Unicode Data Publication Security

(Preliminary Proposal)

Steven R. Loomis — srloomis@us.ibm.com
https://srl295.github.io

2017-10-23

1 Introduction

Trust in electronic data is a common and persistent concern. Unicode data, such as the UCD, is not exempt from these concerns. Unicode properties are used in many security-related applications.[1] This document attempts to introduce some steps which can be taken by the UTC to allow consumers of our data files to verify the integrity of downloaded files, making use of tools such as GPG.[2]

It is an explicit non-goal of this document to introduce change simply for the sake of change. The impact on existing processes, particularly on the UCD generation and release process, and the editorial committee's workload, must be considered.

2 Background

Consumers of data need to be able to verify that (1) a data file has been produced by a trusted source, and secondly that (2) the data file has not been tampered with, whether unintentionally (such as via human or network error) or maliciously. To accomplish goal (1), the *identity* of the author needs to be proved. To accomplish goal (2), the *integrity* of the data must be proved.

Examples of projects proving the identity and integrity of downloadable content include Apache,[3] ICU,[4] and Node.js.[5]

[1] *Creative usernames and Spotify account hijacking.* URL: `https://labs.spotify.com/2013/06/18/creative-usernames/`.

[2] *GnuPG - Support.* URL: `https://www.gnupg.org/documentation/`.

[3] *Apache Release Signing.* URL: `https://www.apache.org/dev/release-signing.html`.

[4] *Verifying Downloads - ICU - International Components for Unicode.* URL: `https://sites.google.com/site/icusite/download/verification`.

[5] *Node.js README.* URL: `https://github.com/nodejs/node/blob/master/README.md#verifying-binaries`.

Please see the Apache[6] document in particular for detailed explanations of the terms and tools used in this document.[7]

3 Process Overview

1. GPG public keys should be used for the purpose of signing Unicode data files. These can be created if needed, or existing ones used. These keys should be kept reasonably safe and up to date.

2. For cases where we have many individual text files, it would be unwieldy to add a signature file for each text file. It would be better to use a single `SHASUM` file containing all hash values, and then one single gpg signature on the `SHASUM` file.

3. Signing of data files is *not* necessary for pre-release data. However, the `SHASUM` file can remain valuable to ensure the integrity of the downloaded files, at least at the beta level.

4. Ideally, data files would be signed by the exact maintainers who generated, edited, and/or uploaded the files. However, to prevent adding undue burden, a simpler process involves adding the signatures after uploading, and then requesting the original maintainers to verify after the fact that the correct files were signed.

[6] *Apache Release Signing*, op. cit.

[7] This version of the document does not repeat generic information and instructions about how to use GPG and other tools, beyond the specifics of how to apply them to Unicode data. This author is willing to expand either this document, or other relevant documentation, as needed to support the UTC's own processes and guidance to consumers downloading the data.

4 Recommendation

4.1 General

1. It is recommended to use encryption in processes where possible: ssh instead of telnet, TLS-encrypted protocols instead of unencrypted smtp, pop, imap, ftp, and http. `https://unicode.org` itself has recently upgraded to use (and prefer) https over http.

2. The appropriate documentation should be updated to give consumers of data information on how to verify their downloaded data.

4.2 Key Management

1. There should be a `KEYS` file for Unicode, perhaps located at `https://unicode.org/KEYS`.

2. This file should contain all public GPG keys used to sign Unicode data files. Ideally, the keys would be cross-signed with each other to produce a web of trust.

3. A key might be created with `gpg --gen-key`

4. The `KEYS` file might be updated as follows:

```
\$ (gpg --list-sigs 8BBE8BC2 ; gpg --armor --export 8BBE8BC2) | tee -a KEYS
pub   1024R/8BBE8BC2 2017-10-26 [expires: 2022-10-25]
uid                  Some Unicoder <unicoder@unicode.example.org>
…
-----BEGIN PGP PUBLIC KEY BLOCK-----

mI0EWfJaUgEEAP0HBJt0HNdV/lle0ADYWopZKjxMo/d8J43br4G8mCFaNtx7V9iU
    …lots of stuff omitted…
rl/kiGzHxXIo6A0t2Tx0OE5URA==
=HmNb
-----END PGP PUBLIC KEY BLOCK-----
```

5. Unicode officers should authorize any changes to the `KEYS` file.

4.3 Zipped and other Binary Files

This section deals with signing for `.zip` files such as in the `zipped/` or `ucdxml/` directories, and *could* be applied to other binary files such as chart `.pdf`s.

