAY** support alternative presentations.

3.6.1. Word Lists

The use of a Word List to encode fingerprint values was introduced by Patrick Juola and Philip Zimmerman for the PGPfone application. The PGP Word List is designed to facilitate exchange and verification of fingerprint values in a voice application. To minimize the risk of misinterpretation, two-word lists of 256 values each are used to encode alternative fingerprint bytes. The compact size of the lists used allowed the compilers to curate them so as to maximize the phonetic distance of the words selected.

The PGP Word List is designed to achieve a balance between ease of entry and verification. Applications where only verification is required may be better served by a much larger word list, permitting shorter fingerprint encodings.

For example, a word list with 16384 entries permits 14 bits of the fingerprint to be encoded at once, 65536 entries permit encoding of 16 bits. These encodings allow a 120-bit fingerprint to be encoded in 9 and 8 words respectively.

3.6.2. Image List

An image list is used in the same manner as a word list affording rapid visual verification of a fingerprint value. For obvious reasons, this approach is not suited to data entry but is preferable for comparison purposes. An image list of 1,048,576 images would provide a 20-bit encoding allowing 120 bit precision fingerprints to be displayed in six images.

4. Fixed Length UDFs

Fixed length UDFs are used to represent cryptographic keys, nonces and secret shares and have a fixed length determined by their function that cannot be truncated without loss of information.

All fixed length Binary Data Sequence values are an integer multiple of eight bits.

4.1. Nonce Type

A Nonce Type UDF consists of the type identifier octet 104 followed by the Binary Data Sequence value.

The Binary Data Sequence value is an integer number of octets that **SHOULD** have been generated in accordance with processes and procedures that ensure that it is sufficiently unpredictable for the purposes of the protocol in which the value is to be used. Requirements for such processes and procedures are described in [RFC4086].

Nonce Type UDFs are intended for use in contexts where it is necessary for a randomly chosen value to be unpredictable but not secret. For example, the challenge in a challenge/response mechanism.

4.2. OID Identified Sequence

An OID Identified Sequence Type UDF consists of the type identifier octet 108 followed by the Binary Data Sequence value.

The Binary Data Sequence value is an octet sequence that contains the DER encoding of the following ASN.1 data:

```
OIDInfo ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     data                 BIT STRING  }

AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

OID Identified Sequences are intended to allow arbitrary data sequences to be encoded in the UDF format without exhausting the limited type identifier space.

In particular, OID Identified Sequences **MAY** be used to specify public and private key values.

Given the following Ed25519 public key:

```
AB DB B8 C3  BC B9 A1 34  E6 DB 4D 69  3D AB F5 B4
27 E0 E2 44  75 84 42 59  41 9F 40 EE  32 E9 82 92
```

The equivalent DER encoding is:

```
30 2E 30 07  06 03 2B 65  70 05 00 03  23 00 04 20
AB DB B8 C3  BC B9 A1 34  E6 DB 4D 69  3D AB F5 B4
27 E0 E2 44  75 84 42 59  41 9F 40 EE  32 E9 82 92
```

To encode this key as a UDF OID sequence we prepend the value OID and convert to Base32:

```
OID:        OAYC-4MAH-AYBS-WZLQ-AUAA-GIYA-AQQK-XW5Y-YO6L-TIJU-43NU-2
        2J5-VP23-IJ7A-4JCH-LBCC-LFAZ-6QHO-GLUY-FEQ
```

The corresponding UDF content digest value is more compact and allows us to identify the key unambiguously but does not provide the value:

MDGT-32T4-MBOH-S5XB-7THD-VD6U-4NOU

4.3. Encryption/Authentication Type

Encryption and Authenticator Type UDFs consists of the type identifier specifying the algorithm to be used on the key data followed by the Binary Data Sequence value.

The Binary Data Sequence value is an integer number of octets that **SHOULD** have been generated in accordance with processes and procedures that ensure that it is sufficiently unpredictable and unguessable for the purposes of the protocol in which the value is to be used. Requirements for such processes and procedures are described in [RFC4086].

Encryption and Authenticator Type UDFs are intended to be used as a means of specifying secret cryptographic keying material. For example, the input to a Key Derivation Function used to encrypt a document. Accordingly, the identifier UDF corresponding to an Encryption or Authenticator type UDF is a UDF fingerprint of the UDF in Base32 presentation with content type 'application/udf'.

4.4. Key Pair Derivation

The key pair derivation type is used to specify a public key pair value by means of a sufficiently random input to a deterministic key generation function.

A key pair derivation Type UDF consists of the type identifier octet 200 followed by the Binary Data Sequence value.

The first two octets of the Binary Data Sequence value comprise the Key Specifier which specifies the algorithm and key uses for which the private key is to be derived.

- Bits 6-7 of the first octet specify the key use.
- Bits 0-5 of the first byte and bits 0-7 of the second specify the key type in network byte order.

In the unlikely event that this code space is ever exhausted, allocation of a new UDF type code will be required.

The following key uses are specified:

Code	Algorithm	Description
0	Any	Any
1	Encryption	Encryption
2	Signature	Signature
3	Authentication	Authentication

Table 2

Two types of key type are defined: Explicit and Generic.

Explicit key types specify a public key cryptographic algorithm and all the parameters required to generate a key pair. Generic key types are used to specify a type of key but not the algorithm which **MUST** be specified when the key is generated.

Derivation of key pairs for the following algorithms is specified in this document:

Code	Algorithm	Description
0	Any	Seed MAY be used to generate keypairs for any algorithm
1	X25519	X25519 keypair as described in RFC7748
2	X448	X448 keypair as described in RFC7748
3	Ed25519	Ed25519 keypair as described in RFC8032
4	Ed448	Ed448 keypair as described in RFC8032
5	P256	NIST curve P-256
6	P384	NIST curve P-384
7	P521	NIST curve P-521
8	RSA2048	2048 bit RSA keypair
9	RSA3072	3072 bit RSA keypair
10	RSA4096	4096 bit RSA keypair
11-255	ReservedIetf	Reserved for IETF recommended algorithms
256	MeshProfileDevice	Mesh device profile
257	MeshActivationDevice	Mesh device activation
258	MeshProfileAccount	Mesh account account
259	MeshActivationAccount	Mesh account activation
260	MeshProfileService	Mesh service profile
261	MeshActivationService	Mesh host activation
262-511	ReservedMesh	Reserved for future Mesh use

Table 3

The key parameter derivation function takes as inputs, the UDF seed value seed, the parameter identifier param and an optional string specifying a key name keyname.

```
KeyParam (seed, param, keyname)
```

The value param is an octet sequence determined by the actual key type generated. The first two octets of parm are always equal to the key identifier for the key algorithm and key usage being generated. If the key derivation algorithm requires multiple inputs, additional octets are specified for each of the different inputs required.

The HKDF function [RFC5869] is used to derive key pairs for all the algorithms specified in this document. Derivation functions for additional key algorithms **MAY** use a different function for this purpose provided that it meets the applicable security requirements.

The HKDF function is specified as a two-step extract-expand process with an optional non-secret value input at both steps.

4.4.1. Extraction step

The HKDF extraction step calculates a PRK value from a salt and IKM:

```
HKDF-Extract(salt, IKM) -> PRK
```

The IKM value is the binary presentation of the complete Binary Data Sequence as originally specified. The salt value is null.

The output from the extraction step forms the input to the expand step:

```
HKDF-Expand(PRK, info, L) -> OKM
```

The info parameter of the HKDF function is the concatenation of alg, param and the UTF8 binary representation of keyname.

