Coded Hashes of Arbitrary Images

(Revised Preliminary Proposal)

Steven R. Loomis
srl@icu-project.org
(individual contribution)

Keith Winstein
keithw@cs.stanford.edu
Stanford University

Jennifer 8. Lee
jenny@jennifer8lee.com
Emojination

2016-11-08

`https://srl295.github.io`

1 Introduction

This proposal is an update to L2/16-105.[1]

> Emoji are pictographs (pictorial symbols) that are typically presented in a colorful cartoon form and used inline in text. […] In Unicode 8.0, there is a total of 1,282 emoji, which are represented using 1,051 code points.[2]

Recently, there has been considerable interest in adding newly created pictorial symbols, not found in any existing character set, to the Unicode Standard as emoji.[3] Advocacy groups and others request these code points because Unicode plain text remains the dominant interoperable interchange format for messaging. In practice, before a new emoji can be used, a code point must be assigned and be recognized by the sender and receiver. The stated longer-term goal for Unicode is that implementations should support "embedded graphics, in addition to the emoji characters".[4]

In this updated proposal, we describe a mechanism to uniquely identify arbitrary images within a plain-text Unicode character sequence. This will allow implementers to retrieve emoji characters as glyphs from a font without needing to request and wait for the assignment of a code point. The basic idea is to encode a globally-unique *secure hash* of the emoji in a Unicode character sequence. Once the implementation knows the image's hash, it may

[1] Steven Loomis, Keith Winstein, and Jennifer Lee. *L2/16-105: Coded Hashes of Arbitrary Images*. 2016. URL: `http://www.unicode.org/L2/L2016/16105-unicode-image-hash.pdf`.

[2] Mark Davis and Peter Edberg, eds. *Unicode Technical Report 51: Unicode Emoji*. 2015. URL: `http://www.unicode.org/reports/tr51/`.

[3] *E.g.,* Taco Emoji campaign, Beard Emoji campaign, Dumpling Emoji campaign. There are currently 79 candidate emoji that have been assigned tentative code points.

[4] Davis and Edberg, *Unicode Technical Report 51: Unicode Emoji*, op. cit., Section 8, "Longer Term Solutions".

already have the corresponding image, or may have a choice of several mechanisms to retrieve it.

Our technique will gracefully degrade on legacy Unicode implementations, but our proposal is limited to allowing Unicode to *uniquely identify* an arbitrary image.

1.1 Background

The demand for emoji characters has surged in the last few years as public has become enthralled by the multicolor glyphs that can appear in-line with text. Emoji have been the subject of an explosion of press pieces, numerous late-night television comedic segments, an upcoming movie from Sony Pictures Animation, and even a dedicated conference in November 2016. Never has the encoding attracted such attention.

As emoji use has proliferated, there have been increasing demands for better representation in the existing emoji set — ranging from hair variations (color and facial), to more female presence, to single-family households, to regional flags.[5]

However, the current emoji approval process takes upwards of 18 months, starting from when a proposal is first introduced, to when it moves out of the Unicode Emoji Subcommittee, to the Unicode Technical Committee, to ISO and back again for final approval. This current process, overseen by a relatively small organization whose stated main mission is focused on encoding existing languages, is widely understood as a bottleneck and also incredibly resource intensive on members' time.

To fill the demand for customizable glyphs, there has been an explosion of emoji-like images from a variety of vendors, ranging from personalized Bitmoji, to highly publicized "Kimoji", to earnest refugee-themed emoji. These rogue "moji" are not technically emoji at all, but rather images or "stickers" that can be sent through messaging apps. In addition, vendors are using technical workarounds such as the Zero Width Joiner (ZWJ) and tagging to create emoji images that do not have to be assigned code points.

Meanwhile, the variants of single emoji created by different vendors leads to confusion in the intent of the sender and recipient.[6] The inconsistent imagery is at tension with Unicode's goal of having interoperable platform-neutral communication. A longer term architectural solution is needed.

