

2015

DeadHash

HELP DOCUMENTATION
DEADLINE

CODEDEAD | <http://codedead.com>

Table of contents

1. Introduction.....	2
2. System requirements	2
2.1 Operating System	2
2.2 Disk space	2
2.3 RAM memory	2
2.4 Internet connection.....	2
2.5 .NET Framework.....	2
3. License	2
4. Support	2
5. Website	2
6. Explanation.....	3
6.1 MD5	3
6.2 SHA-1	3
6.3 SHA-2	3
6.4 RIPEMD.....	4
6.5 CRC	4
6.6 UTF-8	5
7. Frequently Asked Questions.....	6
7.1 I get an error, what do I do ?.....	6
7.2 I found a bug, what do I do ?.....	6
7.3 Can I send this application to other people ?.....	6
7.4 Why can't I open DeadHash ?	6
7.5 Why does my anti-virus warn me about DeadHash ?	6
7.6 Why can't I use the drag and drop functionality?	7
7.7 Which type of text encoding does DeadHash use to generate hashes ?	7
7.8 How does the Folder Monitor work?	7
7.9 Why doesn't the Folder Monitor do anything when I press start?	7
8. Sources	8

1. Introduction

DeadHash was created in order to help people verifying downloads and other files.

Our website:

<http://codedead.com>

DeadHash was created by DeadLine. Icons and images were created by Icons8. For more information about Icons8, please visit their website:

<http://icons8.com>

The GUI uses a theme created by DevComponents (Metro). For more information about DevComponents, please visit their website:

<http://devcomponents.com>

2. System requirements

2.1 Operating System

Windows 7 Service Pack 1; Windows 8; Windows 8.1; Windows Server 2008 R2 SP1; Windows Server 2008 Service Pack 2; Windows Server 2012; Windows Server 2012 R2; Windows Vista Service Pack 2; Windows 10

2.2 Disk space

DeadHash requires at least 20 MB of disk space.

2.3 RAM memory

DeadHash requires at least 30 MB of free RAM memory.

2.4 Internet connection

DeadHash requires an internet connection in order to check for updates. You can, however, disable update checks in the settings menu.

2.5 .NET Framework

DeadHash requires the .NET Framework v4.5.2. You can download the .NET Framework v4.5.2 here: <http://www.microsoft.com/en-us/download/details.aspx?id=42642>

3. License

DeadHash is licensed under GNU GENERAL PUBLIC LICENSE:

<http://codedead.com/Software/DeadHash/gpl.pdf>

4. Support

If you need support, feel free to e-mail us:

admin@codedead.com

5. Website

You can visit our website by navigating to the link below in your webbrowser:

<http://codedead.com>

6. Explanation

6.1 MD5

The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32 digit hexadecimal number.

6.2 SHA-1

In cryptography, SHA-1 is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST.

SHA-1 produces a 160-bit (20-byte) hash value. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.

SHA stands for "secure hash algorithm". The four SHA algorithms are structured differently and are named SHA-0, SHA-1, SHA-2, and SHA-3. SHA-0 is the original version of the 160-bit hash function published in 1993 under the name "SHA": it was not adopted by many applications. Published in 1995, SHA-1 is very similar to SHA-0, but alters the original SHA hash specification to correct alleged weaknesses.

6.3 SHA-2

SHA-2 is a set of cryptographic hash functions designed by the NSA (U.S. National Security Agency). SHA stands for Secure Hash Algorithm. Cryptographic hash functions are mathematical operations run on digital data; by comparing the computed "hash" (the execution of the algorithm) to a known and expected hash value, a person can determine the data's integrity. For example, computing the hash of a downloaded file and comparing the result to a previously published hash result can show whether the download has been modified or tampered with. A key aspect of cryptographic hash functions is their one-way nature: given a computed hash value, it is very difficult to derive the original data.

SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

SHA-256 and SHA-512 are novel hash functions computed with 32-bit and 64-bit words, respectively. They use different shift amounts and additive constants, but their structures are otherwise virtually identical, differing only in the number of rounds. SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values. SHA-512/224 and SHA-512/256 are also truncated versions of SHA-512, but the initial values are generated using the method described in FIPS PUB 180-4. SHA-2 was published in 2001 by the NIST as a U.S. federal standard (FIPS). The SHA-2 family of algorithms are patented in US 6829355. The United States has released the patent under a royalty-free license.

In 2005, security flaws were identified in SHA-1, namely that a mathematical weakness might exist, indicating that a stronger hash function would be desirable. Although SHA-2 bears some similarity to the SHA-1 algorithm, these attacks have not been successfully extended to SHA-2.

6.4 RIPEMD

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) is a family of cryptographic hash functions developed in Leuven, Belgium, by Hans Dobbertin, Antoon Bosselaers and Bart Preneel at the COSIC research group at the Katholieke Universiteit Leuven, and first published in 1996. RIPEMD was based upon the design principles used in MD4, and is similar in performance to the more popular SHA-1.