```
info = alg + param + keyname.UTF8()
```

An X25519 key may be derived as follows:

```
Fingerprint =
    ZAAA-CDJ5-DHPA-DUUW-WIPQ-UXNC-DSAR-U7A

IKM =
  00 01 0D 3D  19 DE 01 D2  96 B2 1F 0A  5D A2 1C 81
  1A 7C

salt =
  00 01

PRK =
  DA 2E 80 6F  2D B1 54 56  7E 27 B4 91  49 0A 35 3A
  5D 99 92 AA  A2 2F 2D 2A  50 4B 13 5B  87 DF 63 67
  62 92 67 9C  B3 B8 10 47  31 52 A2 42  FA 04 84 39
  7A 64 15 84  C0 6B 51 F7  19 4A 20 35  BA 2E D1 59

OKM =
  E7 22 39 E1  AB 77 AC 9C  B4 6A A0 12  27 68 9E 28
  14 60 2F A8  76 08 38 5E  D5 E6 5D E7  0C C8 42 E8

Key =
  E7 22 39 E1  AB 77 AC 9C  B4 6A A0 12  27 68 9E 28
  14 60 2F A8  76 08 38 5E  D5 E6 5D E7  0C C8 42 E8
```

Derivation of an X448 key:

```
Fingerprint =
    ZAAA-FFQA-3LE5-SAHG-E6K6-HOTN-TVLB-K4A

Key =
  AE 6A 6D 0B  CC 48 C3 31  E7 55 0F 52  F9 96 83 C5
  15 7C 8A 74  80 36 B7 E9  17 24 D7 DD  A1 56 76 3C
  15 00 68 B7  23 F5 DB 32  48 1B 72 C0  2E B0 22 45
  A3 B8 80 67  B3 88 06 9F
```

Derivation of an Ed25519 key:

```
Fingerprint =
    ZAAA-GZ5N-PSNF-7LMS-QJZN-3O2X-GJXV-X6I

Key =
  3A 36 00 56  2E EC 2F 24  A7 8C 22 F3  A9 A2 EF 1B
  6E AF 07 D4  99 28 53 A5  5B 0A CC EE  4C 3B 7D 30
```

Derivation of an Ed448 key:

```
Fingerprint =
    ZAAA-ILZB-KTQV-YWUK-FO7E-MQVV-EWPR-UPA

Key =
  DF 5A 89 B8  1D 56 92 41  32 D1 B2 C9  4F 74 69 E3
  C9 E5 5F 23  33 A1 CE 22  54 08 EE 53  46 0F 9B 13
  9D 54 95 2B  F9 D9 77 2A  FA 07 3C 9D  89 CC C5 0E
  7E 86 7E 84  7C 58 5D 89
```

4.4.2. Elliptic Curve Diffie Hellman Key Pairs type 1-4

The generation of key pairs for X25519, X448, Ed25519 and Ed448 is specified in [RFC7748] and [RFC8032]. In each case, the public and private key parameters are generated from a string of random octets whose transformation to the secret scalar function is described in the document.

Thus, info is the null string and the value L is specified as follows:

Algorithm	L
X25519	256
X448	448
Ed25519	256
Ed448	448

Table 4

4.4.3. Elliptic Curve Diffie Hellman Key Pairs type 5-7

The generation of key pairs for the curves P-256, P-384 and P-521 described in [RFC5903] is not mandated by the specification. FIPS 186-4 specifies two approaches. A modified form of the mechanism Key Pair Generation Using Extra Random Bits specified in B.4.1 is used as follows:

The number of random bits L is given by the following table:

Algorithm	L
P-256	320
P-384	448
P-521	592

Table 5

Note that this rounds up the number of random bits required to the nearest integer multiple of 8.

The OKM value is interpreted as an integer in Network Byte Order, that is the first byte contains the most significant bits, to yield the parameter c.

The parameter c is reduced modulo the value of the prime field n to yield the secret scalar value d:

d = (c mod (n?1)) + 1.

A P-256 key may be derived as follows:

```
Fingerprint =
    ZAAA-LLBO-4A4E-LFMH-EJ73-XVFG-7PZ5-V7Y

IKM =
  00 05 AC 2E  E0 38 45 95  87 22 7F BB  D4 A6 FB F3
  DA FF

salt =
  00 05

PRK =
  0F 48 0F 0C  93 30 AE EE  41 FD 8F A2  1C C2 C6 CA
  3A E1 4B 54  E7 83 C0 25  85 F0 CD 2A  65 3F 18 A7
  9F 2A 5A ED  6A E3 64 6A  05 7D 1A 1A  B8 68 B3 F3
  4F A9 10 9A  05 E1 A4 9A  2C CC 40 43  36 8A 24 C0

OKM =
  E2 00 EC 22  63 17 D5 E5  52 F9 CD B6  45 23 A9 8B
  EF 32 26 E0  24 A0 E7 2B  7F CB C2 0B  CB FA 0F 5C
  59 D1 7C 4A  D8 12 2E 4C


Key =   82352103978746514619167815909572981157103618409885983602799410986678676075099
```

Derivation of a P-384 key:

```
Fingerprint =
    ZAAA-NPLI-G7Z3-WFD2-GBJ6-OONN-ELTO-MHA

Key =   369049211431889063087900251703207474490953076630513949620729230126832843214583975749185914333116577244601240468285834
```

Derivation of a P-521 key:

```
Fingerprint =
    ZAAA-PQCC-YFVT-LRWP-7MUZ-GJV3-HLX2-JPQ

Key =   6346542640029401345523427470001786733151503892428821278758 99
            9671798933570807373599102648300868475269998620426934455 0
            4370476919922072068801363203357706689700
```

4.4.4. RSA Key Pairs

Generation of RSA key pairs requires two parameters, p, q which are prime.

The value of the param input used to calculate info is the value of the key identifier value with one of the following tag values concatenated to the end.

Parameter	Tag	UTF8 equivalent string
p	[112]	p
q	[113]	q

Table 6

The value of L is the same for generating the OKM values from which q are derived and is determined by the algorithm identifier:

Algorithm	L
RSA-2048	1024
RSA-3072	1536
RSA-4096	2048

Table 7

The RSA parameter p is the smallest prime integer that is greater than the OKM value corresponding to the info value "p" interpreted as an integer in Network Byte Order.

The RSA parameter q is the smallest prime integer that is greater than the OKM value corresponding to the info value "q" interpreted as an integer in Network Byte Order.

Note that this algorithm does not mandate a particular method of primality testing nor does it impose any additional testing on the values p or q. If an application requires the use of primes with conditions it will be necessary to attempt multiple key derivations with different Binary Data Sequence values until parameters with the desired properties are found.

An RSA-2048 may be derived as follows:

```
Fingerprint =
    ZAAA-RJ5I-OSMI-X2KH-MBHX-KUPB-OC54-NQI

IKM =
  00 08 A7 A8  74 98 8B E9  47 60 4F 75  51 E1 70 BB
  C6 C1

salt =
  00 08

[Generation of the PRK as before]

Info(p) =
  70

OKM(p) =
  92 D4 DA FA  C4 22 DB 17  B0 04 93 C6  F1 D2 7A AF
  34 6F 69 98  54 1A F5 F3  E3 ED DA 98  F5 64 EE 6A

Info(q) =
  71

OKM(q) =
  01 50 07 9F  B3 53 70 5A  7E 95 63 BD  19 8D 52 59
  2F EE 38 E7  8F D4 46 D9  4C 55 E6 DD  39 CA DB 36

Key P = 66413758812235725334838013235321881586339612574162219539634589986848279686793

Key Q = 59371323150670971897831168338735525379591827337915650990989572561891405706 9
```

4.4.5. Any Key Algorithm

The Any key algorithm allows a single UDF value to be used to derive key pairs for multiple algorithms. The IKM value is the same for each key pair derived. The salt value is changed according to the algorithm for which the key is to be derived.

