


CrCheck 5 Guide for End Users

Wording

 CrCheck is used for the program/package CrCheck Multi Platform version 5.xx in general

 crcheck is the renamed program for the platform you want to use it

▼ CrCheck 5 Guide for End Users

- [Wording](#)

- ▼ [Introduction](#)

- **Key Features of CrCheck**
- **Important Note: Incompatibility with CrCheck 4.xx**
- [Use Case/Business Case](#)

- [How CrCheck Works](#)

- ▼ [Getting Started](#)

- [Installation](#)
- [Usage](#)
- [Options](#)
- [Excluded Files](#)
- [CrCheck.txt file](#)
- [Known issues or bugs?](#)

- ▼ [Advanced Usage](#)

- [PGP/GPG Support](#)
- [Batch File Integration](#)
- [Error Levels](#)

- ▼ [Example Output](#)

- **License Terms for CrCheck**
- ▼ **Licensing Overview**
 - **Freeware for Non-Commercial Use**
 - **Commercial Use**
- **Summary of Licensing Model**
- **Registered vs. Unregistered Versions**
- **Intellectual Property and Usage Restrictions**

- [Conclusion](#)

- [History](#)

- [Overview of Hash Algorithms](#)

- ▼ [Hash Algorithms supported by CrCheck](#)

- [1. Adler32 \(32 bit\)](#)
- [2. Castagnoli \(32 bit\)](#)
- [3. CRC32 \(IEEE\) \(32 bit\)](#)
- [4. MD5 \(128 bit\)](#)
- [5. SHA-1 \(SHA-160, 160 bits\)](#)
- [6. SHA-2 \(SHA-256, SHA-512 - 256 or 512 bits\)](#)
- [7. SHA-3 \(256 bit implementation\)](#)
- [8. xxHash \(64 bit\)](#)
- [9. RIPEMD-160 \(160 bit\)](#)
- [10. BLAKE2 and BLAKE3 \(256 bit\)](#)
- [11. BLAKE3-128 \(128 bit\)](#)

- ▼ **Additional Cryptographic Hash Overviews**
 - Whirlpool
 - Tiger
 - Skein
- ▼ [Performance Comparison](#)
 - [Summary Table of Hash Algorithms](#)
 - [Hash Algorithms - Key Points](#)
 - [Performance Ranking by Speed](#)
- [Reference: Empty Hashes](#)
- [End of documentation](#)

Introduction

🧠 **CrCheck** is a versatile and reliable **command-line tool** built to ensure file and directory integrity across **multiple platforms and architectures**. It specializes in calculating checksums (hashes) and comparing them with known values to detect any signs of tampering, alteration, or corruption. Below is a detailed guide to understanding and how to using CrCheck:

Key Features of CrCheck

- **Cross-platform support:** Works seamlessly on modern operating systems, making it a universal choice for file integrity verification. Modern design to support multi-core systems.
- **Comprehensive checksum algorithms:** Utilizes multiple and robust hashing techniques (e.g., SHA-256, SHA-512) for high accuracy in detecting changes.
- **File and directory validation:** Capable of working on both individual files and entire directories to provide a holistic integrity check.
- **Successor to CrCheck 4.xx:** A modern replacement for the legacy DOS version (4.xx), CrCheck 5.xx eliminates compatibility issues while introducing advanced functionality.

👉 Its the successor of CrCheck 4.xx for DOS which is incompatible with modern operating systems.

Important Note: Incompatibility with CrCheck 4.xx

- **CrCheck 5.xx** does **not support checksum files** generated by its predecessor **CrCheck 4.xx/DOS**. Likewise, checksum files created with version 5 cannot be verified using the older 4.xx version.
- This change is due to differences in hashing algorithms, filename handling, attributes and file formats between the two versions, reflecting the shift to modern operating systems and security standards.

Use Case/Business Case

👉 CrCheck is a versatile tool with **two** primary use cases that address data integrity and verification needs:

1. File Verification for Tampering or Modification

In this use case, CrCheck is employed to ensure the integrity of files by checking if they have been altered or tampered with. A common implementation is as follows:

- A checksum file (CrCheck.txt) is generated and provided alongside the files or archives.
- End users can use CrCheck to compare the hash values of their unpacked files with those in the checksum file to verify that the files have not been corrupted or modified during transfer or unpacking by simply executing `crcheck`
- Example: **ROSE SWE's** leverages CrCheck to distribute checksum files with its archives, enabling users to validate the integrity of unpacked data.

