Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 10, 2019

P. Hallam-Baker
Comodo Group Inc.

January 6, 2019

Uniform Data Fingerprint (UDF)
draft-hallambaker-udf-12

Abstract

This document describes means of generating Uniform Data Fingerprint (UDF) values and their presentation as text sequences and as URIs. Uses of UDF fingerprints include but are not limited to creating Strong Internet Names (SINs).

Cryptographic digests provide a means of uniquely identifying static data without the need for a registration authority. A fingerprint is a form of presenting a cryptographic digest that makes it suitable for use in applications where human readability is required. The UDF fingerprint format improves over existing formats through the introduction of a compact algorithm identifier affording an intentionally limited choice of digest algorithm and the inclusion of an IANA registered MIME Content-Type identifier within the scope of the digest input to allow the use of a single fingerprint format in multiple application domains.

Alternative means of rendering fingerprint values are considered including machine-readable codes, word and image lists.

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-udf.html [1] .

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 July 10, 2019.

Table of Contents

1. Introduction

The use of cryptographic digest functions to produce identifiers is well established as a means of generating a unique identifier for fixed data without the need for a registration authority.

While the use of fingerprints of public keys was popularized by PGP, they are employed in many other applications including OpenPGP, SSH, BitCoin and PKIX.

A cryptographic digest is a particular form of hash function that has the properties:

- It is easy to compute the digest value for any given message
- It is infeasible to generate a message from its digest value
- It is infeasible to modify a message without changing the digest value
- It is infeasible to find two different messages with the same digest value.

If these properties are met, the only way that two data objects that map to the same digest value is by random chance. If the number of possible digest values is sufficiently large (i.e. is a sufficiently large number of bits in length), this chance is reduced to an arbitrarily infinitesimal probability. Such values are described as being probabilistically unique.

A fingerprint is a representation of a cryptographic digest value optimized for purposes of verification and in some cases data entry.

1.1. Algorithm Identifier

Although a secure cryptographic digest algorithm has properties that make it ideal for certain types of identifier use, several cryptographic digest algorithms have found widespread use, some of which have been demonstrated to be insecure.

For example the MD5 message digest algorithm [RFC1321] , was widely used in IETF protocols until it was demonstrated to be vulnerable to collision attacks [Dobertin95] .

The secure use of a fingerprint scheme therefore requires the digest algorithm to either be fixed or otherwise determined by the fingerprint value itself. Otherwise an attacker may be able to use a weak, broken digest algorithm to generate a data object matching a fingerprint value generated using a strong digest algorithm.

The two digest algorithms currently used in the UDF scheme are both believed to be strong. These are SHA-2-512 [SHA-2] and SHA-3-512 [SHA-3] . The most secure, 512 bit version of the algorithm is used in both cases although the output is almost invariably truncated to a shorter length. Use of the strongest version of the algorithm in every circumstance eliminates the need to negotiate the algorithm strength.

1.2. 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.

1.3. Representation

The representation 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 five.

1.4. 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.

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.

UDFs are used in the definition of Strong Internet Names [hallambaker-sin] .

2.4. Implementation Status

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

3. UDF Fingerprint

A UDF fingerprint for a given data object is generated by calculating the Binary Fingerprint Value for the given data object and type identifier, truncating it to obtain the desired degree of precision and then converting the truncated value to a representation.

3.1. Binary Fingerprint Value

The binary encoding of a fingerprint is calculated using the formula:

Fingerprint = <Version-ID> + H (<Content-ID> + ?:? + H(<Data>))

Figure 1

Where

H(x) is the cryptographic digest function
<Version-ID> is the fingerprint version and algorithm identifier.
<Content-ID> is the MIME Content-Type of the data.
<Data> is the binary data.

Figure 2

The use of the nested hash function permits a fingerprint to be taken of data for which a digest value is already known without the need to calculate a new digest over the data.

The inclusion of a MIME content type prevents message substitution attacks in which one content type is substituted for another.

3.1.1. Version ID

A Version Identifier consists of a single byte. The following digest algorithm identifiers are specified in this document:

Version ID	Algorithm	Reference
80	HMAC-SHA-2-512	<norm="RFC2014"/>
96	SHA-2-512	<norm="SHA-2"/>
97-100	SHA-2-512 (compressed)	<norm="SHA-2"/>
136	Random data	
144	SHA-3-512	<norm="SHA-3"/>
145-149	SHA-3-512 (compressed)	<norm="SHA-3"/>

Table 1

These algorithm identifiers have been chosen so that the first character in a SHA-2-512 fingerprint will always be ?M? and the first character in a SHA-3-512 fingerprint will always be ?S?. These provide mnemonics for ?Merkle-Damgard? and ?Sponge? respectively.

