

# Hash Code and Checksum - what's the difference?

Asked 15 years, 11 months ago    Modified 1 year, 10 months ago

Viewed 72k times



**144**

My understanding is that a hash code and checksum are similar things - a numeric value, computed for a block of data, that is *relatively* unique.



i.e. The probability of two blocks of data yielding the same numeric hash/checksum value is low enough that it can be ignored for the purposes of the application.



So do we have two words for the same thing, or are there important differences between hash codes and checksums?

language-agnostic

hash

computer-science

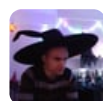
checksum

Share

Improve this question

Follow

edited Aug 22, 2013 at 21:08



**Mark Amery**

154k ● 89 ● 426 ● 469

asked Jan 20, 2009 at 9:28



**Richard Ev**

54k ● 61 ● 194 ● 280

---

5 To summarize the answers below: A hash code reduces the input to a small number, in a way that minimizes the chance of collisions. A checksum, on the other hand, reduces the input to a small number, in a way that minimizes the chance of collisions. You can make one sound different from the other by arbitrarily rephrasing that description. – [Dan Stahlke](#) Jul 28, 2015 at 22:52 ✎

---

4 @DanStahlke - No, that isn't what the answers below say. Yes, they both reduce input to a smaller number. But there are many, many ways to do so, how to choose what algorithm to use? That depends on your goal. To summarize the top two answers: the *goal* of a checksum is "*to detect the most common errors*". Choose an algorithm that yields a different checksum, for whatever errors are "most common" in your scenario. If you are worried about one or two bits being toggled, you can pick an algorithm that *guarantees* detection of that specific error! This is a very specific trade-off. – [ToolmakerSteve](#) Mar 1, 2018 at 5:07 ✎

---

1 @DanStahlke - on the other hand, *hash code* covers a broad range of possible trade-offs. If we mean a value used in making a hash table, we know that there *will* be collisions, lots of them. This is a very different trade-off (than a checksum). We are trying to reduce collisions *on average*. We don't guarantee anything. There may be some inputs that differ by only one bit, yet yield the same hash. This is perfectly fine, if *on average* we get a good spread of hash values. Yet would be unacceptable for a checksum. – [ToolmakerSteve](#) Mar 1, 2018 at 5:17 ✎

---

[More on what hash values are used for and their focus..](#)  
– [CGTheLegend](#) Mar 16, 2019 at 4:53

---

@DanStahlke comments are not intended for summarizing the existing answers of a question. If you think that such a summary would be useful, you can post the summary as a separate answer. See the [help center](#): "*Comments are not*

*recommended for any of the following: [...] Answering a question or providing an alternate solution to an existing answer; instead, post an actual answer (or edit to expand an existing one);"* – [Theodor Zoulas](#) Oct 14, 2023 at 16:10

---

13 Answers

Sorted by:

Highest score (default)



I would say that a [checksum](#) is **necessarily** a [hashcode](#). However, not all hashcodes make good checksums.

93



A checksum has a special purpose --- it verifies or *checks* the integrity of data (some can go beyond that by allowing for [error-correction](#)). "Good" checksums are easy to compute, and can detect many types of data corruptions (e.g. one, two, three erroneous bits).



A hashcode simply describes a [mathematical function](#) that maps data to some value. When used as a means of indexing in data structures (e.g. a hash table), a low collision probability is desirable.

Share Improve this answer

edited Jan 20, 2009 at 9:42

Follow

answered Jan 20, 2009 at 9:31



[Zach Scrivena](#)

29.5k ● 12 ● 65 ● 73

---

7 Maybe one could be used as the other, but considering that they have different design goals this just confuses the issue.  
– [Wim Coenen](#) Jan 20, 2009 at 10:55

---

8 @gumbo: no, not every hashcode is a checksum. See string example from MSalters below. – [March](#) Mar 17, 2016 at 16:18

---



46



There is a different purpose behind each of them:

- Hash code - designed to be random across its domain (to minimize collisions in hash tables and such). Cryptographic hash codes are also designed to be computationally infeasible to reverse.
- Check sum - designed to detect the most common errors in the data and often to be fast to compute (for effective checksumming fast streams of data).