```
Fingerprint =
    ZAAA-A6WP-XMGW-FUOF-2T5L-AHNL-FBPY-RSY

To generate an RSA-2048 key

salt =
  00 08

Key P = 184377705562733023433840697873299239654937691662139401284561
            64676903354830137

Key Q = 741016989403010251265552685128898155357242513700210607224783
            50575007811846521

To generate an X25519 key

salt =
  00 08

Key =
    System.Byte[]
```

4.5. Shamir Shared Secret

The UDF format **MAY** be used to encode shares generated by a secret sharing mechanism. The only secret sharing mechanism currently supported is the Shamir Secret Sharing mechanism [Shamir79]. Each secret share represents a point represents a point *on (x, f(x))*, a polynomial in a modular field *p*. The secret being shared is an integer multiple of 32 bits represented by the polynomial value *f(0)*.

A Shamir Shared Secret Type UDF consists of the type identifier octet 144 followed by the Binary Data Sequence value describing the share value.

The first octet of the Binary Data Sequence value specifies the threshold value and the x value of the particular share:

- *Bits 4-7 of the first byte specify the threshold value.

- *Bits 0-3 of the first byte specify the x value minus 1.

The remaining octets specify the value *f(x)* in network byte (big-endian) order with leading padding if necessary so that the share has the same number of bytes as the secret.

The algorithm requires that the value *p* be a prime larger than the integer representing the largest secret being shared. For compactness of representation we chose *p* to be the smallest prime that is greater than 2^n where *n* is an integer multiple of 32. This approach leaves a small probability that a set of chosen polynomial parameters cause one or more share values be larger than 2^n. Since it is the value of the secret rather than the polynomial parameters that is of important, such parameters **MUST NOT** be used.

4.5.1. Secret Generation

To share a secret of *L* bits with a threshold of *n* we use a *f(x)* a polynomial of degree *n* in the modular field *p*:

$f(x) = a_0 + a_1.x + a_2.x^2 + ... a_n.x^n$

where:

L Is the length of the secret in bits, an integer multiple of 32.

n Is the threshold, the number of shares required to reconstitute the secret.

a_0 Is the integer representation of the secret to be shared.

a_1 ... a_n Are randomly chosen integers less than p

p Is the smallest prime that is greater than 2^L.

For L=128, p = 2^{128}+51.

The values of the key shares are the values *f*(1), *f*(2),... *f*(n).

The most straightforward approach to generation of Shamir secrets is to generate the set of polynomial coefficients, a_0, a_1, ... a_n and use these to generate the share values *f*(1), *f*(2),... *f*(n).

Note that if this approach is adopted, there is a small probability that one or more of the values *f*(1), *f*(2),... *f*(n) exceeds the range of values supported by the encoding. Should this occur, at least one of the polynomial coefficients **MUST** be replaced.

An alternative means of generating the set of secrets is to select up to *n-1* secret share values and use secret recovery to determine at least one additional share. If *n* shares are selected, the shared secret becomes an output of rather than an input to the process.

4.5.2. Recovery

To recover the value of the shared secret, it is necessary to obtain sufficient shares to meet the threshold and recover the value *f(0)* = a_0.

Applications **MAY** employ any approach that returns the correct result. The use of Lagrange basis polynomials is described in Appendix C.

Alice decides to encrypt an important document and split the encryption key so that there are five key shares, three of which will be required to recover the key.

```
Alice's master secret is
  12 0A C3 1B  FF 09 CD 86  CD 3E 6B 4B  CF BA 91 8D
```

This has the UDF representation:

CIFM-GG77-BHGY-NTJ6-NNF4-7OUR-RU

The master secret is converted to an integer applying network byte order conventions. Since the master secret is 128 bits, it is guaranteed to be smaller than the modulus. The resulting value becomes the polynomial value a0.

Since a threshold of three shares is required, we will need a second order polynomial. The co-efficients of the polynomial a1, a2 are random numbers smaller than the modulus:

```
a0 = 239819841806774623580252113294492020 61
a1 = 217449633820028444820075594055263988236
a2 = 299283543253615188179136358544176182525
```

The master secret is the value f(0) = a0. The key shares are the values f(1), f(2)...f(5):

```
f(1) = 200432794333382631893862556497121161315
f(2) = 294885957151441250861223403889609062605
f(3) = 307341472634853319260107753506912905931
f(4) = 237799340783618837090515605349032691293
f(5) = 862595615977378043524469594159684186 91
```

The first byte of each share specifies the recovery information (quorum, x value), the remaining bytes specify the share value in network byte order:

```
f(1) =
  30 96 C9 F3  B9 2E 52 75  AE C9 C5 75  B1 93 D0 B0
  63
f(2) =
  31 DD D8 F8  31 86 F7 0B  B9 E2 74 52  07 3E 2A B8
  CD
f(3) =
  32 E7 37 D0  85 08 F7 8F  A8 17 4B 00  4C CE C8 AA
  CB
f(4) =
  33 B2 E6 7C  B3 B4 54 01  79 68 49 80  82 45 AA 86
  5D
f(5) =
  34 40 E4 FC  BD 89 0C 61  2D D5 6F D2  A7 A2 D0 4B
  83
```

The UDF presentation of the key shares is thus:

```
f(1) = SAYJ-NSPT-XEXF-E5NO-ZHCX-LMMT-2CYG-G
f(2) = SAY5-3WHY-GGDP-OC5Z-4J2F-EBZ6-FK4M-2
f(3) = SAZO-ON6Q-QUEP-PD5I-C5FQ-ATGO-ZCVM-W
f(4) = SAZ3-FZT4-WO2F-IALZ-NBEY-BASF-VKDF-2
f(5) = SA2E-BZH4-XWEQ-YYJN-2VX5-FJ5C-2BFY-G
```

To recover the value f(0) from any three shares, we need to fit a polynomial curve to the three points and use it to calculate the value at x=0 using the Lagrange polynomial basis.

5. Variable Length UDFs

Variable length UDFs are used to represent fingerprint values calculated over a content type identifier and the cryptographic digest of a content data item. The fingerprint value **MAY** be specified at any integer multiple of 20 bits that provides a work factor sufficient for the intended purpose.

Two types of fingerprint are specified:

Digest fingerprints
: Are computed with the same cryptographic digest algorithm used to calculate the digest of the content data.

Message Authentication Code fingerprints
: Are computed using a Message Authentication Code.

For a given algorithm (and key, if requires), if two UDF fingerprints are of the same content data and content type, either the fingerprint values will be the same or the initial characters of one will be exactly equal to the other.

5.1. Content Digest

A Content Digest Type UDF consists of the type identifier octet followed by the Binary Data Sequence value.

The type identifier specifies the digest algorithm used and the compression level. Two digest algorithms are currently specified with four compression levels for each making a total of eight possible type identifiers.

The Content Digest UDF for given content data is generated by the steps of:

0. Applying the digest algorithm to determine the Content Digest Value

1. Applying the digest algorithm to determine the Typed Content Digest Value

2. Determining the compression level from bytes 0-3 of the Typed Content Digest Value.

3. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.

4. Truncating bytes 4-63 of the Typed Content Digest Value to determine the Binary Data Sequence value.

5.1.1. Content Digest Value

The Content Digest Value (CDV) is determined by applying the digest algorithm to the content data:

```
CDV = H(<Data>)
```

Where

H(x) is the cryptographic digest function

<Data> is the binary data.

5.1.2. Typed Content Digest Value

The Typed Content Digest Value (TCDV) is determined by applying the digest algorithm to the content type identifier and the CDV:

```
TCDV = H (<Content-ID> + ?:? + CDV)
```

Where

A + B represents concatenation of the binary sequences A and B.

<Content-ID> is the IANA Content Type of the data in UTF8 encoding

The two-step approach to calculating the Type Content Digest Value allows an application to attempt to match a set of content data against multiple types without the need to recalculate the value of the content data digest.

5.1.3. Content Digest Compression

The compression factor is determined according to the number of trailing zero bits in the first 8 bytes of the Typed Content Digest Value as follows:

19 or fewer trailing zero bits Compression factor = 0

29 or fewer trailing zero bits Compression factor = 20

39 or fewer trailing zero bits Compression factor = 30

49 or fewer trailing zero bits Compression factor = 40

50 or more trailing zero bits Compression factor = 50

The least significant bits of each octet are regarded to be 'trailing'.

Applications **MUST** use compression when creating and comparing UDFs. Applications **MAY** support content generation techniques that search for UDF values that use a compressed representation. Presentation of a content digest value indicating use of compression **MAY** be used as an indicator of 'proof of work'.

5.1.4. Content Digest Presentation

The type identifier is determined by the algorithm and compression factor as specified above.

The Binary Data Sequence value is taken from the Typed Content Digest Value starting at the 9th octet and as many additional bytes as are required to meet the presentation precision.

5.1.5. Example Encoding

In the following examples, <Content-ID> is the UTF8 encoding of the string "text/plain" and <Data> is the UTF8 encoding of the string "UDF Data Value"

```
Data =
  55 44 46 20  44 61 74 61  20 56 61 6C  75 65

ContentType =
  74 65 78 74  2F 70 6C 61  69 6E
```

5.1.6. Using SHA-2-512 Digest

```
H(<Data>) =
  48 DA 47 CC  AB FE A4 5C  76 61 D3 21  BA 34 3E 58
  10 87 2A 03  B4 02 9D AB  84 7C CE D2  22 B6 9C AB
  02 38 D4 E9  1E 2F 6B 36  A0 9E ED 11  09 8A EA AC
  99 D9 E0 BD  EA 47 93 15  BD 7A E9 E1  2E AD C4 15

<Content-ID> + ':' + H(<Data>) =
  74 65 78 74  2F 70 6C 61  69 6E 3A 48  DA 47 CC AB
  FE A4 5C 76  61 D3 21 BA  34 3E 58 10  87 2A 03 B4
  02 9D AB 84  7C CE D2 22  B6 9C AB 02  38 D4 E9 1E
  2F 6B 36 A0  9E ED 11 09  8A EA AC 99  D9 E0 BD EA
  47 93 15 BD  7A E9 E1 2E  AD C4 15

H(<Content-ID> + ':' + H(<Data>)) =
  C6 AF B7 C0  FE BE 04 E5  AE 94 E3 7B  AA 5F 1A 40
  5B A3 CE CC  97 4D 55 C0  9E 61 E4 B0  EF 9C AE F9
  EB 83 BB 9D  5F 0F 39 F6  5F AA 06 DC  67 2A 67 71
  4F FF 8F 83  C4 55 38 36  38 AE 42 7A  82 9C 85 BB

The prefixed Binary Data Sequence is thus
  60 C6 AF B7  C0 FE BE 04  E5 AE 94 E3  7B AA 5F 1A
  40 5B A3 CE  CC 97 4D 55  C0 9E 61 E4  B0 EF 9C AE
  F9 EB 83 BB  9D 5F 0F 39  F6 5F AA 06  DC 67 2A 67
  71 4F FF 8F  83 C4 55 38  36 38 AE 42  7A 82 9C 85
```

The 125 bit fingerprint value is MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF

This fingerprint MAY be specified with higher or lower precision as appropriate.

100 bit precision MDDK-7N6A-727A-JZNO-STRX

120 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7

200 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF-XI6O-ZSLU-2VOA

260 bit precision MDDK-7N6A-727A-JZNO-STRX-XKS7-DJAF-XI6O-ZSLU-2VOA-TZQ6-JMHP-TSXP

5.1.7. Using SHA-3-512 Digest

```
H(<Data>) =
  6D 2E CF E6  93 5A 0C FC  F2 A9 1A 49  E0 0C D8 07
  A1 4E 70 AB  72 94 6E CC  BB 47 48 F1  8E 41 49 95
  07 1D F3 6E  0D 0C 8B 60  39 C1 8E B4  0F 6E C8 08
  65 B4 C4 45  9B A2 7E 97  74 7B BE 68  BC A8 C2 17

<Content-ID> + ':' + H(<Data>) =
  74 65 78 74  2F 70 6C 61  69 6E 3A 6D  2E CF E6 93
  5A 0C FC F2  A9 1A 49 E0  0C D8 07 A1  4E 70 AB 72
  94 6E CC BB  47 48 F1 8E  41 49 95 07  1D F3 6E 0D
  0C 8B 60 39  C1 8E B4 0F  6E C8 08 65  B4 C4 45 9B
  A2 7E 97 74  7B BE 68 BC  A8 C2 17

H(<Content-ID> + ':' + H(<Data>)) =
  8A 86 8A 06  1C 54 6E 7E  3F 75 5F 39  88 F9 FD 2F
  8E C8 45 93  1B 80 A8 2F  29 16 7B A3  BE 21 1F 8A
  75 61 88 A1  D5 7F 07 D5  9D 68 A4 2D  17 F4 4D 23
  F9 E4 0B B2  1A 8D B9 F5  8D FC EC BD  01 F4 37 7C

The prefixed Binary Data Sequence is thus
  50 8A 86 8A  06 1C 54 6E  7E 3F 75 5F  39 88 F9 FD
  2F 8E C8 45  93 1B 80 A8  2F 29 16 7B  A3 BE 21 1F
  8A 75 61 88  A1 D5 7F 07  D5 9D 68 A4  2D 17 F4 4D
  23 F9 E4 0B  B2 1A 8D B9  F5 8D FC EC  BD 01 F4 37
```

The 125 bit fingerprint value is KCFI-NCQG-DRKG-47R7-OVPT-TCHZ-7UXY

5.1.8. Using SHA-2-512 Digest with Compression

The content data "UDF Compressed Document 4187123" produces a UDF Content Digest SHA-2-512 binary value with 20 trailing zeros and is therefore presented using compressed presentation:

```
Data = "
  55 44 46 20  43 6F 6D 70  72 65 73 73  65 64 20 44
  6F 63 75 6D  65 6E 74 20  34 31 38 37  31 32 33"
```

The UTF8 Content Digest is given as:

```
H(<Data>) =
  36 21 FA 2A  C5 D8 62 5C  2D 0B 45 FB  65 93 FC 69
  C1 ED F7 00  AE 6F E3 3D  38 13 FE AB  76 AA 74 13
  6D 5A 2B 20  DE D6 A5 CF  6C 04 E6 56  3F F3 C0 C7
  C4 1D 3F 43  DD DC F1 A5  67 A7 E0 67  9A B0 C6 B7

