

SHA-256 verstehen

Bytes, Hashwerte und die Grenzen einer Prüfsumme

Öffentliche Leseausgabe · Version 1.0 · 18. September 2026

Webfassung: raphaellugmayr.at/gedanken/sha256-verstehen

Die kurze Antwort

SHA-256 berechnet aus einer Eingabe einen Hashwert mit 256 Bits. Als Hexadezimalzahl geschrieben sind das 64 Zeichen. Identische Eingabebytes ergeben denselben Hashwert. Ein Hashvergleich hilft deshalb, eine Datei mit einem bekannten Referenzstand zu vergleichen. Er sagt für sich genommen weder, wer die Datei erstellt hat, noch ob ihr Inhalt richtig oder ungefährlich ist.

Dieser Text entwickelt einen Ausschnitt meines Unterrichtsleitfadens „Wie Computer funktionieren“ weiter. Mich interessiert daran die Verbindung zur praktischen Arbeit: Wer eine Prüfsumme bei einem Download, einer Sicherung oder einem Protokoll verwendet, sollte erklären können, welche Frage damit beantwortet wird.

Warum 256 Bits genau 64 Hexzeichen ergeben

Ein Bit kann zwei Werte annehmen. Vier Bits besitzen 16 mögliche Muster, passend zu den Hexziffern von 0 bis F. Eine Hexziffer beschreibt daher vier Bits. Daraus folgt für die Darstellung eines SHA-256-Ergebnisses:

256 Bits / 8 = 32 Bytes
256 Bits / 4 = 64 Hexzeichen

Die Zeichenfolge ist eine Schreibweise des Ergebnisses. Sie ist nicht die ursprüngliche Datei in komprimierter Form. Selbst eine sehr große Datei hat nach dem Hashen einen gleich langen Ergebniswert. Das bedeutet auch: Aus diesem kurzen Wert lässt sich die Datei nicht allgemein wiederherstellen.

Eine konkrete Rechnung macht die Zahl greifbar: Ein Byte besitzt acht Bits, also 256 mögliche Muster. In Hex lassen sich diese Muster mit zwei Stellen ausdrücken, von 00 bis FF. Ein Hash aus 32 Bytes benötigt entsprechend 32 solche Paare. Groß- und Kleinschreibung ändern die dargestellten Zahlenwerte nicht.

Der sichtbare Text ist noch nicht die Eingabe

Für Text muss feststehen, welche Bytes gehasht werden. Eine Zeichenkodierung übersetzt Zeichen in Bytes; Zeilenenden gehören ebenfalls zur Datei. Die drei ASCII-Zeichen abc bestehen in UTF-8 aus den Bytes 61 62 63.

```
abc ohne Zeilenende: 61 62 63
abc mit LF:          61 62 63 0A
abc mit CRLF:        61 62 63 0D 0A
```

Das sind drei verschiedene Eingaben. Wenn zwei Werkzeuge unterschiedliche Werte liefern, würde ich zuerst die Bytes prüfen: Wurde ein Zeilenende angefügt? Ist eine andere Kodierung im Spiel? Enthält die Datei am Anfang eine Byte-Order-Mark? Auch Text, der gleich aussieht, kann unterschiedlich kodiert sein.

Für die drei Bytes 61 62 63 lautet der SHA-256-Wert:

```
ba7816bf8f01cfea414140de5dae2223
b00361a396177a9cb410ff61f20015ad
```

Beide Zeilen gehören zusammen. Der Umbruch dient hier nur der Lesbarkeit. Die Webfassung dieses Leitfadens enthält eine interaktive Browser-Demo. Die ausführliche Dateiübung steht im Beitrag [Dateien mit PowerShell prüfen](#).

Was im Algorithmus geschieht

Der [Secure Hash Standard von NIST](#) beschreibt die genaue Verarbeitung. Vereinfacht ergänzt SHA-256 die Nachricht um Padding und ihre ursprüngliche Länge. Die ergänzte Nachricht wird in 512-Bit-Blöcke geteilt. Jeder Block durchläuft 64 Runden mit 32-Bit-Wörtern, Bitoperationen und Addition modulo 2 hoch 32. Der fortgeschriebene Zustand ergibt schließlich die 256-Bit-Ausgabe.