In practice, the same functions are often good for both purposes. In particular, a cryptographically strong hash code is a good checksum (it is almost impossible that a random error will break a strong hash function), if you can afford the computational cost.

Share Improve this answer

answered Jan 20, 2009 at 9:43

Follow



[Rafał Dowgird](#)


45k ● 11 ● 79 ● 94

- 
- 1 Also it's good to mention that non-cryptographic version of hash codes may provide a good tradeoff between computation time (close to CRC) and error detection, whether it's intentional or just communication error/bit rot (CRC cannot be expected to detect intentional tampering because it's relatively easy to intentionally design a collision).

– [gaborous](#) Mar 4, 2015 at 5:34

---

- 2 To me, the key phrase in your answer, is that checksum is *designed to detect the most common errors*. Yes, that's it. it is a hash algorithm that has been chosen to yield different values for *likely* corruptions of the data. That is a specific purpose, and leads to specific algorithms, which optimize for that - depending on the types of perturbations one is concerned about. – [ToolmakerSteve](#) Mar 1, 2018 at 4:59
- 

A functional checksum might have lots of collisions across its entire domain, as long as errors you are checking for are unlikely to lead to such collisions. For example, there could be some periodicity in your checksum that you don't care about, because errors never take you there. (A good example is parity, which only protects against single-bit errors.) So the definition of a good checksum inherently depends on the error model. – [prolyx](#) Oct 16, 2020 at 14:06 

---



26



There are indeed some differences:

- Checksums just need to be different when the input is different (as often as possible), but it's almost as important that they're fast to compute.
- Hash codes (for use in hashtables) have the same requirements, and additionally they should be evenly distributed across the code space, especially for inputs that are similar.
- Cryptographic hashes have the *much* more stringent requirement that given a hash, you cannot construct an input that produces this hash. Computation times comes second, and depending on the applicatin it may even be desirable for the hash to be very slow to compute (in order to combat brute force attacks).

answered Jan 20, 2009 at 9:46

**Michael Borgwardt**

346k ● 80 ● 486 ● 723

---

1 I don't think checksums being different for different inputs has any benefits. They're just for checking integrity, not for hashing. – [user541686](#) Jul 3, 2012 at 22:58

---

1 @Mehrdad: so how do you propose checking integrity without getting different results for different inputs? – [Michael Borgwardt](#) Jul 6, 2012 at 7:05

---

Er, maybe I misworded what I said? I was referring to the part where you said "as far as possible" -- I'm just saying there's no reason for them to be unpredictable or "far" like hashes are. As long as there is *some* change in the checksum when the input undergoes a typical change, it's a fine checksum. Contrast that with hashes, which also have the goal of distributing things as evenly/randomly/unpredictably/"far" as possible onto their codomain. – [user541686](#) Jul 6, 2012 at 7:27 ✎

---

I think you just misinterpreted what I meant with "as far as possible" - I just meant that collisions should be as rare as possible, though of course they're unavoidable. I'll change the wording. – [Michael Borgwardt](#) Nov 24, 2016 at 9:53

---

- 1 @ToolmakerSteve: It's been over 5 years, but yeah, I think that's what I was referring to. Checksums aren't meant to protect against adversaries. Even if you can find a 1 KB string that produces the same checksum as a 1 MB string, that's not really a problem for a checksum since the probability of that happening by accident is likely to be practically zero. – [user541686](#) Mar 1, 2018 at 5:13
- 



19



Hashcodes and checksums are both used to create short numerical values from a data item. The difference is that a checksum value should change, even if only a small modification is made to the data item. For a hash value, the requirement is merely that real-world data items should have distinct hash values.

A clear example are strings. A checksum for a string should include each and every bit, and order matters. A hashcode on the other hand can often be implemented as a checksum of a limited-length prefix. That would mean that "aaaaaaaaaaba" would hash the same as "aaaaaaaaaaaab", but hash algorithms can deal with such collisions.