2 Proposal

Outline: implementations will be able to create "implementation-defined emoji," and allow their users to send them to receivers. The sender encodes a secure hash of the emoji (which

[5] Mark Davis and Peter Edberg, eds. *Unicode Technical Report 52: Unicode Emoji Mechanisms*. 2016. URL: `http://www.unicode.org/reports/tr52/`.

[6] Hannah Miller et al. "'Blissfully happy' or 'ready to fight': Varying Interpretations of Emoji". In: *ICWSM '16* (2016). URL: `http://grouplens.org/site-content/uploads/ICWSM16_Emoji-Final_Version.pdf`.

serves as an unforgeable globally unique identifier) in a sequence of coded characters, which we call a Coded Hash of an Arbitrary Image (CHAI).

New emoji can then be created without (1) needing permission from any other party, (2) getting a code point assigned from any central registry (such as Unicode or ISO 10646) (3) claiming an unknown/unassigned ZWJ sequence or (4) waiting for the publication of Unicode, UCS, or even updated data files from the UTC Emoji SubCommittee. Implementers could update the repertoire of emoji as frequently or infrequently as they wish.

2.1 Representing the emoji

The first step to creating an implementation-defined emoji will be to represent the emoji in a canonical form. This format will need to be specified, but we do not do so here. As a strawman, we suggest a JSON document with three keys: `content-type` (any IANA-registered MIME Media Type), `image` (a Base64-encoded suggested rendering of the emoji, interpreted as the format named in the `content-type` field), and `name` (name of the emoji, which may be used in fallback rendering for visually-impaired users). We call this representation the "emoji description."

2.2 Secure hash

The implementation will identify the emoji by taking the SHA-256 hash of the emoji description. Given a secure hash value, it is intended to be intractable to find a different input that produces the same hash value. This is known as "second-preimage resistance"; SHA-256 is believed to have strong second-preimage resistance.

2.3 Code points allocated to express the secure hash

The prior proposal proposed to allocate 65,536 (2^{16}) code points to represent bits of the secure hash, resulting in a 50% efficiency in each Unicode encoding scheme (16 bits of the hash will consume 32 bits in memory, on disk, or on the wire).[7] This current proposal has a reduced encoding "footprint", meaning that the coded hash will take more bits on the wire. The general mechanism remains similar to the prior proposal.

2.4 Encoding an implementation-defined emoji

To code a CHAI, the sender first encodes a "fallback" base character that most closely approximates the implementation-defined emoji. This will allow rendering to degrade gracefully on receivers that do not support CHAIs.

Next, the special tag *U+E0002 TAG CODED HASH MODIFIER* is sent.

Next, the sender encodes the secure hash by appending between between multiple TAG characters (chosing from 0-9 and a-f). This will represent a prefix of the SHA-256 hash of

[7] Loomis, Winstein, and Lee, *L2/16-105: Coded Hashes of Arbitrary Images*, op. cit.

the emoji description. The sender chooses how many bits of the hash to include. 128 bits, or 32 TAG characters, is the minimum required. The risk of including too few bits is that the hash may no longer be globally unique, especially in the presence of an attacker who wishes to create a different image with the same hash prefix.

Therefore, the actual encoding is:

<basechar> + <U+E0002 TAG CODED HASH MODIFIER> + < TAG [0-9a-f] > +< TAG [0-9a-f] > +< TAG [0-9a-f] > + …<U+E007F CANCEL TAG>

where the *basechar* represents a fallback base character, and each TAG [0-9a-f] character (from the range U+E0030 .. U+E0039 and U+E0061 .. U+E0066) represents 4 bits of the secure hash of the emoji description.

2.5 Receiving and rendering a CHAI

On receipt, the receiver determines if it already knows of an emoji whose hash matches the prefix encoded in the CHAI characters. If so, it displays the emoji (either using the image provided in the emoji description, or an image known locally). Otherwise, the receiver displays the base character.