2. Checksum File Creation for Data Protection or Validation

This use case caters to individuals or organizations who want to proactively safeguard and validate their data. CrCheck is used to create a checksum file for data stored in specific locations (e.g., cloud drives or local folders). The benefits include:

- **Protection:** Regularly verifying the checksum file ensures that the data remains unchanged over time.
- **Validation:** Facilitates easy detection of unintended modifications or corruption due to issues such as software bugs, unauthorized access, or hardware failures.

- Example: A user maintaining a folder in a cloud drive can periodically generate a checksum file using CrCheck and later revalidate their files to confirm that no modifications have occurred.

These two scenarios highlight CrCheck's utility in promoting data integrity, whether for end-user distribution or personal data management.

How CrCheck Works

CrCheck utilizes different sophisticated hash algorithms. These hash algorithms serve as unique fingerprints for your files. By comparing these fingerprints against trusted values, you can ensure the integrity of your data.

In addition to hash fingerprint validation, CrCheck also verifies file length. This multi-faceted approach provides a comprehensive assessment of file integrity.

Getting Started

Installation

1. Download the CrCheck MP archive from the official website.
2. Extract (unpack) the archive to a directory of your choice.

Usage

1. Open a command prompt or terminal window.
2. Navigate to the directory where you extracted the archive.
3. Rename one of the fitting executables to `crcheck` or `crcheck.exe` :

- * `crcheck-darwin-amd64`
- * `crcheck-darwin-arm64`
- * `crcheck-linux-386`
- * `crcheck-linux-amd64`
- * `crcheck-linux-arm`
- * `crcheck-linux-arm64`
- * `crcheck-windows-386.exe`
- * `crcheck-windows-amd64.exe`

On Linux or MacOS also change the attribute to executable (`chmod +x crcheck`)

4. To verify files in the current directory, simply type `crcheck` and press Enter. This will compare the checksums of your data and files against the values stored in a `CRCHECK.TXT` file (if present).
5. To generate a `CRCHECK.TXT` file for your files, type `crcheck -create > CRCHECK.TXT` . This will create a file named `CRCHECK.TXT` containing the checksums of all files in the current and sub directories.

Options

CrCheck offers several command-line options to customize its behavior:

- `-hash HASH` : Specifies the hash algorithm to use (e.g., `md5` , `sha1` , `sha256`). The default is (currently) `sha1` .
- `-create` : Forces the output of hashes even if a checksum file exists.
- `-help` or `-?` : Displays the help message, showing available options and excluded files.
- `-verbose` : Enable verbose output if file verification fails.

Excluded Files

CrCheck automatically excludes certain files and directories from its verification process. These include:

- `crcheck.txt` , `crcheck.asc` , `crcheck.crc`
- `cvs/entries` , `cvs/root` , `cvs/repository` , `cvs/template`
- `cvs/entries.log` , `cvs/repository.log` , `cvs/root.log`
- `testboot.exe`

👁️ Further files to exclude on user request...

CrCheck.txt file

CrCheck looks for different files that may contain pre-saved filenames, size and hashes. We refer to this as a checksum file.

- 🤖 The following files are supported and checked if they exist:
{"crcheck.txt", "CRCHECK.TXT", "crcheck.txt.asc", "crcheck.crc", "crcheck.asc", "CrCheck.txt", "CrCheck.txt.asc", "CrCheck.crc", "C after the first find this file is used
- 🙌 please note on Linux the file names are case sensitive, on Windows not 😊

Known issues or bugs?

- **Bugs** are yet unknown
- **Issues:** Do to multi platform support all file names are "normalized. This means "/" and "\" as well as upper/lowercase file names are normalized. Therefore files like e.g. "Makefile" and "makefile" in the same directory are **not** supported!

Advanced Usage

PGP/GPG Support

🙌 CrCheck supports PGP and GNUPG signing for enhanced security. You can sign your `CRCHECK.TXT` file with a PGP/GPG key to ensure its authenticity. CrCheck will automatically detect the signature. For Windows, we provide the batch file `cr2gpg.bat` which we use in-house for this task.

