Coded Hashes of Arbitrary Images (revised)

(or: the last frontier of emoji encoding)

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

Keith Winstein
keithw@cs.stanford.edu
Stanford University

Jennifer 8. Lee
jenny@jennifer8lee.com
Emojination

2016-10-draft
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 create their own emoji 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.

[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".

Once the receiver knows the image's hash, it may 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. We propose to leave to implementers the details of how to retrieve the actual 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, and even an upcoming movie from Sony Pictures Animation. 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 serves as an unforgeable globally unique identifier) in a sequence of coded characters, 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`.

we call a Coded Hash of an Arbitrary Image (CHAI). Upon receiving the CHAI sequence, the receiving implementation may already know the corresponding image, or may have to request it either from the sender or from a third party.

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 straw-man, 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 goals and much of the mechanism remains the same.

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 IMAGE HASH TAG* 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 the emoji description. The sender chooses how many bits of the hash to include. (We expect 80 bits, or 20 TAG characters, to be sufficient for typical use today. 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.)

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

Therefore, the actual encoding is:

<basechar> + <U+E0002 IMAGE HASH TAG> + < 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 while it attempts to retrieve an emoji description whose hash matches the encoded hash prefix.

We do not specify how this retrieval will take place; we expect it to vary based on the messaging protocol. In some protocols, it might be easiest for the receiver to simply ask the sender to send the full emoji description. In others, the receiver might consult an online repository run by the implementing vendor. Alternately, the receiver might consult a community repository where anybody can contribute any emoji description.

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 from anywhere, it can display it per the sender's intent.

Depending on the protocol, as the input arrives, the receiver may have some ambiguity about when the sequence of CHAI characters ends. A receiver may choose to wait until the next non-combining character (signaling the end of the combining character sequence), or a protocol-defined end-of-message signal, before retrieving the emoji description.

2.6 Privacy

As Unicode Technical Report #51 notes: "There are also privacy aspects to implementations of embedded graphics: if the graphic itself is not packaged with the text, but instead is just a reference to an image on a server, then that server could track usage."[8]

We agree. A similar exposure could occur if the protocol provides for the receiver to ask the sender to send an unknown emoji description; this could unwittingly expose if anybody *else* has ever sent the same emoji to the receiver.

It is not the goal of this document to address all aspects of embedded graphics, and we intend to leave to domain experts and implementers the decision on how to handle these questions. For example, a client might behave similarly to current practice in retrieving external images in email. If the receiver already knows of the identified emoji, it can display

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

it, but otherwise the receiver would display the fallback (base) character and ask the user to push a button before retrieving the emoji description.

3 Example

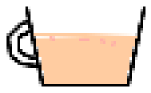
3.1 chai.json

Example Canonical Emoji Description

```
{"name":"CHAI","content-type":"image/png","image":"iVBORw0KGgoAAAANSUhEUgAAAEAAAABACAYAAACqaXHeAAACT0lEQVR42u2ZMUvDQBiGS4YM4tChU/EHiFMRp45OpTg4dBR/
        gENnp0IH8QdIR2lTTdfqoOBQRRAEG8VNOnVy6iTFsX7epReJIU2rufQu8D7wUS5pL+/39
        vLl7pLJAAAAAAAAAAAAYXyyyKZEa1bolQqxqKbEgKrQK92AQUoMGCRhwGMSnSYECb1S2UuZAXuyOltjcSM65ZHXPPm80GnK6KzmS9wzoa65AXVZI7UWkjBvjzQ3YCTDgNqMf/
        sqBXWAhE6pyRfEsXFKDNiRlXzBVwN4bPJPcizZMWbxzuKNntt96ls9clpdctoW9VsN9nnM2ofs3AFr75NztksvrW16am6R01ynVytPD6erpmluCJ1GHPeOAmOSRvwcMwzjhF/
        QvTAXwIVwQRedD7q2u65QLpgLnyZguQnxxHiCPNFpwmOZRn71WxR3hPIflwPtsO9MQkXcnhFddhYT3O38juB53k/v/
        E8GcFOyDGjMMcBO6DaINnMBc4VeO64BwzkGFJdiwD9GhNBbjGPAMJA0RRhVXLoJ0ckXZTyhGiEGDEOqqjvUNDPAlmFAOdBJyfckKPmOT5TcBvOH/
        ySuASuiozvfMcN3a3hrgqamBpzImElVfMnmZqwKvRmhoUnyhtCzLms6mfOZUImacmpiwE5SU/T7wFTYK4pesbzRxIBEF2krojg2ArWAdKkDqpbpdcOMWPpGzc/Wk+
        LkTZVbde7mo2IDlG7WutvPig1Qul0/UF0HVL+wqWpigLJXdtnM9CUkKY5sBgAAAAAAAAAAAAAAP7ANw53SLcSCEY2AAAAAElFTkSuQmCC"}
```

3.2 chai.png

Sample PNG



Sample encoding

```
# Read chai.png, wrote chai.json
# SHA-256 hash:
 37d8c5403d29ec7d6f59b02690414de77b0597735a953c2a00fbea50b5f79bb5
# Going to use 20 CHAI chars or 80 bits.
# <U+2615> (base)
# <U+E0002 IMAGE HASH TAG>
# <U+E0033              TAG DIGIT 3>
# <U+E0037              TAG DIGIT 7>
# <U+E0037 TAG LATIN SMALL LETTER d>
# <U+E0038              TAG DIGIT 8>
# <U+E0038 TAG LATIN SMALL LETTER c>
# <U+E0035              TAG DIGIT 5>
# <U+E0034              TAG DIGIT 4>
# <U+E0030              TAG DIGIT 0>
# <U+E0033              TAG DIGIT 3>
# <U+E0033 TAG LATIN SMALL LETTER d>
# <U+E0032              TAG DIGIT 2>
# <U+E0039              TAG DIGIT 9>
# <U+E0039 TAG LATIN SMALL LETTER e>
# <U+E0039 TAG LATIN SMALL LETTER c>
# <U+E0037              TAG DIGIT 7>
# <U+E0037 TAG LATIN SMALL LETTER d>
# <U+E0036                 TAG DIGIT 6>
# <U+E0036 TAG LATIN SMALL LETTER f>
# <U+E0035                 TAG DIGIT 5>
# <U+E0039                 TAG DIGIT 9>
# <U+E007 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;IMAGE HASH TAG;Cf;0;BN;;;;;N;;;;;
```

References

Davis, Mark and Peter Edberg, 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.