<Content-ID> + ':' + H(<Data>) =
  74 65 78 74  2F 70 6C 61  69 6E 3A 36  21 FA 2A C5
  D8 62 5C 2D  0B 45 FB 65  93 FC 69 C1  ED F7 00 AE
  6F E3 3D 38  13 FE AB 76  AA 74 13 6D  5A 2B 20 DE
  D6 A5 CF 6C  04 E6 56 3F  F3 C0 C7 C4  1D 3F 43 DD
  DC F1 A5 67  A7 E0 67 9A  B0 C6 B7

H(<Content-ID> + ':' + H(<Data>)) =
  8E 14 D9 19  4E D6 02 12  C3 30 A7 BB  5F C7 17 6D
  AE 9A 56 7C  A8 2A 23 1F  96 75 ED 53  10 EC E8 F2
  60 14 24 D0  C8 BC 55 3D  C0 70 F7 5E  86 38 1A 0B
  CB 55 9C B2  87 81 27 FF  3C EC E2 F0  90 A0 00 00

The prefixed Binary Data Sequence is thus
  61 8E 14 D9  19 4E D6 02  12 C3 30 A7  BB 5F C7 17
  6D AE 9A 56  7C A8 2A 23  1F 96 75 ED  53 10 EC E8
  F2 60 14 24  D0 C8 BC 55  3D C0 70 F7  5E 86 38 1A
  0B CB 55 9C  B2 87 81 27  FF 3C EC E2  F0 90 A0 00
```

The 125 bit fingerprint value is MGHB-JWIZ-J3LA-EEWD-GCT3-WX6H-C5W2

5.1.9. Using SHA-3-512 Digest with Compression

The content data "UDF Compressed Document 774665" produces a UDF Content Digest SHA-3-512 binary value with 20 trailing zeros and is therefore presented using compressed presentation:

```
Data =
  55 44 46 20  43 6F 6D 70  72 65 73 73  65 64 20 44
  6F 63 75 6D  65 6E 74 20  37 37 34 36  36 35
```

The UTF8 SHA-3-512 Content Digest is KEJI-Y225-BDUG-XX22-MXKE-5ITF-YVYM

5.2. Authenticator UDF

An authenticator Type UDF consists of the type identifier octet followed by the Binary Data Sequence value.

The type identifier specifies the digest and Message Authentication Code algorithm. Two algorithm suites are currently specified. Use of compression is not supported.

The Authenticator UDF for given content data and key is generated by the steps of:

0. Applying the digest algorithm to determine the Content Digest Value

1. Applying the MAC algorithm to determine the Authentication value

2. Determining the Type Identifier octet from the Digest algorithm identifier and compression level.

3. Truncating the Authentication value to determine the Binary Data Sequence value.

The key used to calculate and Authenticator type UDF is always a UNICODE string. If use of a binary value as a key is required, the value **MUST** be converted to a string format first. For example, by conversion to an Encryption/Authentication type UDF.

5.2.1. Authentication Content Digest Value

The Content Digest Value (CDV) is determined in the exact same fashion as for a Content Digest UDF by applying the digest algorithm to the content data:

```
CDV = H(<Data>))
```

Where

H(x) is the cryptographic digest function

<Data> is the binary data.

5.2.2. Authentication Value

The Authentication Value (AV) is determined by applying the digest algorithm to the content type identifier and the CDV:

```
AV = MAC (<OKM>, (<Content-ID> + ?:? + CDV))
```

Where

<OKM> is the authentication key as specified below

MAC(<Key>, <data>) is the result of applying the Message Authentication Code algorithm to with Key <Key> and data <data>

The value <OKM> is calculated as follows:

```
IKM = UTF8 (Key)
PRK = MAC (UTF8 ("KeyedUDFMaster"), IKM)
OKM = HKDF-Expand(PRK, UTF8 ("KeyedUDFExpand"), HashLen)
```

Where the function UTF8(string) converts a string to the binary UTF8 representation, HKDF-Expand is as defined in [RFC5869] and the function MAC(k,m) is the HMAC function formed from the specified hash H(m) as specified in [RFC2014].

Keyed UDFs are typically used in circumstances where user interaction requires a cryptographic commitment type functionality

In the following example, <Content-ID> is the UTF8 encoding of the string "text/plain" and <Data> is the UTF8 encoding of the string "Konrad is the traitor". The randomly chosen key is NDD7-6CMX-H2FW-ISAL-K4VB-DQ3E-PEDM.

```
Data =
  4B 6F 6E 72  61 64 20 69  73 20 74 68  65 20 74 72
  61 69 74 6F  72

ContentType =
  74 65 78 74  2F 70 6C 61  69 6E

Key =
  4E 44 44 37  2D 36 43 4D  58 2D 48 32  46 57 2D 49
  53 41 4C 2D  4B 34 56 42  2D 44 51 33  45 2D 50 45
  44 4D
```

Processing is performed in the same manner as an unkeyed fingerprint except that compression is never used:

```
H(<Data>) =
  93 FC DA F9  FA FD 1E 26  50 26 C3 C1  28 43 40 73
  D8 BC 3D 62  87 73 2B 73  B8 EC 93 B6  DE 80 FF DA
  70 0A D1 CE  E8 F4 36 68  EF 4E 71 63  41 53 91 5C
  CE 8C 5C CE  C7 9A 46 94  6A 35 79 F9  33 70 85 01

<Content-ID> + ':' + H(<Data>) =
  74 65 78 74  2F 70 6C 61  69 6E 3A 93  FC DA F9 FA
  FD 1E 26 50  26 C3 C1 28  43 40 73 D8  BC 3D 62 87
  73 2B 73 B8  EC 93 B6 DE  80 FF DA 70  0A D1 CE E8
  F4 36 68 EF  4E 71 63 41  53 91 5C CE  8C 5C CE C7
  9A 46 94 6A  35 79 F9 33  70 85 01

PRK(Key) =
  77 D3 0A 08  39 BD 9D C0  97 44 DA 33  15 0A 42 5E
  CD 17 80 03  B3 CF CC 89  7A C7 84 12  B4 51 5B 25
  DC 26 F5 E1  1B 20 F3 89  2E 9A 1A 7B  0E 73 23 39
  0E C3 4C EF  2D 40 DA 05  B4 70 C6 1C  82 C1 49 33

HKDF(Key) =
  BF A9 B4 58  9C 1D 68 D7  9A B7 11 F6  C8 98 59 14
  20 D7 82 67  C5 84 22 E5  A0 F9 93 52  B1 C3 87 EB
  05 06 CB C4  E4 D6 E6 EE  1F F0 D4 7A  97 68 5E CE
  28 1C CA AF  D8 B5 D1 24  4A 71 EC E3  AC B5 D2 04

MAC(<key>, <Content-ID> + ':' + H(<Data>)) =
  4C C3 7F D3  F9 9E 52 CF  07 90 74 53  84 65 95 BC
  1A 2B A5 D1  68 9D 05 6D  06 C5 CA BF  17 CB E0 49
  95 39 57 08  79 C4 E5 49  D3 3A 59 A3  32 05 45 A6
  30 26 25 AE  8A F4 47 C6  1F B5 33 7F  AD 69 A6 30

