http://doi.org/10.7207/twgn20-12

Which checksum algorithm should I use?

Matthew Addis

DPC Technology Watch Guidance Note

December 2020

DigitalPreservationCoalition

1 Introduction

This report is intended to help answer one of the perennial questions in digital preservation: what checksum algorithm should I use?

A **checksum** (DPC, 2020) is a digital fingerprint that can be made from a sequence of bytes, otherwise known as a bitstream. The most common example of a bitstream is the contents of a file. Checksums are typically created for entire files, but they can also be made at a more granular level such as for the individual frames in a video, for data that is recorded in a database, or for data stored as an object in the cloud. Just like a fingerprint, a checksum is unique to the bitstream. Any change to the bitstream, however big or small, will cause the value of its checksum to change completely. For example, checksums can be used to detect changes in the contents of a file, or to compare two or more files to see if they have the same or different contents.

A **checksum algorithm**, for example MD5 or SHA512, is used to generate a checksum from a bitstream. The checksum can then be recorded and used in the future to see if the bitstream has changed in any way. This is done by generating a new checksum at some time in the future or when the bitstream has been transferred between locations. The new checksum is compared with the original checksum. If the checksums are the same, then the bitstream hasn't changed. If the checksums are different, then the bitstream has become corrupted or altered in some way. This process of generating and comparing checksums is called **fixity checking** (DPC, 2020).

The use of checksums for fixity checking and data integrity is part of **digital preservation good practice**, for example as described in the DPC RAM (DPC, 2020), NDSA preservation levels (NDSA, 2020), digital preservation storage criteria (Schaefer, McGovern, Goethals, Zierau & Truman, 2020) and CoreTrustSeal (DSA–WDS Partnership Working Group, 2019), to name but a few.

The choice of which checksum algorithm(s) to use depends on the purpose of fixity checking. For example, is the purpose to detect accidental corruption of data when it is stored or transferred? Is the purpose to help protect against malicious data tampering? Or is the purpose to help identify and manage duplicates in your archive?

Checksums can be used to identify if a file has been changed, or whether two files are the same or different, but checksums do not tell you (a) what was changed, (b) when it was changed, (c) who changed it or (d) how to reverse or repair any changes or damage to files. Therefore, checksums are only one part of protecting against data loss or data corruption. Other measures will also be needed, for example information security, logging and audit trails, safe storage of multiple copies of data accompanied by fixity checking and repair, and processes and procedures for backup and disaster recovery.

2 Why do I need to use checksums?

Checksums can have many uses. The choice of checksum algorithm depends on the use that checksums are being put to. Uses of checksums include:

- Detection of data corruption or loss when data is stored, for example when keeping data on disks, tapes, USB drives or in the cloud.
- Detection of data corruption or loss when data is transferred, for example when sending files over the internet or uploading/downloading them from a server.
- Detection of deliberate and malicious attempts to alter the contents of data, for example detecting if someone has deliberately tried to tamper with a document.

- Detection of identical copies of the same data in different files or objects, for example looking for the same contents in multiple files that have different names or locations.
- Confirmation that handover of one or more files between people or organisations has been done successfully, for example verification that a set of files has been successfully transferred to an archive by a depositor.
- Making an inventory of data in preparation for future preservation or archiving activities, for example when making a record of the contents of a hard drive before the contents are extracted and ingested into an archive.
- Making strong and long-term assertions on the integrity and authenticity of data, for example as part of guarantees or testaments that data hasn't changed over time.

Checksums help guard against **accidental data corruption or loss**, which can be the result of IT issues such as storage failures, or the result of unintentional errors by people such as accidentally deleting or editing files or not following proper processes.

Checksums help guard against **deliberate data tampering or deletion**, which can be the result of cyber-attacks or deliberate attempts by individuals to alter data without detection.

Checksums help when making **bitstream level inventories and evidence**, in effect 'knowing and proving what you have', which can be put to use in applications such as selection, evidential weight and bitstream deduplication.

Before considering what checksum algorithm and tools best meet your needs, it is important to understand the reasons you have for using checksums.

3 Which checksum algorithms should I use?

There are many factors that determine which checksum algorithm(s) to use (NDSA, 2014). These include:

- What is the purpose of the checksums (see above)?
- What tools are available for generating and checking checksums, for example are tools available with easy-to-use GUIs or command lines?
- Are the checksums portable, for example can they be used in multiple tools or can they be easily shared with and used by others?
- How quickly can checksums be generated and checked, for example how much computing power is needed to check a large collection of files?