To sign the CrCheck.txt file by hand do:

```
gpg --clear-sign CrCheck.txt    ## Windows + Linux, gpg2.exe for cygwin
# You should now have a file `CrCheck.txt.asc`
mv CrCheck.txt.asc CrCheck.txt
```

To verify if the GPG signed file was altered use:

```
gpg --verify CrCheck.txt
```

For Windows and Linux we provide the scripts `cr2gpg.bat` and `cr2gpg.sh` and some helper tools in the subdirectory "helpers". If you want to use the `cr2gpg` script we recommend to copy the helper tools to a directory where your PATH variable points to.

Batch File Integration

You can integrate CrCheck into batch files to automate file verification tasks. This is particularly useful for repetitive checks or scheduled maintenance routines.

Error Levels

CrCheck provides error levels (exit codes) to indicate the outcome of the verification process. You can use these error levels in batch files or scripts to trigger specific actions based on the result.

- 0 : All good.
- ErrCodeFileHash = 1 : File hash mismatch.
- ErrCodeUnknownHash = 2 : Unknown hash algorithm.
- ErrOldCrCheck = 3 : Old version of CrCheck detected.

Example Output

```
c:> crcheck -create -hash sha3 > crcheck.txt

c:> crcheck
CrCheck 5.00 - (c) 1990-2025 by ROSE SWE, Ralph Roth - Antivirus Tool!
-----
Reading checksum file: crcheck.txt
HashType: SHA3
-----[ OK ]----- Makefile
-----[ OK ]----- build/crcheck-darwin-amd64
-----[ OK ]----- build/crcheck-darwin-arm64
-----[ OK ]----- build/crcheck-linux-386
-----[ OK ]----- build/crcheck-linux-amd64
-----[ OK ]----- build/crcheck-linux-arm
-----[ OK ]----- build/crcheck-linux-arm64
-----[ OK ]----- build/crcheck-windows-386.exe
-----[ OK ]----- build/crcheck-windows-amd64.exe
-----[ OK ]----- cr2gpg.bat
-----[ OK ]----- crcheck.go
-----[ OK ]----- crcheck.html
-----[ OK ]----- crcheck.md
-----[ OK ]----- crcheck.pdf
-----[ OK ]----- go.mod
-----[ OK ]----- go.sum
-----[ OK ]----- main.ico
-----[ OK ]----- versioninfo.json
```

🧠 Please note that the file `crcheck.txt` itself is excluded from being added!

After modifying some files we see that these files were tampered

```
c:> crcheck
CrCheck 5.00 - (c) 1990-2025 by ROSE SWE, Ralph Roth - Antivirus Tool!
-----
Reading checksum file: crcheck.txt
HashType: SHA3
-----[ OK ]----- Makefile
-----[ OK ]----- build/crcheck-darwin-amd64
-----[ OK ]----- build/crcheck-darwin-arm64
-----[ OK ]----- build/crcheck-linux-386
-----[ OK ]----- build/crcheck-linux-amd64
-----[ OK ]----- build/crcheck-linux-arm
-----[ OK ]----- build/crcheck-linux-arm64
-----[ OK ]----- build/crcheck-windows-386.exe
-----[ OK ]----- build/crcheck-windows-amd64.exe
-----[ OK ]----- cr2gpg.bat
-----[ OK ]----- crcheck.go
--[ !FAILED! ]-- crcheck.html
--[ !FAILED! ]-- crcheck.md
-----[ OK ]----- crcheck.pdf
-----[ OK ]----- go.mod
-----[ OK ]----- go.sum
-----[ OK ]----- main.ico
-----[ OK ]----- versioninfo.json
```

🧐 If you want to see technical details use the verbose option:

```
C:> crcheck -verbose
CrCheck 5.00 - (c) 1990-2025 by ROSE SWE, Ralph Roth - Antivirus Tool!
-----
Reading checksum file: crcheck.txt
HashType: SHA3
-----[ OK ]----- Makefile
-----[ OK ]----- build/crcheck-darwin-amd64
-----[ OK ]----- build/crcheck-darwin-arm64
-----[ OK ]----- build/crcheck-linux-386
-----[ OK ]----- build/crcheck-linux-amd64
-----[ OK ]----- build/crcheck-linux-arm
-----[ OK ]----- build/crcheck-linux-arm64
-----[ OK ]----- build/crcheck-windows-386.exe
-----[ OK ]----- build/crcheck-windows-amd64.exe
-----[ OK ]----- cr2pgp.bat
-----[ OK ]----- crcheck.go
-=[ !FAILED! ]=- crcheck.html (Size: 37368 -> 38790) (HASH=afa89ce2e96d85433ea25da7b707086409b5235441056280a99e95ec826bde39 -> 5a79
-=[ !FAILED! ]=- crcheck.md (Size: 13337 -> 15257) (HASH=2abdefa29e7f6b71489c0c872abb08cd5b1e3b14279732ff3eb4e7292b5d4581 -> ab1171
-----[ OK ]----- crcheck.pdf
-----[ OK ]----- go.mod
-----[ OK ]----- go.sum
-----[ OK ]----- main.ico
-----[ OK ]----- versioninfo.json
```