The prefixed Binary Data Sequence is thus
  00 4C C3 7F  D3 F9 9E 52  CF 07 90 74  53 84 65 95
  BC 1A 2B A5  D1 68 9D 05  6D 06 C5 CA  BF 17 CB E0
  49 95 39 57  08 79 C4 E5  49 D3 3A 59  A3 32 05 45
  A6 30 26 25  AE 8A F4 47  C6 1F B5 33  7F AD 69 A6
```

The 125 bit fingerprint value is ABGM-G76T-7GPF-FTYH-SB2F-HBDF-SW6B

5.3. Content Type Values

While a UDF fingerprint **MAY** be used to identify any form of static data, the use of a UDF fingerprint to identify a public key signature key provides a level of indirection and thus the ability to identify dynamic data. The content types used to identify public keys are thus of particular interest.

As described in the security considerations section, the use of fingerprints to identify a bare public key and the use of fingerprints to identify a public key and associated security policy information are quite different.

application/pkix-cert A PKIX Certificate

application/pkix-crl A PKIX CRL

application/pkix-keyinfo Content type identifier for PKIX KeyInfo data type

application/pgp-keys Content type identifier for OpenPGP Key

application/dns A DNS resource record in binary format

application/udf-encryption UDF Fingerprint list

application/udf-lock UDF Fingerprint list

5.3.1. PKIX Certificates and Keys

UDF fingerprints **MAY** be used to identify PKIX certificates, CRLs and public keys in the ASN.1 encoding used in PKIX certificates.

Since PKIX certificates and CLRs contain security policy information, UDF fingerprints used to identify certificates or CRLs **SHOULD** be presented with a minimum of 200 bits of precision. PKIX applications **MUST** not accept UDF fingerprints specified with less than 200 bits of precision for purposes of identifying trust anchors.

PKIX certificates, keys and related content data are identified by the following content types:

application/pkix-cert A PKIX Certificate

application/pkix-crl A PKIX CRL

application/pkix-keyinfo The SubjectPublicKeyInfo structure defined in the PKIX certificate specification encoded using DER encoding rules.

The SubjectPublicKeyInfo structure is defined in [RFC5280] as follows:

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }
```

This schema results in an identical DER encoding to the OIDInfo sequence specified in section XXX. The distinction between these productions is that the OIDInfo schema is intended to be used to encode arbitrary data while the application/pkix-keyinfo content type is only intended to be used to describe public keys.

5.3.2. OpenPGP Key

OpenPGPv5 keys and key set content data are identified by the following content type:

application/pgp-keys An OpenPGP key set.

5.3.3. DNSSEC

DNSSEC record data consists of DNS records which are identified by the following content type:

application/dns A DNS resource record in binary format

6. UDF URIs

The UDF URI scheme describes a means of constructing URIs from a UDF value.

Two forms or UDF URI are specified, Name and Locator. In both cases the URI **MUST** specify the scheme type "UDF", and a UDF fingerprint and **MAY** specify a query identifier and/or a fragment identifier.

By definition a Locator form URI contains an authority field which **MUST** be a DNS domain name. The use of IP address forms for this purpose is not permitted.

Name Form URIs allow static content data to be identified without specifying the means by which the content data may be retrieved. Locator form URIs allow static content data or dynamic network resources to be identified and the means of retrieval.

The syntax of a UDF URI is a subset of the generic URI syntax specified in [RFC3986]. The use of userinfo and port numbers is not supported and the path part of the uri is a UDF in base32 presentation.

```
URI           = "UDF:" udf [ "?" query ] [ "" fragment ]

udf           = name-form / locator-form

name-form     = udf-value
locator-form  = "//" authority "/" udf-value

authority     = host
host          = reg-name
```

6.1. Name form URI

Name form UDF URIs provide a means of presenting a UDF value in a context in which a URI form of a name is required without providing a means of resolution.

Adding the UDF scheme prefix to a UDF fingerprint does not change the semantics of the fingerprint itself. The semantics of the name result from the context in which it is used.

For example, a UDF value of any type **MAY** be used to give a unique targetNamespace value in an XML Schema [XMLSchema]

6.2. Locator form URI

The locator form of an unkeyed UDF URI is resolved by the following steps:

- Use DNS Web service discovery to determine the Web Service Endpoint.
- Determine the content identifier from the source URI.
- Append the content identifier to the Web Service Endpoint as a suffix to form the target URI.
- Retrieve content from the Web Service Endpoint by means of a GET method.
- Perform post processing as specified by the UDF type.

6.2.1. DNS Web service discovery

DNS Web Discovery is performed as specified in [draft-hallambaker-web-service-discovery] for the service mmm-udf and domain name specified in the URI. For a full description of the discovery mechanism, consult the referenced specification.

The use of DNS Web Discovery permits service providers to make full use of the load balancing and service description capabilities afforded by use of DNS SRV and TXT records in accordance with the approach described in [RFC6763].

If no SRV or TXT records are specified, DNS Web Discovery specifies that the Web Service Endpoint be the Well Known Service [RFC5785] with the prefix /.well-known/srv/mmm-udf.

6.2.2. Content Identifier

For all UDF types other than Secret Share, the Content Identifier value is the UDF SHA-2-512 Content Digest of the canonical form of the UDF specified in the source URI presented at twice the precision to a maximum of 440 bits.

If the UDF is of type Secret Share, the shared secret **MUST** be recovered before the content identifier can be resolved. The shared secret is then expressed as a UDF of type Encryption/Authentication and the Content Identifier determined as for an Encryption/ Authentication type UDF.

6.2.3. Target URI

The target URI is formed by appending a slash separator '/' and the Content Identifier value to the Web Service Endpoint.

Since the path portion of a URI is case sensitive, the UDF value **MUST** be specified in upper case and **MUST** include separator marks.

6.2.4. Postprocessing

After retrieving the content data, the resolver **MUST** perform post processing as indicated by the content type:

Nonce No additional post processing is required.

Content Digest The resolver **MUST** verify that the content returned matches the UDF fingerprint value.

Authenticator The resolver **MUST** verify that the content returned matches the UDF fingerprint value.

Encryption/Authentication The content data returned is decrypted and authenticated using the key specified in the UDF value as the initial keying material (see below).

Secret Share (set) The content data returned is decrypted and authenticated using the shared secret as the initial keying material (see below).

6.2.5. Decryption and Authentication

The steps performed to decode cryptographically enhanced content data depends on the content type specified in the returned content. Two formats are currently supported:

* DARE Envelope format as specified in [draft-hallambaker-mesh-dare]

* Cryptographic Message Syntax (CMS) Symmetric Key Package as specified in [RFC6031]

6.2.6. QR Presentation

Encoding of a UDF URI as a QR code requires only the characters in alphanumeric encoding, thus achieving compactness with minimal overhead.

7. Strong Internet Names

A Strong Internet Name is an Internet address that is bound to a policy governing interpretation of that address by means of a Content Digest type UDF of the policy expressed as a UDF prefixed DNS label within the address itself.

The Reserved LDH labels as defined in [RFC5890] that begin with the prefix mm-- are reserved for use as Strong Internet Names. The characters following the prefix are a Content Digest type UDF in Base32 presentation.

Since DNS labels are limited to 63 characters, the presentation of the SIN itself is limited to 59 characters and thus 240 bits of precision.

8. Security Considerations

This section describes security considerations arising from the use of UDF in general applications.

Additional security considerations for use of UDFs in Mesh services and applications are described in the Mesh Security Considerations guide [draft-hallambaker-mesh-security].

8.1. Confidentiality

Encrypted locator is a bearer token

8.2. Availability

Corruption of a part of a shared secret may prevent recovery

8.3. Integrity

Shared secret parts do not contain context information to specify which secret they relate to.

8.4. Work Factor and Precision

A given UDF data object has a single fingerprint value that may be presented at different precisions. The shortest legitimate precision with which a UDF fingerprint may be presented has 96 significant bits

A UDF fingerprint presents the same work factor as any other cryptographic digest function. The difficulty of finding a second data item that matches a given fingerprint is 2^n and the difficulty or finding two data items that have the same fingerprint is $2^{(n/2)}$. Where n is the precision of the fingerprint.