The first character of a keyed fingerprint will be 'K', a mnemonic for 'Keyed'.

The version id 135 is used to identify random data such as nonce values with a first character mnemonic of 'R'. While such data is typically generated using a digest function, there is no need to specify which one was used.

3.2. Truncation

The Binary Fingerprint Value is truncated to an integer multiple of 25 bits regardless of the intended output presentation.

The output of the hash function is truncated to a sequence of n bits by first selecting the first n/8 bytes of the output function. If n is an integer multiple of 8, no additional bits are required and this is the result. Otherwise the remaining bits are taken from the most significant bits of the next byte and any unused bits set to 0.

For example, to truncate the byte sequence [a0, b1, c2, d3, e4] to 25 bits. 25/8 = 3 bytes with 1 bit remaining, the first three bytes of the truncated sequence is [a0, b1, c2] and the final byte is e4 AND 80 = 80 which we add to the previous result to obtain the final truncated sequence of [a0, b1, c2, 80]

3.3. Base32 Representation

A modified version of Base32 [RFC4648] encoding is used to present the fingerprint in text form grouping the output text into groups of five characters separated by a dash ?-?. This representation improves the accuracy of both data entry and verification.

3.4. 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
```

Figure 3

3.4.1. 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

Prefixed, compressed, trimmed =
  60 C6 AF B7  C0 FE BE 04  E5 AE 94 E3  7B AA 5F 1A
  40 ...
```

Figure 4

The 125 bit fingerprint value is MDDK7-N6A72-7AJZN-OSTRX-XKS7D

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

100 bit precision MDDK7-N6A72-7AJZN-OSTRX

150 bit precision MDDK7-N6A72-7AJZN-OSTRX-XKS7D-JAFXI

200 bit precision MDDK7-N6A72-7AJZN-OSTRX-XKS7D-JAFXI-6OZSL-U2VOA

250 bit precision MDDK7-N6A72-7AJZN-OSTRX-XKS7D-JAFXI-6OZSL-U2VOA-TZQ6J-MHPTS

3.4.2. 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

Prefixed, compressed, trimmed =
  90 8A 86 8A  06 1C 54 6E  7E 3F 75 5F  39 88 F9 FD
  2F ...
```

Figure 5

The 125 bit fingerprint value is SCFIN-CQGDR-KG47R-7OVPT-TCHZ7

3.5. Fingerprint Improvement

Since an application must always calculate the full fingerprint value as part of the verification process, an application MAY accept a low precision (e.g. 100 bit) fingerprint value from the user and replace it with a higher precision fingerprint (e.g. 250 bits) after verification.

Applications are encouraged to make use of the practice of fingerprint improvement wherever possible.

3.6. Compressed Presentation

Fingerprint compression permits the use of shorter fingerprint presentation without a reduction in the attacker work factor by requiring the fingerprint value to match a particular pattern.

UDF fingerprints MUST use compression if possible. A compressed fingerprint uses a version identifier that specifies the form of compression used as follows:

Version ID	Algorithm	Compression
96	SHA-2-512	None
97	SHA-2-512	First 24 bits are zeros
98	SHA-2-512	First 32 bits are zeros
99	SHA-2-512	First 40 bits are zeros
100	SHA-2-512	First 48 bits are zeros
101	SHA-2-512	First 56 bits are zeros
144	SHA-3-512	None
145	SHA-3-512	First 24 bits are zeros
146	SHA-3-512	First 32 bits are zeros
147	SHA-3-512	First 40 bits are zeros
148	SHA-3-512	First 48 bits are zeros
149	SHA-3-512	First 56 bits are zeros

Table 2

The compression prefixes are all multiples of 8 bits for ease of implementation.

Currently, 24 bit compression may be achieved on commodity machines with modest impact on key generation allowing use of a 100 bit (i.e. 20 character) presentation for a slight reduction in work factor. Use of 40 bit compression has a noticeable impact, but can still be achieved within hours without the use of special purpose hardware (e.g. use of a GPU unit). Use of 48 bit compression is feasible with a GPU and use of 56 bit compression which would allow a fingerprint to be shortened by ten significant characters with increased work factor is on the outer edge of practicality. While support for even higher levels of compression is conceivable, it is probably not very sensible.

Support for compression may introduce perverse incentives such as performing key generation on machines that less secure but offer fast (or cheap) processing power. An attacker might even offer to generate public key pairs for free using their 'ultra fast' machine. For this reason, it is probably desirable to at least support if not mandate the use of some sort of salting scheme when compression is in use. This allows the key to be generated in secure, trusted hardware and only the discovery of a salt providing the desired compression being performed on less trusted or untrusted devices. Such approaches are outside the scope of this specification and certain implementations may be subject to intellectual property claims.

3.6.1. Example of Compressed Encoding.

The string "290668103" has a SHA-2-512 UDF fingerprint with 29 leading zero bits. The inputs to the fingerprint are:

```
Data =
  32 39 30 36  36 38 31 30  33

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

H ( <Content-ID> + ':' + H(<Data>))=

  AF ED 7C 65  22 CD 97 28  C9 1F AA D8  23 B6 0A 7C
  1F 5B DB 51  D1 25 FE 15  FB DC 13 4D  54 80 67 3E
  FA 91 5E F8  B1 57 AC A2  5A E5 EE D5  E9 AC B9 EE
  1B 43 F1 23  2B F8 2E 01  EA 7F 34 24  47 FF 2C 13
```

Figure 6

Since the first three bytes of the final hash value are zeros, these are dropped and the version identifier increased by 1:

```
H(<Data>) =
  61 31 26 83  80 72 8F BF  C5 86 94 57  28 E6 82 E9
  7F 7D 2D 99  34 88 C1 7B  5A 26 D8 C2  B0 22 45 07
  1C 2A 76 16  AC F7 C7 66  BA 66 26 E8  B4 65 84 51
  8C BE B3 87  DF B7 7B 05  B4 69 BE 9C  BB AF C9 F3

<Content-ID> + ':' + H(<Data>) =
  74 65 78 74  2F 70 6C 61  69 6E 3A 61  31 26 83 80
  72 8F BF C5  86 94 57 28  E6 82 E9 7F  7D 2D 99 34
  88 C1 7B 5A  26 D8 C2 B0  22 45 07 1C  2A 76 16 AC
  F7 C7 66 BA  66 26 E8 B4  65 84 51 8C  BE B3 87 DF
  B7 7B 05 B4  69 BE 9C BB  AF C9 F3

H(<Content-ID> + ':' + H(<Data>)) =
  00 00 00 3B  AD 4A E2 93  42 5C 6C E9  08 00 3D 4A
  84 95 34 BB  CE C6 6C AC  9E C1 E0 C0  72 E2 43 5D
  CA 91 F7 84  93 E5 66 BC  DE BC 93 F9  FA 52 27 98
  86 DA EC CE  A1 4D EE 55  C0 27 15 41  6B 30 8F 0E

Prefixed, compressed, trimmed =
  61 3B AD 4A  E2 93 42 5C  6C E9 08 00  3D 4A 84 95
  34 ...
```

Figure 7

The 125 bit fingerprint value is ME522-SXCSN-BFY3H-JBAAD-2SUES

Note that the use of compression does not reduce the number of characters presented. Compression increases the work factor that is achieved for a given fingerprint length but does not in itself cause the presentation to be changed.

The 125 bit UDF of the string "44870804" using SHA-3-512 is SETHM-SHUAF-R7L7V-HRIEW-MQ5KT.

4. UDF Keyed Fingerprint

A Keyed UDF is a fingerprint derived using a Message Authentication Code rather than a digest.

The inputs to the fingerprint function are

<Data> The content data

<Content-ID> The IANA content identifier

<Version-ID> Identifies the content digest and MAC algorithms

<KeyText> The key in text form.

The fingerprint value is

```
Fingerprint = <Version-ID> +
              MAC (<Key>, <Content-ID> + ?:? + H(<Data>))
```

Figure 8

Where the value is calculated as follows:

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

Figure 9

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 RBQ26-MEZGP-4SVCU-RYOWO-QTURA.