👉 If you add files to the “distribution” they are also flagged as new files. This is useful if your distributor adds BBS files or ads files to the archive 😊 Hash algorithm for new files is the same found in the checksum file. Here an example of a GPG signed checksum file:

```
$ crcheck
CrCheck 5.00 - (c) 1990-2025 by ROSE SWE, Ralph Roth - Antivirus Tool!
-----
Reading checksum file: crcheck.txt.asc
HashType: SHA3
a7ffc6f8bf1ed76651c14756a061d662f580ff4de43b49fa82d80a4b80f8434a      0  1  2  3.test (new)
-----[ OK ]----- Makefile
-----[ OK ]----- build/crcheck-darwin-amd64
-----[ OK ]----- build/crcheck-darwin-arm64
-----[ OK ]----- build/crcheck-linux-386
-----[ OK ]----- build/crcheck-linux-amd64
-----[ OK ]----- build/crcheck-linux-arm
-----[ OK ]----- build/crcheck-linux-arm64
-----[ OK ]----- build/crcheck-windows-386.exe
-----[ OK ]----- build/crcheck-windows-amd64.exe
-----[ OK ]----- cr2pgp.bat
-----[ OK ]----- crcheck.go
-=[ !FAILED! ]=- crcheck.html
-=[ !FAILED! ]=- crcheck.md
-=[ !FAILED! ]=- crcheck.pdf
9dc0ce7c7a735358403062cf5aeed12ac74ce0ca5ca7ee926980895894d3ac67      2272  crcheck.txt.asc (new)
-----[ OK ]----- go.mod
-----[ OK ]----- go.sum
-----[ OK ]----- main.ico
-----[ OK ]----- versioninfo.json
```

License Terms for CrCheck

CrCheck is offered under a **user-friendly and flexible licensing model** that balances accessibility with the protection of intellectual property. Below are the detailed licensing terms and usage conditions:

Licensing Overview

Freeware for Non-Commercial Use

- **Use Case 1** (Verifying files for tampering or modification):
 - Free to use for **personal, educational, or non-commercial purposes**.
 - Ideal for ensuring the integrity of files downloaded or shared with checksum files (e.g., verifying software from ROSE SWE).
- **Use Case 2** (Creating checksum files for personal data validation):
 - Also free for **non-commercial purposes**.
 - Useful for individuals maintaining data integrity, such as securing files in personal cloud drives or backups.

Commercial Use

- **Use Case 1** (File verification):
 - **Free for commercial purposes**.
 - Companies can use CrCheck to validate the integrity of distributed files without requiring a license (e.g., check the integrity of unpacked archives from ROSE SWE).
- **Use Case 2** (Checksum file creation for internal or operational data validation):
 - **Requires a commercial license**.
 - Businesses employing CrCheck for creating checksum files (e.g., for internal compliance, auditing, or protecting sensitive operational data) must acquire a license to use this feature legally.

Summary of Licensing Model

Use Case	Non-Commercial	Commercial
Verify files for tampering	Freeware	Freeware
Create checksum files/distributing checksum files	Freeware	Requires a license

Registered vs. Unregistered Versions

- The **functional differences** between registered and unregistered versions are minimal, apart from the **“beg remark”** in the unregistered version.
- Registering CrCheck grants access to the **latest available version**, ensuring you have all recent updates and improvements.
- To register, users need to complete the **REGISTER.TXT** form and submit it via mail or email.