RIPEMD-160 is an improved, 160-bit version of the original RIPEMD, and the most common version in the family. RIPEMD-160 was designed in the open academic community, in contrast to the NSA designed SHA-1 and SHA-2 algorithms. On the other hand, RIPEMD-160 appears to be used somewhat less frequently than SHA-1, which may have caused it to be less scrutinized than SHA. RIPEMD-160 is not known to be constrained by any patents.

As well as 160-bit, there also exist 128, 256 and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320, respectively. The 128-bit version was intended only as a drop-in replacement for the original RIPEMD, which was also 128-bit, and which had been found to have questionable security. The 256 and 320-bit versions diminish only the chance of accidental collision, and don't have higher levels of security (against preimage attacks) as compared to, respectively, RIPEMD-128 and RIPEMD-160.

6.5 CRC

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

The CRC was invented by W. Wesley Peterson in 1961; the 32-bit CRC function of Ethernet and many other standards is the work of several researchers and was published in 1975.

6.6 UTF-8

UTF-8 (U from Universal Character Set + Transformation Format—8-bit) is a character encoding capable of encoding all possible characters (called code points) in Unicode. The encoding is variable-length and uses 8-bit code units. It was designed for backward compatibility with ASCII and to avoid the complications of endianness and byte order marks in UTF-16 and UTF-32.

UTF-8 encodes each of the 1,112,064 valid code points in the Unicode code space (1,114,112 code points minus 2,048 surrogate code points) using one to four 8-bit bytes (a group of 8 bits is known as an octet in the Unicode Standard). Code points with lower numerical values (i.e. earlier code positions in the Unicode character set, which tend to occur more frequently) are encoded using fewer bytes. The first 128 characters of Unicode, which correspond one-to-one with ASCII, are encoded using a single octet with the same binary value as ASCII, making valid ASCII text valid UTF-8-encoded Unicode as well.

The official IANA code for the UTF-8 character encoding is UTF-8.

7. Frequently Asked Questions

7.1 I get an error, what do I do ?

If you get an exception, please take a screenshot of the exception and e-mail it to: admin@codedead.com

Please include the following information in your e-mail:

- Operating System version
- Application version
- Screenshot of the exception
- Description

We will try to help you as soon as possible.

7.2 I found a bug, what do I do ?

Unfortunately it is possible that a bug has slipped through our fingers. If you happen to find a bug, feel free to e-mail us at: admin@codedead.com

Please include the following information in your e-mail:

- Screenshot or video (not required)
- Detailed description
- Application version

We will do our best to fix the issue !

7.3 Can I send this application to other people ?

Yes, but please read our license agreement before sending DeadHash to someone else:

<http://codedead.com/Software/DeadHash/gpl.pdf>

7.4 Why can't I open DeadHash ?

There could be several reasons why you are unable to open DeadHash. Please try one of the following methods:

- Disable your anti-virus
- Disable Windows SmartScreen
- Reinstall DeadHash
- Check the system requirements

Warning: If you disable your antivirus or Windows SmartScreen, please re-enable it after testing.

If none of these methods worked for you, please contact us via e-mail:

admin@codedead.com

7.5 Why does my anti-virus warn me about DeadHash ?

It is possible that your anti-virus falsely identifies DeadHash as malware. This is known as a false positive. Please contact your anti-virus support service if this is the case.

7.6 Why can't I use the drag and drop functionality?

If you run DeadHash elevated (Run as Administrator), you will not be able to use the drag and drop functionality. In order to prevent potential elevation of privilege attacks, certain functionality needs to be blocked. This is implemented through Mandatory Integrity Control (MIC). All processes and all resources (files, registry, etc.) have an integrity level assigned. MIC prevents a standard user process from writing to a protected per machine location like Program Files or the HKLM registry hive. You can fix this issue by running DeadHash without administrative privileges.

7.7 Which type of text encoding does DeadHash use to generate hashes ?

DeadHash uses UTF-8 encoding to generate file or text hashes.

7.8 How does the Folder Monitor work?

The Folder Monitor uses a FileSystemWatcher. The FileSystemWatcher is a standard component of the .NET Framework. It listens to the file system change notifications and raises events when a directory, or file in a directory, changes.

7.9 Why doesn't the Folder Monitor do anything when I press start?

The Folder Monitor uses a FileSystemWatcher. Sometimes, it is required to manually create, change, delete or rename a file in order to start the FileSystemWatcher.

For more information about the FileSystemWatcher, please visit the website below:

<https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher%28v=vs.110%29.aspx>

8. Sources

<https://en.wikipedia.org/wiki/MD5>

<https://en.wikipedia.org/wiki/SHA-1>

<https://en.wikipedia.org/wiki/SHA-2>

<https://en.wikipedia.org/wiki/RIPEMD>

https://en.wikipedia.org/wiki/Cyclic_redundancy_check

<https://en.wikipedia.org/wiki/UTF-8>

<https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher%28v=vs.110%29.aspx>