Because the CHAI serves as an unforgeable globally unique identifier for the emoji description, the receiver is intended to have confidence that if it finds a matching emoji description, it can display it per the sender's intent.

3 Example

3.1 chai.json

Example Canonical Emoji Description


```
{"name":"CHAI","content-type":"image/png","image":"iVBORwOKGgoAAAANSUhEUgAAAEAAAABACAYAAACqaXHeAAACTOlEQVR42u2ZMUvDQBiGS4YM4tChU/EHiFMRp45OpTg4dBR/
      gENnp0IH8QdIR21TTdfqoOBQRRAEG8VNOnVy6iTFsX7epReJIU2rufQu8D7wUS5pL+/39
      vL17pLJAAAAAAAAAAAAYXyyyKZEa1bolQqxqKbEgKrQK92AQUoMGCRhwGMSnSYECb1S2UuZAXuyOltjcSM65ZHXPPm8OGnK6KzmS9wzoa65AXVZI7UWkjBvjzQ3YCTDgNqMf/
      sqBXWAhE6pyRfEsXFKDNiRlXzBVwN4bPJPcizZMWbxzuKNntt96ls9clpdctoW9VsN9nnM2ofs3AFr75NztksvrW16am6R01ynVytPD6erpmluCJ1GHPeOAm0SRvwcMwzjhF/
      QvTAXwIVwQRedD7q2u65QLpgLnyZguQnxxHiCPNFpwmOZRn71WxR3hPIflwPtsO9MQkXcnhFddhYT3O38juB53k/v/
      E8GcFOyDGjMMcBO6DaINnMBc4VeO64BwzkGFJdiwD9GhNBbjGPAMJAORRhVXLoJOckXZTyhGiEGDEOqqjvUNDPAlmFAOdBJyfckKPmOT5TcBvOH/
      ySuASuiozvfMcN3a3hrgqamBpzImElVfMnmZqwKvRmhoUnyhtCzLms6mfOZUImacmpiwE5SU/T7wFTYK4pesbzRxIBEF2krojg2ArWAdKkDqpbpdc0MWPpGzc/Wk+
      LkTZVbde7mo2IDlG7WutvPig1Qul0/UF0HVL+wqWpigLJXdtnM9CUkKY5sBgAAAAAAAAAAAAAAAP7ANw53SLcSCEY2AAAAAElFTkSuQmCC"}
```


3.2 chai.png

Sample PNG

3.3 chai.txt

Sample encoding

```
# Writing chai.json
# SHA-256 hash:
#  37d8c5403d29ec7d6f59b02690414de77b059773
#  5a953c2a00fbea50b5f79bb5
# Going to use 32 CHAI chars or 128 bits.
# Hash String: 37d8c5403d29ec7d6f59b02690414de7
# <U+2615> (base)
# <U+E0002 TAG CODED HASH MODIFIER>
# <U+E0033 TAG DIGIT 3>                    ;3
# <U+E0037 TAG DIGIT 7>                    ;7
# <U+E0064 TAG LATIN SMALL LETTER D>       ;d
# <U+E0038 TAG DIGIT 8>                    ;8
# <U+E0063 TAG LATIN SMALL LETTER C>       ;c
# <U+E0035 TAG DIGIT 5>                    ;5
# <U+E0034 TAG DIGIT 4>                    ;4
# <U+E0030 TAG DIGIT 0>                    ;0
# <U+E0033 TAG DIGIT 3>                    ;3
# <U+E0064 TAG LATIN SMALL LETTER D>       ;d
# <U+E0032 TAG DIGIT 2>                    ;2
# <U+E0039 TAG DIGIT 9>                    ;9
# <U+E0065 TAG LATIN SMALL LETTER E>       ;e
# <U+E0063 TAG LATIN SMALL LETTER C>       ;c
# <U+E0037 TAG DIGIT 7>                    ;7
# <U+E0064 TAG LATIN SMALL LETTER D>       ;d
# <U+E0036 TAG DIGIT 6>                    ;6
# <U+E0066 TAG LATIN SMALL LETTER F>       ;f
# <U+E0035 TAG DIGIT 5>                    ;5
# <U+E0039 TAG DIGIT 9>                    ;9
# <U+E0062 TAG LATIN SMALL LETTER B>       ;b
# <U+E0030 TAG DIGIT 0>                    ;0
# <U+E0032 TAG DIGIT 2>                    ;2
# <U+E0036 TAG DIGIT 6>                    ;6
# <U+E0039 TAG DIGIT 9>                    ;9
# <U+E0030 TAG DIGIT 0>                    ;0
# <U+E0034 TAG DIGIT 4>                    ;4
# <U+E0031 TAG DIGIT 1>                    ;1
# <U+E0034 TAG DIGIT 4>                    ;4
# <U+E0064 TAG LATIN SMALL LETTER D>       ;d
# <U+E0065 TAG LATIN SMALL LETTER E>       ;e
# <U+E0037 TAG DIGIT 7>                    ;7
# <U+E007F CANCEL TAG>
```