For the algorithms specified in this document, n = 512 and thus the work factor for finding collisions is 2^{256}, a value that is generally considered to be computationally infeasible.

Since the use of 512 bit fingerprints is impractical in the type of applications where fingerprints are generally used, truncation is a practical necessity. The longer a fingerprint is, the less likely it is that a user will check every character. It is therefore important to consider carefully whether the security of an application depends on second pre-image resistance or collision resistance.

In most fingerprint applications, such as the use of fingerprints to identify public keys, the fact that a malicious party might generate two keys that have the same fingerprint value is a minor concern. Combined with a flawed protocol architecture, such a vulnerability may permit an attacker to construct a document such that the signature will be accepted as valid by some parties but not by others.

For example, Alice generates keypairs until two are generated that have the same 100 bit UDF presentation (typically 2^{48} attempts). She registers one keypair with a merchant and the other with her bank. This allows Alice to create a payment instrument that will be accepted as valid by one and rejected by the other.

The ability to generate of two PKIX certificates with the same fingerprint and different certificate attributes raises very different and more serious security concerns. For example, an attacker might generate two certificates with the same key and different use constraints. This might allow an attacker to present a highly constrained certificate that does not present a security risk

to an application for purposes of gaining approval and an unconstrained certificate to request a malicious action.

In general, any use of fingerprints to identify data that has security policy semantics requires the risk of collision attacks to be considered. For this reason, the use of short, 'user friendly' fingerprint presentations (Less than 200 bits) **SHOULD** only be used for public key values.

8.5. 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.

The UDF format provides protection against high level semantic substitution attacks by incorporating the content type into the input to the outermost fingerprint digest function. The work factor for generating a UDF fingerprint that is valid in both contexts is thus the same as the work factor for finding a second preimage in the digest function (2^{512} for the specified digest algorithms).

It is thus infeasible to generate a data item such that some applications will interpret it as a PKIX key and others will accept as an OpenPGP key. While attempting to parse a PKIX key as an OpenPGP key is virtually certain to fail to return the correct key parameters it cannot be assumed that the attempt is guaranteed to fail with an error message.

The UDF format does not provide protection against semantic substitution attacks that do not affect the content type.

8.6. 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.

9. IANA Considerations

Registrations are requested in the following registries:

- *Service Name and Transport Protocol Port Number
- *well-known URI registry
- *Uniform Resource Identifier (URI) Schemes
- *Media Types

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

9.1. Protocol Service Name

The following registration is requested in the Service Name and Transport Protocol Port Number Registry in accordance with [RFC6355]

Service Name (REQUIRED) mmm-udf

Transport Protocol(s) (REQUIRED) TCP

Assignee (REQUIRED) Phillip Hallam-Baker, phill@hallambaker.com

Contact (REQUIRED) Phillip Hallam-Baker, phill@hallambaker.com

Description (REQUIRED) mmm-udf is a Web Service protocol that resolves Mathematical Mesh Uniform Data Fingerprints (UDF) to resources. The mmm-udf service name is used in service discovery to identify a Web Service endpoint to perform resolution of a UDF presented in URI locator form.

Reference (REQUIRED) [This document]

Port Number (OPTIONAL) None

Service Code (REQUIRED for DCCP only) None

Known Unauthorized Uses (OPTIONAL) None

Assignment Notes (OPTIONAL) None

9.2. Well Known

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

URI suffix
srv/mmm-udf

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

Specification document(s): [This document]

Related information [draft-hallambaker-web-service-discovery]

9.3. URI Registration

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

Scheme name: UDF

Status: Provisional

Applications/protocols that use this scheme name: Mathematical Mesh Service protocols (mmm)

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

Change controller: Phillip Hallam-Baker

References: [This document]

9.4. Media Types Registrations

9.4.1. Media Type: application/pkix-keyinfo

Type name: application

Subtype name: pkix-keyinfo

Required parameters: None

Optional parameters: None

Encoding considerations: Binary

Security considerations: Described in [This]

Interoperability considerations: None

Published specification: [This]

Applications that use this media type: Uniform Data Fingerprint

Fragment identifier considerations: None

Additional information:
Deprecated alias names for this type: None

Magic number(s): None

File extension(s): None

Macintosh file type code(s): None

Person & email address to contact for further information: Phillip Hallam-Baker @hallambaker.com>

Intended usage: Content type identifier to be used in constructing UDF Content Digests and Authenticators and related cryptographic purposes.

Restrictions on usage: None

Author: Phillip Hallam-Baker

Change controller: Phillip Hallam-Baker

Provisional registration? (standards tree only): Yes

9.4.2. Media Type: application/udf

Type name: application

Subtype name: udf

Required parameters: None

Optional parameters: None

Encoding considerations: None

Security considerations: Described in [This]

Interoperability considerations: None

Published specification: [This]

Applications that use this media type: Uniform Data Fingerprint

Fragment identifier considerations: None

Additional information: Deprecated alias names for this type: None

Magic number(s): None

File extension(s): None

Macintosh file type code(s): None

Person & email address to contact for further information: Phillip Hallam-Baker @hallambaker.com>

Intended usage: Content type identifier to be used in constructing UDF Content Digests and Authenticators and related cryptographic purposes.

Restrictions on usage: None

Author: Phillip Hallam-Baker

Change controller: Phillip Hallam-Baker

Provisional registration? (standards tree only): Yes

9.5. Uniform Data Fingerprint Type Identifier Registry

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

9.5.1. The name of the registry

Uniform Data Fingerprint Type Identifier Registry

9.5.2. 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.

9.5.3. 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.

9.5.4. Size, format, and syntax of registry entries

Each registry entry consists of a single byte code,

9.5.5. Initial assignments and reservations

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

Code	Description	Reference
0	HMAC_SHA_2_512	[This document]
1	HMAC_SHA_3_512	[This document]
32	HKDF_AES_512	[This document]
33	HKDF_AES_512	[This document]
80	SHA_3_512	[This document]
81	SHA_3_512 (20 compressed)	[This document]
82	SHA_3_512 (30 compressed)	[This document]
83	SHA_3_512 (40 compressed)	[This document]
84	SHA_3_512 (50 compressed)	[This document]
96	SHA_2_512	[This document]
97	SHA_2_512 (20 compressed)	[This document]
98	SHA_2_512 (30 compressed)	[This document]
99	SHA_2_512 (40 compressed)	[This document]
100	SHA_2_512 (50 compressed)	[This document]
104	Nonce Data	[This document]
112	OID distinguished sequence (DER encoded)	[This document]
144	Shamir Secret Share	[This document]
200	Secret seed	[This document]

10. Acknowledgements

A list of people who have contributed to the design of the Mesh is presented in [draft-hallambaker-mesh-architecture].

Thanks are due to Viktor Dukhovni, Damian Weber and an anonymous member of the cryptography@metzdowd.com list for assisting in the compilation of the table of prime values.

11. Appendix A: Prime Values for Secret Sharing

The following are the prime values to be used for sharing secrets of up to 512 bits.

If it is necessary to share larger secrets, the corresponding prime may be found by choosing a value $(2^8)^n$ that is larger than the secret to be encoded and determining the next largest number that is prime.