Intellectual Property and Usage Restrictions

By using CrCheck, you agree to the following terms:

1. **Respect for Copyright:** Users must respect the intellectual property rights of the program's creators.
2. **Prohibition of Code Modification:** Modifying the program code or attempting to decompile the software is strictly prohibited.
3. **No Circumvention:** Efforts to bypass the license protection mechanisms or use the software without an appropriate license are considered a violation of the terms of use.












Violations, such as reverse engineering or unauthorized use, constitute a breach of the license agreement and may lead to legal consequences.



This licensing approach ensures that CrCheck remains accessible for various users while maintaining clear boundaries to protect the developers' rights and support sustainable development.

Conclusion

CrCheck is a valuable tool for anyone concerned about file integrity and security. Its comprehensive features, user-friendly interface, and flexible options make it a versatile solution for individuals, organizations, and developers alike.

History

Version	Date	 / 	Changes
5.16	26.04.2025		Non user visible enhancements. Fix of cr2gpg.bat
5.14/5.15	25.03.2025		Small internal enhancements. Help screen now fits into Windows 80 char terminals
5.13	17.02.2025		More small bugfixes (Windows) and enhancements, tested with latest compiler
5.12	09.02.2025		Small enhancements like thousand separator. More fixes for compatibility between Linux and Windows
5.10	08.02.2025		Added number of files checked and total size checked. Added RIPEMD-160, BLAKE2 and BLAKE3 hash algorithm
5.05	03.02.2025		First working multi platform version. Beside the HASH used, now also the original OS and architecture is added
5.04	26.01.2025		Various small enhancements, trying to get the Windows version bugfree
5.01	30.12.2024		The filename is now enclosed by pipe characters " " so we can even handle filenames that start with a tab or space. Missing files are now also reported.
5.00	Oct. 2024		Initial port of CrCheck 4.80 to Multi Platform

Legend
 Nonpublic release
 Published release

Overview of Hash Algorithms

Hash algorithms are essential in computer science and used for various applications such as data integrity verification, checksums, and cryptographic security. This chapter provides an overview of several popular hash algorithms, including Adler32, Castagnoli, CRC32, MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, Blake and xxHash, along with a comparison of their performance.

Hash Algorithms supported by CrCheck

👉 Additional HASH Algorithms can be added on user request.

In **bold** are the keyword you can use for the option `-hash HASH_ALGORITHM` , e.g. **CRC32** and **IEEE** are the same hash algorithm.

1. Adler32 (32 bit)

Adler32/Adler is a checksum algorithm that combines the speed of the Fletcher algorithm with the simplicity of the checksum. It is primarily used in applications like zlib for data compression. While it is fast, it is not cryptographically secure and is mainly used for error-checking.

2. Castagnoli (32 bit)

Castagnoli/Cast is a variant of the CRC32 (Cyclic Redundancy Check) algorithm, specifically designed to improve error detection capabilities. It is often used in networking and storage applications. Like Adler32, it is not suitable for cryptographic purposes.

3. CRC32 (IEEE) (32 bit)

CRC32/IEEE is a widely used checksum algorithm that produces a 32-bit hash value. It is commonly used in network communications and file integrity checks. The IEEE variant is standardized and provides a good balance between speed and error detection.

4. MD5 (128 bit)

MD5 (Message-Digest Algorithm 5) is a widely used cryptographic hash function that produces a 128-bit hash value. It is fast and efficient but has known vulnerabilities to collision attacks, making it unsuitable for security-sensitive applications.

5. SHA-1 (SHA-160, 160 bits)

SHA1/SHA160 is a cryptographic hash function that produces a 160-bit hash value. While it was widely used for digital signatures and certificates, it is now considered weak due to vulnerabilities that allow for collision attacks.

6. SHA-2 (SHA-256, SHA-512 - 256 or 512 bits)

SHA2/SHA256/SHA512/SHA2-512 is a family of cryptographic hash functions that includes SHA-256 and SHA-512. These algorithms are more secure than MD5 and SHA-1, providing resistance against collision and pre-image attacks. SHA-256 produces a 256-bit hash, while SHA-512 produces a 512-bit hash.

7. SHA-3 (256 bit implementation)

SHA3/SHA3-256 is the latest member of the Secure Hash Algorithm family, designed to provide a higher level of security and performance. It uses a different construction method (Keccak) compared to SHA-2 and is considered highly secure.