The resulting hash matches the first 80 bits of

37d8c5403d29ec7d6f59b02690414de77b0597735a953c2a00fbea50b5f79bb5

4 Character Data

This proposal requests a total of 1 new character to be encoded.

4.1 Character Properties

```
E0002;TAG CODED HASH MODIFIER;Cf;0;BN;;;;;N;;;;;
```

Table 1: UTS52 Hash Sequence

tag_key	tag-H
tag_base	Any character with Emoji=Yes
tag_value	[\xE0030-\xE0039\xE0061-\xE0066]

5 Alternate structure — UTS 52

As of this writing, there is a proposal on the UTC agenda to re-activate the UTS 52 tag mechanism.[8] An alternative, which would not require any additional characters encoded, would be to make use of the UTS 52 tag mechanism.

A possible alternate structure under the UTS 52 model is given in Table 1. The net effect is replacing the proposed new character `U+E0002 TAG CODED HASH MODIFIER` with `U+E0048 TAG CAPITAL LETTER H`.

The above example could be re-encoded as:
`<U+2615>`H37d8c5403d29ec7d6f59b02690414de7`<U+E007F>`
(where the H is tag-H and 0-9a-f are the corresponding tag characters)

References

Davis, Mark and Peter Edberg. *L2/16-226: Reactivate UTS 52 mechanism in reduced form*. 2016. URL: http://www.unicode.org/cgi-bin/GetMatchingDocs.pl?L2/16-226.

— eds. *Unicode Technical Report 51: Unicode Emoji*. 2015. URL: http://www.unicode.org/reports/tr51/.

— eds. *Unicode Technical Report 52: Unicode Emoji Mechanisms*. 2016. URL: http://www.unicode.org/reports/tr52/.

Loomis, Steven, Keith Winstein, and Jennifer Lee. *L2/16-105: Coded Hashes of Arbitrary Images*. 2016. URL: http://www.unicode.org/L2/L2016/16105-unicode-image-hash.pdf.

Miller, Hannah et al. "'Blissfully happy' or 'ready to fight': Varying Interpretations of Emoji". In: *ICWSM '16* (2016). URL: http://grouplens.org/site-content/uploads/ICWSM16_Emoji-Final_Version.pdf.

Colophon

Repo URL: https://github.com/srl295/srl-unicode-proposals

Typeset by LaTeX. Made with 100% recycled bits. All opinions belong to the authors and do not reflect the opinions of their associated employers. Thank you to the Computer History Museum for hosting the "Emoji (and more)" event, where this proposal was originally penned.

[8] Mark Davis and Peter Edberg. *L2/16-226: Reactivate UTS 52 mechanism in reduced form*. 2016. URL: http://www.unicode.org/cgi-bin/GetMatchingDocs.pl?L2/16-226.