In the case of detecting accidental data corruption and loss, for example when storing data in IT systems or transferring data from one location to another, there are a large number of algorithms that are suitable. A checksum algorithm in this scenario only needs to be 'good enough' to detect unintentional changes to the data. For example, MD5 is perfectly suitable - it is a very widely adopted, there is good tool support, and checksums are quick to generate and compare.

In the case of detecting malicious tampering of data, or when checksums are used to unambiguously locate multiple files that have exactly the same contents, then 'strong cryptographic hash algorithms' will typically be needed, for example SHA256 or SHA512. These algorithms are commonly used in related technologies such as digital signatures and deduplication tools. This is also an area where new algorithms tend to emerge, for example Elliptic Curve Cryptography (Wikipedia, 2020). In a digital preservation context, these newer algorithms are typically unnecessary unless there are very stringent security requirements.

MD5 is only suitable for detecting accidental data corruption and not for data security applications. Strong cryptographic hash algorithms such as SHA256 and SHA512 can be used for both data security and data safety. This makes SHA256 and SHA512 more universally usable – but at the expense of being more computationally expensive to calculate and somewhat less widely supported by some tools (esp. SHA512).

In some scenarios, in particular when processing very large volumes of data, the speed and computational overhead of checksum calculation can become a factor in the choice of checksum algorithm. Within the audiovisual domain, where data volumes can be huge, the use of xxhash (xxHash, 2021) is becoming increasingly popular and is used by several organisations, equipment manufacturers and service providers. At the time of writing, the use of xxhash is not yet widespread in the digital preservation community, but it is worth considering as a very fast but non-cryptographically strong algorithm for detecting accidental data corruption in very large datasets.

If all you are concerned with is detecting accidental data corruption in short-term storage or in transit between two locations, then MD5 or Adler32 are just fine and both widely supported and used. Don't be put off by articles that say 'MD5 is broken' or that tell you not to use MD5 under any circumstances. However, if you want to use checksums for additional applications, for example checking for duplicate files inside a large archive (Wikipedia, 2020), or for proving the integrity and authenticity of archived files in adversarial conditions (Rosenthal, 2017), then go for something that is cryptographically strong. SHA512 is currently a good bet.

It is possible, and often sensible, to use multiple algorithms. For example, you might compute both MD5 and SHA256 for your files. An example of this can be seen in the DROID download page where both SHA256 and MD5 checksums are available for the DROID software distribution. This can be helpful when exchanging checksums with other people or tools where not everyone or every application supports the same algorithm. The use of multiple algorithms can help achieve an unbroken and end-to-end chain of custody between depositors, archive systems, and end-users. Multiple algorithms also allow quick fixity checks to be performed using say MD5, for example when copying files to a new location, whilst also having something stronger such as SHA512 for specific circumstances, for example evidential weight.

The digital preservation world is headed in the direction of stronger algorithms. For example, the latest version (v1.0) of the Bagit specification (Library of Congress, 2018) mandates support for SHA256 and SHA512 whereas support for MD5 is now only optional.

4 What tools can help create checksums and perform fixity checks?

There are many tools available for generating and checking checksums, with some listed on COPTR (COPTR, 2020). When considering tools, some questions to consider include:

- Do I need a Graphical User Interface (GUI) or am I comfortable with the command line?
- What platforms do I need to run the tool on, for example Windows, Mac or Linux?
- Do I need support, or am I confident in my abilities to use community open-source tools?
- Am I allowed to install tools on my systems, or will my IT department need to help me?
- Do I need to generate and check checksums in large batches with automation?
- Do I need to import/export/share checksums with others in a standardised way?

If you want a GUI, then there are many freely available tools for different operating systems, but no single tool that is universally and widely used across all platforms. One tool to consider is DROID (The

National Archives, 2020) which can be used to generate MD5 and SHA256 checksums as well as having other useful features such as file format identification.

If you are comfortable with the command-line, then simple tools such as md5sum on Linux get the job done for checksumming files. There are equivalents on Windows and Mac. These tools produce portable checksum manifests (lists of checksums and files) that can be used in a wide range of other tools. The British Library's Minimal Preservation Toolkit (British Library, 2020) goes further and provides a range of checksum generation and validation options such as checksum trees. Command-line tools offer more flexibility than GUI tools, especially when run on servers. They are easier to automate and integrate into fixity workflows.