Bytes	Bits	Prime
1	8	2^8+1
2	16	2^16+1
3	24	2^24+43
4	32	2^32+15
5	40	2^40+15
6	48	2^48+21
7	56	2^56+81
8	64	2^64+13
9	72	2^72+15
10	80	2^80+13
11	88	2^88+7
12	96	2^96+61
13	104	2^104+111
14	112	2^112+25
15	120	2^120+451
16	128	2^128+51
17	136	2^136+85
18	144	2^144+175
19	152	2^152+253
20	160	2^160+7
21	168	2^168+87
22	176	2^176+427
23	184	2^184+27
24	192	2^192+133
25	200	2^200+235
26	208	2^208+375
27	216	2^216+423
28	224	2^224+735
29	232	2^232+357
30	240	2^240+115
31	248	2^248+81
32	256	2^256+297
33	264	2^264+175
34	272	2^272+57
35	280	2^280+45
36	288	2^288+127
37	296	2^296+61

Bytes	Bits	Prime
38	304	2^304+37
39	312	2^312+91
40	320	2^320+27
41	328	2^328+15
42	336	2^336+241
43	344	2^344+231
44	352	2^352+55
45	360	2^360+105
46	368	2^368+127
47	376	2^376+115
48	384	2^384+231
49	392	2^392+207
50	400	2^400+181
51	408	2^408+37
52	416	2^416+235
53	424	2^424+163
54	432	2^432+1093
55	440	2^440+187
56	448	2^448+211
57	456	2^456+21
58	464	2^464+841
59	472	2^472+445
60	480	2^480+165
61	488	2^488+777
62	496	2^496+583
63	504	2^504+133
64	512	2^512+75

Table 8

For example, the prime to be used to share a 128 bit value is 2^{128} + 51.

12. Appendix B: Shamir Shared Secret Recovery Using Lagrange Interpolation

The value of a Shamir Shared secret may be recovered using Lagrange basis polynomials.

To share a secret with a threshold of *n* shares and L bits we constructed *f(x)* a polynomial of degree *n* in the modular field *p* where *p* is the smallest prime greater than 2^L:

$f(x) = a_0 + a_1.x + a_2.x^2 + ... a_n.x^n$

The shared secret is the binary representation of the value a_0

Given n shares (x_0, y_0), (x_1, y_1), ... (x_{n-1}, y_{n-1}), The corresponding the Lagrange basis polynomials l_0, l_1, .. l_{n-1} are given by:

$$lm = ((x - x(m_0)) / (x(m) - x(m_0))) . ((x - x(m_1)) / (x(m) - x(m_1))) . ... . ((x - x(m_{n-2})) / (x(m) - x(m_{n-2})))$$

Where the values m_0, m_1, ... m_{n-2}, are the integers 0, 1, .. n-1, excluding the value m.

These can be used to compute $f(x)$ as follows:

$$f(x) = y_0 l_0 + y_1 l_1 + ... y_{n-1} l_{n-1}$$

Since it is only the value of $f(0)$ that we are interested in, we compute the Lagrange basis for the value $x = 0$:

$$lz_m = ((x(m_1)) / (x(m) - x(m_1))) . ((x(m_2)) / (x(m) - x(m_2)))$$

Hence,

$$a_0 = f(0) = y_0 lz_0 + y_1 lz_1 + ... y_{n-1} l_{n-1}$$

The following C# code recovers the values.

```
using System;
using System.Collections.Generic;
using System.Numerics;

namespace Examples {

    class Examples {

        ///
        /// Combine a set of  points (x, f(x))
        /// on a polynomial of degree  in a
        /// discrete field modulo prime  to
        /// recover the value f(0) using Lagrange basis polynomials.
        ///
        /// The values f(x).
        /// The values for x.
        /// The modulus.
        /// The polynomial degree.
        /// The value f(0).
        static BigInteger CombineNK(
                    BigInteger[] fx,
                    int[] x,
                    BigInteger p,
                    int n) {
            if (fx.Length < n) {
                throw new Exception("Insufficient shares");
                }

            BigInteger accumulator = 0;
            for (var formula = 0; formula < n; formula++) {
                var value = fx[formula];

                BigInteger numerator = 1, denominator = 1;
                for (var count = 0; count < n; count++) {
                    if (formula == count) {
                        continue;  // If not the same value
                        }

                    var start = x[formula];
                    var next = x[count];

                    numerator = (numerator * -next) % p;
                    denominator = (denominator * (start - next)) % p;
                    }

                var InvDenominator = ModInverse(denominator, p);

                accumulator = Modulus((accumulator +
                    (fx[formula] * numerator * InvDenominator)), p);
                }
```

```
            return accumulator;
            }

        ///
        /// Compute the modular multiplicative inverse of the value
        ///  modulo
        ///
        /// The value to find the inverse of
        /// The modulus.
        ///
        static BigInteger ModInverse(
                    BigInteger k,
                    BigInteger p) {
            var m2 = p - 2;
            if (k < 0) {
                k = k + p;
                }

            return BigInteger.ModPow(k, m2, p);
            }

        ///
        /// Calculate the modulus of a number with correct handling
        /// for negative numbers.
        ///
        /// Value
        /// The modulus.
        /// x mod p
        public static BigInteger Modulus(
                    BigInteger x,
                    BigInteger p) {
            var Result = x % p;
            return Result.Sign >= 0 ? Result : Result + p;
            }
        }
    }
```

13. Normative References

[draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021,

<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>.

[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-14, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-dare-14>.

[draft-hallambaker-mesh-security]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part IX Security Considerations", Work in Progress, Internet-Draft, draft-hallambaker-mesh-security-08, 20 September 2021, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-security-08>.

[draft-hallambaker-web-service-discovery]
Hallam-Baker, P., "DNS Web Service Discovery", Work in Progress, Internet-Draft, draft-hallambaker-web-service-discovery-06, 5 August 2021, <https://datatracker.ietf.org/doc/html/draft-hallambaker-web-service-discovery-06>.

[RFC2014] Weinrib, A. and J. Postel, "IRTF Research Group Guidelines and Procedures", BCP 8, RFC 2014, DOI 10.17487/RFC2014, October 1996, <https://www.rfc-editor.org/rfc/rfc2014>.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation

List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[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>.

[RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, DOI 10.17487/RFC5903, June 2010, <https://www.rfc-editor.org/rfc/rfc5903>.

[RFC6031] Turner, S. and R. Housley, "Cryptographic Message Syntax (CMS) Symmetric Key Package Content Type", RFC 6031, DOI 10.17487/RFC6031, December 2010, <https://www.rfc-editor.org/rfc/rfc6031>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/rfc/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[SHA-2] NIST, "Secure Hash Standard", August 2015.

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

14. Informative References

[draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", Work in Progress, Internet-Draft, draft-hallambaker-mesh-developer-10, 27 July 2020, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-developer-10>.

[draft-hallambaker-mesh-trust]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part X: The Trust Mesh", Work in Progress, Internet-Draft, draft-hallambaker-mesh-trust-09, 5 August 2021, <https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-trust-09>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC

4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <https://www.rfc-editor.org/rfc/rfc4880>.

[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>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/rfc/rfc5890>.

[RFC6355] Narten, T. and J. Johnson, "Definition of the UUID-Based DHCPv6 Unique Identifier (DUID-UUID)", RFC 6355, DOI 10.17487/RFC6355, August 2011, <https://www.rfc-editor.org/rfc/rfc6355>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/rfc/rfc6763>.

[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>.

[Shamir79] Shamir, A., "How to share a secret.", 1979.

[XMLSchema] Gao, S., Sperberg-McQueen, C. M., Thompson, H. S., Mendelsohn, N., Beech, D., and M. Maloney, "W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures", 5 April 2012.