Share Improve this answer

Follow

edited Oct 1, 2021 at 11:48



[Matthias Braun](#)

34.1k ● 27 ● 152 ● 176

answered Jan 20, 2009 at 9:46



[MSalters](#)

179k ● 11 ● 164 ● 368

- 
- 1 This answer is the one that rings the bell for me. So data integrity is not the focus of a hash. – [daparic](#) Jun 4, 2020 at 19:37
- 



10



Although hashing and checksums are similar in that they both create a value based on the contents of a file, hashing is not the same as creating a checksum. A checksum is intended to verify (check) the integrity of data and identify data-transmission errors, while a hash is designed to create a unique digital fingerprint of the data.

Source: CompTIA® Security+ Guide to Network Security Fundamentals - Fifth Edition - Mark Ciampa -Page 191

Share Improve this answer

answered Mar 13, 2018 at 11:15

Follow



[N Randhawa](#)

9,461 ● 4 ● 46 ● 48



7



[Wikipedia](#) puts it well:

Checksum functions are related to hash functions, fingerprints, randomisation functions, and cryptographic hash functions. However, each of those concepts has different applications and therefore different design goals. Check digits and parity bits are special cases of checksums,





appropriate for small blocks of data (such as Social Security numbers, bank account numbers, computer words, single bytes, etc.). Some error-correcting codes are based on special checksums that not only detect common errors but also allow the original data to be recovered in certain cases.

Share Improve this answer

answered Jan 20, 2009 at 9:32

Follow



[Jon Skeet](#)

1.5m ● 889 ● 9.3k ● 9.3k

---

35 After reading that, I'm still wondering what the difference is.  
– [kirk.burleson](#) Jul 27, 2010 at 13:50

---

@kirk.burleson - I would say that they are the same *principle*, but in practice one always makes *tradeoffs*. In different situations, different tradeoffs apply, so different approaches are used. Not really a justification for there being two different words, just saying that if you search for good techniques for checksums, you may find a different set of algorithms than when searching for hash codes.  
– [ToolmakerSteve](#) Mar 1, 2018 at 4:45

---



A checksum protects against accidental changes.

6



A cryptographic hash protects against a very motivated attacker.

When you send bits on the wire, it may accidentally happen that some bits are either flipped, or deleted, or





inserted. To allow the receiver to detect (or sometimes correct) accidents like this, the sender uses a checksum.

But if you assume there is someone actively and intelligently modifying the message on the wire and you want to protect against this sort of attacker, then use a cryptographic hash (I am ignoring cryptographically signing the hash, or using a secondary channel or such, since the question does not seem to elude to this).

Share Improve this answer

edited Nov 20, 2014 at 20:22

Follow

answered Nov 20, 2014 at 20:14



user3464863

61 ● 1 ● 2

- 
- 3 "cryptographic hash" increases the confusion between "hash" and "checksum". "cryptographic checksum" is better because it does not. – [March](#) Mar 17, 2016 at 16:15
- 



5

The difference between hash-code and checksum functions is, they are being designed for different purposes.



- A checksum is used to find out **if** something in the input has changed.
- A hash-code is used to find out **if** something in the input has changed **and** to have as much "distance" between individual hash-code values as possible.



Also, there *might* be further requirements for a hash-function, in opposition to this rule, like the ability to form trees/clusters/buckets of hash-code values early.

And if you add some shared initial randomization, you get to the concept for modern encryption/key-exchanges.

---

### About Probability:

For example, lets assume that the input data actually always changes (100% of the time). And lets assume you have a "perfect" hash/checksum function, that generates a 1-bit hash/checksum value. Therefore, you will get different hash/checksum values, 50% of the time, for random input-data.

- If exactly 1 bit in your random input data has changed, you will be able to detect that 100% of the time, no matter how large the input data is.
- If 2 bits in your random input data have changed, your probability of detecting "a change" is divided by 2, because both changes could neutralize each other, and no hash/checksum function would detect that 2 bits are actually different in the input data.