Die unterschiedlichen Größen beantworten unterschiedliche Fragen: 512 Bits beschreiben einen verarbeiteten Block; 32 Bits die verwendeten Wörter; 256 Bits das Ergebnis. Die Zahl 64 taucht sowohl bei den Runden als auch bei der Hexdarstellung auf, bezeichnet dort aber etwas anderes.

Diese Übersicht ist eine Erklärung des Ablaufs. Für eigene Anwendungen würde ich eine etablierte Bibliothek einsetzen. Eine selbst geschriebene Lernimplementierung kann den Algorithmus verständlich machen, braucht aber eine andere Einordnung als eine geprüfte Implementierung im Produkt.

Was ein Vergleich belegt

Angenommen, ich bewahre die Prüfsumme einer bestimmten PDF-Ausgabe auf. Später berechne ich den Hash der erhaltenen Datei erneut. Ein abweichender Wert zeigt, dass sie nicht dieselbe Bytefolge ist. Warum die Datei anders ist, bleibt offen: Ein Exportprogramm könnte nur Metadaten geändert haben; ebenso könnte der sichtbare Inhalt verändert worden sein.

Ein gleicher Wert ist unter den üblichen kryptografischen Annahmen ein starkes Indiz für gleiche Bytes. Er ist kein mathematischer Eindeutigkeitsbeweis für alle möglichen Dateien: Unterschiedliche Eingaben können denselben Ausgabewert besitzen. Die praktische Anforderung ist, solche Kollisionen nicht mit vertretbarem Aufwand gezielt finden zu können.

Entscheidend ist die Herkunft des Vergleichswerts. Wenn ein Angreifer Datei und daneben veröffentlichte Prüfsumme austauschen kann, stimmen beide wieder überein. Der Vergleich mit einer unabhängig vertrauenswürdigen Referenz beantwortet eine stärkere Frage als der Vergleich mit irgendeinem Wert auf derselben unbekannten Seite.

Grenzen, die im Projekt wichtig werden

Ein Hash verschlüsselt keine Daten. Einen privaten Inhalt zusätzlich zu hashen macht seine gespeicherte Originaldatei nicht vertraulich. Ebenso wird aus einem Hash allein keine digitale Signatur. Die Unterscheidung erkläre ich in [Hash, Verschlüsselung und Signatur](#).

Auch eine verkettete Protokolldatei braucht einen vertrauenswürdigen Bezugspunkt. Wer die ganze Kette neu schreiben und alle Hashwerte neu berechnen kann, kann eine in sich passende andere Geschichte erzeugen. Bei [belegwerk](#) ist diese Grenze für das Verständnis des Registers wichtig. Ein Hashverfahren übernimmt weder Quellenprüfung noch die Verantwortung für die ursprüngliche Eingabe.

Für die Speicherung von Passwörtern ist ein einzelner schneller SHA-256-Durchlauf ungeeignet. Dafür werden Verfahren mit Salt und bewusst erhöhtem Rechen- beziehungsweise Speicheraufwand verwendet. Die [OWASP-Empfehlungen zur Passwortspeicherung](#) behandeln diese Aufgabe gesondert.

Drei Fragen für die eigene Verwendung

1. Welche exakte Datei oder Bytefolge wird geprüft?
2. Woher kommt der vertrauenswürdige Referenzwert?
3. Welche Aussage soll der Vergleich tragen: unveränderte Bytes, Herkunft oder fachliche Richtigkeit?

Wenn die dritte Frage mehrere Antworten enthält, braucht der Ablauf meistens mehrere Prüfungen. Ein guter Einsatz von Kryptografie beginnt für mich damit, diese Erwartungen auseinanderzuhalten.

Quellen und Verwendung

NIST: Secure Hash Standard, FIPS 180-4
<https://csrc.nist.gov/pubs/fips/180-4/upd1/final>

Microsoft: Get-FileHash
<https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash>

OWASP: Password Storage Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Autor: Raphael Lugmayr. Diese Leseausgabe entwickelt ausgewählte Inhalte eigener Lehrunterlagen weiter. Sie ist eine Einführung mit Beispielen; die verlinkten Standards und Dokumentationen beschreiben die fachlichen Einzelheiten. Quellenstand: 18. September 2026.

Für Unterricht und Weiterempfehlungen bitte auf die Webfassung verweisen. Sie enthält den aktuellen Stand und zugehörige Materialien. Zitiervorschlag: Raphael Lugmayr (2026), SHA-256 verstehen, Version 1.0, raphaellugmayr.at. Rückmeldungen: raphael.lugmayr@stoicera.com.