If you want to generate checksums for a specific set of files/folders, then tools that support the bagit specification can be useful, for example GUI tools include Fixity (AVP, 2020) and Bagger (Library of Congress, 2020). Python bagit (Library of Congress, 2020) is a good choice if you are happy with the command-line. Bagit is great when transferring a bundle of files and checksums between organisations or locations and you want to know that nothing has been lost or corrupted in the process.

For some content types, for example audiovisual data, there may be specialist tools available for generating checksums for parts of the file (ffmpeg supports checksums for each frame in a video stream (FFmpeg, 2020)). This can be useful, especially for large files, because if corruption occurs then it tells you what was corrupted and can allow the rest of the file to be recovered and used.

It is important to consider where checksum tools will be installed and executed, especially if your data is on remote servers or in cloud storage. In order to generate or check a checksum for a file, then the whole file has to be read by the tool. This means transferring the file data to the location where the tool is running. For example, if you run the tool on your desktop and the files you want to check are on a network drive, then this will mean that the tool will read all the files over the network. This can slow down checking and use a lot of bandwidth. If you are downloading files from cloud storage, then this can incur data retrieval and data egress charges. In these situations, it can be better to run the tool close to the data, for example on a network server or in the cloud. This may require a different choice of tool, or it may mean IT support is needed to install and run the tool in a remote location.

5 Where should I keep checksums?

Checksums are relatively small and easy to store. For example, an MD5 checksum is only 32 characters long in ASCII format irrespective of the size of the bitstream it represents (which could be GBs or even TBs in size). Therefore, checksums can be stored in many ways, including in log files, in databases, in spreadsheets, in manifest files, in inventories (OCFL, 2020), or as part of preservation metadata using standards such as PREMIS (Library of Congress, 2020) within archive packages (DILCIS Board, 2020). Checksum manifests are a simple way to record checksums and can be produced/consumed by many tools. A manifest is a list of checksums and the files that they belong to. There isn't a formal standard for checksum manifests, but most tools follow the format of a text-based manifest where there is a line for each file with the checksum followed by whitespace and then the file name.

For example, a SHA256 manifest for three files might look like this:

```
e09090bece6bfc39e06ebaa69c065291de22fb2741e142f549a87073432a0b40 file1.doc
5b3de41ea284742322cc53ff0490185b04dd3e06401d41baad8d984a59a13c36 file2.jpg
908e01171abbad352e17263b702b5a403063070d85852a8858b91349ecc9c70d file3.mp3
```

It is advisable to store more than one copy of your checksums and to keep them in different locations. Put simply, just like your data, you should keep your checksums safe and secure. For example, you might store one copy of your checksums in manifest files alongside your data so that the checksums are easily accessible and can be used in fixity checking. You might store another copy of the checksums on separate media or in a database to provide a backup. This mitigates the case where your files and checksums are both lost or corrupted at the same time. It is not advisable to put checksums into the name of your files. It is better to store checksums separately/alongside files. The files themselves can then be named using a separate naming convention (DPC, 2018).

If you use bagit for storing files and checksums together, for example archiving content as bags, then bear in mind that if you want to edit any of the files then the whole bag will need to be regenerated. It is also easy to invalidate bags (unless they are zipped) by inspecting the contents using file browsers such as Explorer or Finder that can inject hidden files such as Thumbs.db for images.

6 References

AVP (2020) Fixity available at: https://web.archive.org/web/20201111215157/https://www.weareavp.com/products/fixity/ (Wayback Machine capture from 11th November 2020, accessed 3rd December 2020)

British Library (2020) MPT – GitHub available at: https://web.archive.org/web/20201105215245/https://github.com/britishlibrary/mpt (Wayback Machine capture from 5th November 2020, accessed 3rd December 2020)

COPTR (2020) Category: Fixity available at: https://web.archive.org/web/20201210164733/https://coptr.digipres.org/Category:Fixity (Wayback Machine capture from 10th December 2020, accessed 10th December 2020)

DILCIS Board (2020) Common Specification for Information Packages available at: https://web.archive.org/web/20200924161952/https://earkcsip.dilcis.eu/ (Wayback Machine capture from 24th September 2020, accessed 3rd December 2020)

DPC (2018) File Naming and Formats available at: https://web.archive.org/web/20200927143137/https://www.dpconline.org/docs/knowledge-base/1865-dp-note-4-file-naming-and-formats/file (Wayback Machine capture from 27th September 2020, accessed 10th December 2020)