```
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 =
  52 42 51 32  36 2D 4D 45  5A 47 50 2D  34 53 56 43
  55 2D 52 59  4F 57 4F 2D  51 54 55 52  41
```

Figure 10

Processing is performed in the same manner as an unkeyed fingerprint:

```
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 0B FA BC  7D AB 3C EF  4F 13 3D 3F  BC D8 CE 89
  CC A2 89 10  F0 93 D4 44  D0 45 EA 23  AB AB C0 8E
  9D 6F CB EB  37 EC EA DB  B6 04 B3 1F  61 02 3B 9A
  B8 29 48 45  36 9D 78 AC  D6 DA 42 36  79 13 E9 51

HKDF(Key) =
  7A 10 08 F5  9F 46 3C FF  09 7F 8E 59  41 FB 9B 22
  28 FF 7E C5  A4 1D 01 11  18 A1 EC A9  DD A4 1D 48
  29 6A B8 C9  98 7C 13 C9  15 74 C4 16  1A AA 6E 94
  09 46 7F F7  88 84 15 A0  85 6F E5 19  82 06 20 58

MAC(<key>, <Content-ID> + ':' + H(<Data>)) =
  87 5F 7C 18  D7 D8 2C E4  CB D6 58 6D  C0 7B 8B DC
  C9 E4 7F 79  0B 7E 3E 13  63 EC 86 C4  AB 36 6D 78
  74 D2 C0 D5  B9 A5 33 AB  EE CA 4A 70  30 45 D9 D6
  63 08 E0 5C  85 1B 1B C9  69 D0 55 6E  8A E0 2C 8D

Prefixed, compressed, trimmed =
  50 87 5F 7C  18 D7 D8 2C  E4 CB D6 58  6D C0 7B 8B
  DC ...
```

Figure 11

The 125 bit fingerprint value is KCDV6-7AY27-MCZZG-L2ZMG-3QD3R

5. Content Types

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 very different.

5.1. 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 KeyInfo structure defined in the PKIX certificate specification

5.2. OpenPGP Key

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

application/pgp-key-v5 An OpenPGP key

application/pgp-keys An OpenPGP key set.

5.3. 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. URI Scheme

[RFC6920] .

6.1. Scheme Syntax

6.2. Scheme Semantics

6.3. Encoding considerations

6.4. Interoperability considerations

6.5. Security considerations

7. Additional UDF Renderings

By default, a UDF fingerprint is rendered in the Base32 encoding described in this document. Additional renderings MAY be employed to facilitate entry and/or verification of fingerprint values.

7.1. Machine Readable Rendering

The use of a machine-readable rendering such as a QR Code allows a UDF value to be input directly using a smartphone or other device equipped with a camera.

A QR code fixed to a network capable device might contain the fingerprint of a machine-readable description of the device.

7.2. 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 permits 16. These encodings allow a 125 bit fingerprint to be encoded in 9 and 8 words respectively.

7.3. 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.

8. Security Considerations

8.1. 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.

8.2. 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.

9. IANA Considerations

9.1. URI Registration

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]

9.2. Version Registry

[Here request creation of a version registry for the UDF prefix values]

80 = HMAC and SHA-2-512
81 = HMAC and SHA-3-512
96 = SHA-2-512
97 = SHA-2-512 with 24 leading zeros
98 = SHA-2-512 with 32 leading zeros
99 = SHA-2-512 with 40 leading zeros
100 = SHA-2-512 with 48 leading zeros
101 = SHA-2-512 with 56 leading zeros
136 = Random nonce
144 = SHA-3-512
145 = SHA-3-512 with 24 leading zeros
146 = SHA-3-512 with 32 leading zeros
147 = SHA-3-512 with 40 leading zeros
148 = SHA-3-512 with 48 leading zeros
149 = SHA-3-512 with 56 leading zeros

Figure 12

10. References

10.1. Normative References

[RFC2014] Weinrib, A. and J. Postel, "IRTF Research Group Guidelines and Procedures", BCP 8, RFC 2014, DOI 10.17487/RFC2014, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010.

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

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

10.2. Informative References

[Dobertin95]
Eurocrypt 1996, "Cryptanalysis of MD5 Compress".

[draft-hallambaker-mesh-developer]
Hallam-Baker, P., "Mathematical Mesh: Reference Implementation", draft-hallambaker-mesh-developer-07 (work in progress), April 2018.

[hallambaker-sin]
"[Reference Not Found!]".

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013.

10.3. URIs

[1] http://mathmesh.com/Documents/draft-hallambaker-udf.html

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: phill@hallambaker.com