8. xxHash (64 bit)

xxHash/xxh64 is a non-cryptographic hash function known for its exceptional speed and efficiency. It operates at RAM speed limits and is suitable for applications requiring fast hashing without the need for cryptographic security. Variants include xxHash32 and xxHash64, with xxHash3 being the latest and fastest. See also

<https://chromium.googlesource.com/external/github.com/Cyan4973/xxHash/+/375d401bd4a4eba07ee75d6e627546052cb5b0ec/README.md>

9. RIPEMD-160 (160 bit)

RIPEMD-160/RMD160 is a cryptographic hash function that produces a 160-bit (20-byte) hash value. It was developed as part of the RIPEMD family by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel in 1996. RIPEMD-160 is designed for secure message integrity verification and is similar in structure to MD4 and MD5 but with added security enhancements. While it is less common than SHA-1 or SHA-2, it is still used in some blockchain applications and cryptographic systems.

10. BLAKE2 and BLAKE3 (256 bit)

BLAKE2 is a cryptographic hash function that serves as an improved version of the original BLAKE algorithm. It was designed to be faster than MD5, SHA-1, and SHA-2 while maintaining a comparable level of security. BLAKE2 comes in two primary variants:

- **BLAKE2b/BLAKE2b-256**: Optimized for 64-bit platforms, typically producing a 256-bit hash (among other possible output lengths).
- **BLAKE2s/BLAKE2s-256**: Optimized for 32-bit platforms, also configured here to produce a 256-bit output.

BLAKE3/BLAKE3-256 is a more recent hash function that builds on the design principles of BLAKE2 but further enhances speed, simplicity, and parallelism. It is designed to offer improved performance without compromising security, and in our context, it also produces a 256-bit hash output.

11. BLAKE3-128 (128 bit)

BLAKE3-128 is a variant of the BLAKE3 hash function configured to produce a 128-bit (16-byte) output. BLAKE3 itself is known for its speed, parallelism, and robust security properties, and the 128-bit version offers these same benefits in terms of performance. However, by reducing the output size, the cryptographic strength is proportionally lowered:

- **Security**: With a 128-bit digest, the collision resistance and pre-image resistance are less robust compared to the default 256-bit configuration. While this may be adequate for non-critical applications or situations where a shorter checksum is sufficient, it is generally not recommended for high-security contexts where collision resistance is paramount.
- **Performance**: Like its 256-bit counterpart, BLAKE3-128 benefits from the algorithm's inherent speed and efficiency. The reduced output size might offer a marginal performance improvement, but the primary advantage remains the exceptional throughput and parallelism of BLAKE3.
- **Usage Considerations**: Choosing BLAKE3-128 can be a trade-off between digest size and security level. For applications where minimizing storage or bandwidth is crucial and the security requirements are moderate, BLAKE3-128 can be an attractive option. However, for most cryptographic purposes, the 256-bit version is recommended to ensure a higher level of security.

This variant maintains the design simplicity and speed of BLAKE3 while offering a shorter hash output that might be suitable for specific use cases with relaxed security demands.

Additional Cryptographic Hash Overviews

Whirlpool

- Produces a **512-bit hash** and is resistant to many common attacks.
- Less commonly used than SHA-2 or BLAKE2 but remains a strong alternative.

Tiger

- Designed for **64-bit systems** and known for its speed, but it has been largely replaced by stronger algorithms.

Skein

- A flexible hash function that supports **variable-length outputs**, developed for the **SHA-3 competition** (though it lost to Keccak).

Performance Comparison

Summary Table of Hash Algorithms

Here's a comparative summary of various hash algorithms (not all implemented in CrCheck), including both cryptographic and non-cryptographic options, focusing on their **speed**, **bit output**, and **security**.

Algorithm	Speed	Bit Output	Security Level
xxHash	Extremely fast	32/64 bits (XXH3)	Non-cryptographic
t1ha	Very fast	32/64 bits	Non-cryptographic
MurmurHash	Fast	32/128 bits	Non-cryptographic
GxHash	Very fast	Variable (32 bits+)	Non-cryptographic
MD5	Fast	128 bits	Insecure
SHA-1	Moderate	160 bits	Insecure
RIPEMD-160 (RIPE160)	Moderate	160 bits	Secure
SHA-2	Moderate	224/256/384/512 bits	Secure
SHA-3	Moderate	224/256/384/512 bits	Secure
BLAKE2	Fast	256/512 bits	Secure
BLAKE3	Very fast	256 bits	Secure
Whirlpool	Moderate	512 bits	Secure
Tiger	Fast	192 bits	Secure (legacy)
Skein	Moderate	Variable (256-512 bits)	Secure
CRC32	Very fast	32 bits	Non-cryptographic
Adler32	Very fast	32 bits	Non-cryptographic
Castagnoli	Very fast	32 bits	Non-cryptographic