DPC (2020a) DPC Handbook available at: https://web.archive.org/web/20201202040053/https://www.dpconline.org/handbook/technical-solutions-and-tools/fixity-and-checksums (Wayback Machine capture from 2nd December 2020, accessed 3rd December 2020)

DPC (2020b) Rapid Assessment Model available at: https://web.archive.org/web/20201203132307/https://www.dpconline.org/digipres/dpc-ram (Wayback Machine capture from 3rd December 2020, accessed 3rd December 2020)

DPC (2020c) "What is integrity checking?" available at: https://youtu.be/qx3YQRRKwqw

DSA–WDS Partnership Working Group (2019) CoreTrustSeal Trustworthy Data Repositories Requirements 2020–2022 available at: https://web.archive.org/web/20200303020924if_/https://zenodo.org/record/3638211#.Xl28W7lxdPY (Wayback Machine capture from 3rd March 2020, accessed 3rd December 2020)

FFmpeg (2020) FFmpeg Formats Documentation available at: https://web.archive.org/web/20201112030011/https://ffmpeg.org/ffmpeg-formats.html (Wayback Machine capture from 12th November 2020, accessed 3rd December 2020)

Library of Congress (2018) The BagIt File Packaging Format (V1.0) - ISSN 2070-1721 available at: https://web.archive.org/web/20201019090024/https://tools.ietf.org/html/rfc8493 (Wayback Machine capture from 19th October 2020, accessed 3rd December 2020)

Library of Congress (2020a) bagger – GitHub available at: https://web.archive.org/web/20201202014337/https://github.com/LibraryOfCongress/bagger (Wayback Machine capture from 2nd December 2020, accessed 3rd December 2020)

Library of Congress (2020b) bagit-python – GitHub available at: https://web.archive.org/web/20200917212825/https://github.com/LibraryOfCongress/bagit-python (Wayback Machine capture from 17th September 2020, accessed 3rd December 2020)

Library of Congress (2020c) PREMIS available at: https://web.archive.org/web/20201106020948/https://www.loc.gov/standards/premis/ (Wayback Machine capture from 6th November 2020, accessed 3rd December 2020)

NDSA (2014) "Checking your Digital Content" available at: http://webarchive.loc.gov/all/20180208112216/http://www.digitalpreservation.gov/documents/NDSA-Fixity-Guidance-Report-final100214.pdf (Library of Congress Web Archive from 8th February 2018, accessed 3rd December 2020)

NDSA (2020) Levels of Digital Preservation available at: https://web.archive.org/web/20201203133104/https://ndsa.org/publications/levels-of-digital-preservation/ (Wayback Machine capture from 3rd December 2020, accessed 3rd December 2020)

OCFL (2020) Oxford Common File Layout available at: https://web.archive.org/web/20201101133041/https://ocfl.io/ (Wayback Machine capture from 1st November 2020, accessed 3rd December 2020)

Rosenthal (2017) "SHA1 is dead" available at: https://web.archive.org/web/20200926043723/https://blog.dshr.org/2017/03/sha1-is-dead.html (Wayback Machine capture from 26th September 2020, accessed 3rd December 2020)

Schaefer, McGovern, Goethals, Zierau & Truman (2020) OSF Digital Preservation Storage Criteria available at: https://web.archive.org/web/20201105165535/https://osf.io/sjc6u/ (Wayback Machine capture from 5th November 2020, accessed 3rd December 2020)

The National Archives (2020) DROID: file format identification tool available at: https://web.archive.org/web/20201114001951/https://www.nationalarchives.gov.uk/information-management/manage-information/preserving-digital-records/droid/ (Wayback Machine capture from 14th November 2020, accessed 3rd December 2020)

Wikipedia (2020a) available at: https://web.archive.org/web/20201125202907/https://en.wikipedia.org/wiki/Data_deduplication (Wayback Machine capture from 25th November 2020, accessed 3rd December 2020)

Wikipedia (2020b) available at: https://web.archive.org/web/20201203141959/https://en.wikipedia.org/wiki/Elliptic-curve_cryptography (Wayback Machine capture from 3rd December 2020, accessed 3rd December 2020)

xxHash (2021) available at: GitHub - Cyan4973/xxHash: Extremely fast non-cryptographic hash algorithm (archive.org) (Wayback Machine capture from 24th February 2021, accessed 3rd March 2021)