...

This means, If the number of bits in your input data is multiple times larger than the number of bits in your hash/checksum value, your probability of actually getting

different hash/checksum values, for different input values, gets reduced and **is not a constant**.

Share Improve this answer

answered Feb 27, 2014 at 4:35

Follow



[Sascha Wedler](#)

413 ● 4 ● 7



4



These days they are interchangeable, but in days of yore a checksum was a very simple technique where you'd add all the data up (usually in bytes) and tack a byte on the end with that value in.. then you'd hopefully know if any of the original data had been corrupted. Similar to a check bit, but with bytes.



Share Improve this answer

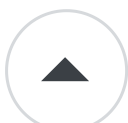
answered Jan 20, 2009 at 9:33

Follow



[Steven Robbins](#)

26.6k ● 7 ● 78 ● 90



2



I tend to use the word checksum when referring to the code (numeric or otherwise) created for a file or piece of data that can be used to *check* that the file or data has not been corrupted. The most common usage I come across is to check that files sent across the network have not been altered (deliberately or otherwise).



Share Improve this answer

answered Jan 20, 2009 at 9:38

Follow



[Ian1971](#)

3,696 ● 7 ● 36 ● 61

- 
- 1 Because checksums are not made to be difficult to reverse, this suggests that they wouldn't be good for checking whether something was deliberately altered. – [benblasdell](#)  
Oct 17, 2012 at 18:59
- 



0



In Redis cluster data sharding, it uses a `hash slot` to decide which node it goes. Take for example the modulo operation below:

```
123 % 9 = 6
122 % 9 = 5
141 % 9 = 6
```



The `6` comes up twice across differing inputs. The purpose of the hash is simply to map an input value to an output value and uniqueness is not part of the deal. So two different inputs that produces the same output is fine in the world of hashes.

A checksum, on the other hand, must differ the output even if one bit in the input changes because its purpose is not to map, but to detect data corruption. So two different inputs that produces the same output is not acceptable in a checksum.

Share Improve this answer

Follow

answered Jun 4, 2020 at 19:50



[daparic](#)

4,394 ● 2 ● 44 ● 44

---



-1



## Hash code vs Check sum

- `hash code` (Sip Hash) usually is used for hash table based structures(Dictionary, Set, HashMap...) [\[Swift dictionary, Set\]](#) where basic operations has a constant time -  $O(1)$
- `check sum` (MD5, SHA) is used to indicate data integrity. For example check sum is calculated for creating `Digital signature` [\[About\]](#)

The main difference is that `check sum` must be **unique** while `hash code` can be the same for different objects. For example in Java or Swift you `hash code` is limited by `Int`. Usually it used in conjunction with `equals` function. Two different objects can have the same `hash code`.

[\[Java hash code\]](#)

Share Improve this answer

edited Feb 9, 2023 at 15:59

Follow

answered Nov 19, 2020 at 21:39



yoAlex5

34k ● 10 ● 223 ● 239



-5

A checksum is simply a number generated from the data field by oring(by logical addition hence sum). The checksum has the capability to detect a corruption of any bit or number of bits within the data field from which it is generated ie it checks for errors that is all, it can not



correct them. A checksum is a hash because the size of the checksum is smaller than the original data. Yes you will have collisions because the checksum is not at all sensitive to bit position in the data field.

A cyclic redundancy check (CRC) is something quite different, more complex **and is NOT called a checksum**. It is the application of a polynomial series which has the capability of correcting any chosen number of individual corrupted bits within the data field from which it was generated. The creation of a CRC results in a number greater in size than the original datafield (unlike the checksum) - hence the name including the word "redundancy" and the price you pay for the error correcting capability. A CRC is therefore NOT a hash and must not be confused or named as a checksum, because the redundancy necessarily adds to the size of the original data.

Share Improve this answer

Follow

edited Feb 24, 2020 at 16:54



user229044 ♦

239k ● 41 ● 344 ● 346

answered Feb 24, 2020 at 15:59



CaptainSensible

1