Hash Algorithms - Key Points

- **Speed:** xxHash, t1ha, MurmurHash, GxHash, CRC32, and Adler32 are the fastest. **BLAKE3** is the fastest secure cryptographic hash.
- **Bit Output:** Cryptographic hashes like **SHA-2, SHA-3, RIPEMD-160, BLAKE2, BLAKE3, Whirlpool, Tiger, and Skein** produce larger outputs, improving security.
- **Security:**
 - **Insecure:** MD5 and SHA-1 are vulnerable to collisions and should not be used.

- **Secure:** SHA-2, SHA-3, BLAKE2, BLAKE3, RIPEMD-160, Whirlpool, and Skein are secure cryptographic options.
- **Legacy Secure:** Tiger was once used in cryptographic applications but is now mostly outdated.
- **Non-cryptographic:** xxHash, CRC32, Adler32, and Castagnoli are designed for speed, not security.

Performance Ranking by Speed

When comparing speed, xxHash is the fastest, operating at near RAM speed. Below is a general ranking:

1. **xxHash** – Fastest, ideal for non-cryptographic use.
2. **BLAKE3** – Fastest cryptographic hash, excellent for modern applications.
3. **Adler32** – Fast checksum algorithm.
4. **Castagnoli** – Fast CRC variant.
5. **CRC32** – Fast, widely used for checksums.
6. **MD5** – Fast but insecure.
7. **SHA-1** – Slower than MD5, with known vulnerabilities.
8. **RIPEMD-160** – Moderate speed, more secure than SHA-1.
9. **SHA-2 (SHA-256, SHA-512)** – Slower but more secure.
10. **SHA-3** – Generally slower than SHA-2 but provides enhanced security.
11. **Whirlpool** – 512-bit output, moderate speed.
12. **Skein** – Secure but not widely adopted.

Reference: Empty Hashes

Below is a table that lists each hash algorithm alongside its hash value for an empty file (i.e. an empty input, file size zero):

Hash Algorithm	Hash Value
Adler32	00000001
BLAKE2b-256	0e5751c026e543b2e8ab2eb06099daa1c9e8b7526c5c9b9a32bfed7ac92b48d1
BLAKE3	af1349b9d1a7de16e5a26e0ef8a5d2a0e9b78b12d44e0c155f3b8a9e7dfd9f7f
CRC32, IEEE	00000000
CRC32 (Castagnoli)	00000000
MD4	31d6cfe0d16ae931b73c59d7e0c089c0
MD5	d41d8cd98f00b204e9800998ecf8427e
RIPEMD160	9c1185a5c5e9fc54612808977ee8f548b2258d31
SHA-1	da39a3ee5e6b4b0d3255bfef95601890afd80709
SHA-256	e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
SHA-384	38b060a751ac96384cd9327eb1b1e36a21fdb71114be07434c0cc7bf63f6e1da274edebfe76f65fbd51ad2f14898b95b
SHA-512	cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a5
SHA3-224	6b4e03423667dbb73b6e15454f0e6b22a2e0371be0b4d7eaa5d30f3e
SHA3-256	a7ffc6f8bf1ed76651c14756a061e667b6e56dd99a0ac2a0f2bb6dbd7f6cdb0d
SHA3-384	0c63a75b845e4f7d01107d852e4c2485c51a50aaaa94fc824d7a7b17d5c32c6e9a286dcddb9ac4c0a56c
SHA3-512	a69f73cca23a9ac5c8b567dc185a756e97c982164fe25859e0d1dccf8e6d0a14d6e2c85ed87f7f1d9c9c2ea2c1061fa1c86f2d0a3e10f8a4dfe6
XXH32	02cc5d05

Hash Algorithm	Hash Value
xxHash (64 bit)	0ef46db3751d8e999

Each value is computed from hashing an empty input. These constants are defined by each algorithm and can be used as test vectors to verify correct implementations.

End of documentation