For zipped files, such as in `https://www.unicode.org/Public/zipped/10.0.0/` we currently have: `ReadMe.txt`, `UCD.zip`, `Unihan.zip`.

1. A SHASUM file should be created. (The GPG signatures also verify the integrity and authorship of the files, however, the SHASUM can be validated even in the absence of the GPG software or key validation.)

   Note: the output is very wide, so ellipses are used.

   ```
   \$ shasum -a 512 * | tee SHASUM512.txt
   c6def2…ea6  ReadMe.txt
   4e232…eee0  UCD.zip
   377f53…b12  Unihan.zip
   ```

2. Next, a *detached signature* is created for each file, including the SHASUM512.txt. (Skip this for pre-release content.)

   ```
   $ for file in SHASUM512.txt *.zip; do \
   gpg -u 8BBE8BC2 --armor --output ${file}.asc --detach-sig ${file}; \
   done
   ```

3. As a result, the following files are now created in the above directory:

   ReadMe.txt SHASUM512.txt SHASUM512.txt.asc UCD.zip UCD.zip.asc Unihan.zip Unihan.zip.asc

4. Users can recalculate the SHA512 sums and compare against SHSUM512.txt. Users can import the keys from the KEYS file and verify any of the signatures (.asc files) with gpg --verify UCD.zip.asc.

4.4 Plain Text Files

It would be unwieldy to have a separate signed file for each of the nearly 100 text files that makes up a UCD release such as found in https://www.unicode.org/Public/10.0.0/. Instead, a single SHASUM file is generated.

1. Generate a single SHASUM512.txt file in the 10.0.0 directory.

   ```
   $ shasum -a 512 $(find charts ucd ucdxml -type f) | tee SHASUM512.txt
   …1d19cccf26bed24f07d8979fe2f4ea93fb88d8b charts/CodeCharts.pdf
   …84579cb3cd9f2d08c21f87153aacae49bcc4ece ucd/ArabicShaping.txt
   …da3f7cc592514b970716492b60b861af9e3b847 ucd/auxiliary/GraphemeBreakProperty.txt
   …cc6e944f9076c6869d481a3b9550228e59521b8 ucd/auxiliary/GraphemeBreakTest.html
   …8e9bbee82bbf0f9125c1f0017e0c9f791ea4526 ucd/auxiliary/GraphemeBreakTest.txt
   …0b91820e089eb139b4444e743a97ba5368f2d3d ucd/auxiliary/LineBreakTest.html
   …c69070676c474f0270bf38f5a78795face74b33 ucd/auxiliary/LineBreakTest.txt
   …8b1c9a88979bc8a7d65f2abf1e72a943aaa6191 ucd/auxiliary/SentenceBreakProperty.txt
   …(many more)…
   ```

2. Now, GPG sign the SHASUM.

```
$ gpg -u 8BBE8BC2 --armor --output SHASUM512.txt.asc --detach-sig SHASUM512.txt
```

3. We now have `SHASUM512.txt` and `SHASUM512.txt.asc` its signature file.

References

Apache Release Signing. URL: https://www.apache.org/dev/release-signing.html.

Creative usernames and Spotify account hijacking. URL: https://labs.spotify.com/2013/06/18/creative-usernames/.

GnuPG - Support. URL: https://www.gnupg.org/documentation/.

Node.js README. URL: https://github.com/nodejs/node/blob/master/README.md#verifying-binaries.

Verifying Downloads - ICU - International Components for Unicode. URL: https://sites.google.com/site/icusite/download/verification.

Colophon

Repo URL: https://github.com/srl295/srl-unicode-proposals Typeset by LaTeX. Made with 100% recycled bits.

My GPG fingerprint is `BA90 283A 60D6 7BA0 DD91 0A89 3932 080F 4FB4 19E3`