Hash-Funktionen und Fingerprints

https://iuk.one/1010-1015

Clemens H. **Cap**
ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock, Germany
clemens.cap@uni-rostock.de

3. 1. 2021 Vers. 3

Ziel

(Kryptographische) Hash-Funktionen sind **eine Art Fingerabdruck**
ganz ähnlich dem Fingerabdruck eines Menschen.

- Man kann das Dokument nicht ändern, ohne den Fingerabdruck stark zu ändern.
- Aus dem Fingerabdruck kann man das Dokument nicht rekonstruieren.
- Der Fingerabdruck ist eine hinreichend gute **Charakterisierung** des Dokuments in dem **folgenden anschaulichen Sinn**:
  1. Es ist unwahrscheinlich, daß zwei Dokumente denselben Fingerabdruck haben.
  2. Wenn ich den Fingerabdruck eines Dokuments habe, dann kann mir niemand ein anderes Dokument mit demselben Fingerabdruck unterschieben, da ich das durch Prüfung des Fingerabdrucks erkennen kann.

Ziel dieses Abschnittes ist, diese Anschauung etwas präziser zu machen
und Anwendungen in der Kryptographie aufzuzeigen.

1. Klassische und kryptographische Hash-Funktionen

Ziele: Wir lernen die Grundbegriffe kennen und erste Beispiele. Wichtig ist die Unterscheidung von (klassischen, normalen) Hash-funktionen und kryptographischen Hash-Funktionen, bei denen deutlich mehr gefordert wird.

1. Klassische und kryptographische Hash-Funktionen

2. Anwendungen von Hash-Funktionen

Motivation: Warum brauchen wir normale Hash-Funktionen?

Informelle Beschreibung: Eine Hash-Funktion ist eine Funktion $h\colon D \to K$, die ein langes Dokument $d \in D$ auf einen sehr kurzen Hash-Wert $h(d)$ abbildet.

Oft: $K = \{0,1\}^n$ mit relativ kleinem n, oft im Bereich 10–30.

Beispiel: Sei $d = \alpha_0\alpha_1\alpha_2\ldots\alpha_q$ wobei α_j Buchstaben in 8-bit Darstellung sind.
Wähle etwa: $h(\alpha_0\alpha_1\alpha_2\ldots\alpha_q) = \alpha_0 + \alpha_1 + \ldots + \alpha_q$ **mod** 1024.
Es könnte sein: $h(\text{Deutsches Grundgesetz}) = 314$

Hash Funktionen entsprechen einer **Durchnummerierung** von Dokumenten.

Anwendung: Speicheradressen.

- Wir haben $2^{10} = 1024$ Speicher-Orte.
- Gegeben sei ein Dokument d.
- Lege das Dokument d in Speicherstelle $h(d)$ ab.

Beobachtung

Beobachtung 1: Geht für kleine Anzahl von Dokumenten.
Bei größerer Zahl von Dokus: Risiko von Kollisionen: $h(d_1) = h(d_2)$ für $d_1 \neq d_2$.

Lösung: Bei Dokumentenspeicherung Overflow-Bereiche einführen.

Idee: Mache die Länge der Hash-Werte größer: $K = \{0,1\}^{256}$
Wir haben nie so viele **sinnvolle** Dokumente – Kollisionsrisiko praktisch 0.

Aber: Wir haben nie so viel Speicherplatz.

Beobachtung 2: Kollisionen immer noch leicht zu finden
$h(\text{"Job begrüßte Alice ..."}) = h(\text{"Bob begrüßte Alice ..."}) + 8$
$h(\text{"Job begrüßte Alice ..."}) = h(\text{"Job begrüßte Adice ..."})$
Änderungen im Dokument schlagen vorhersagbar auf Hash-Werte durch.

Frage: Geht es auch anders?

Motivation für kryptographische Hash-Funktion

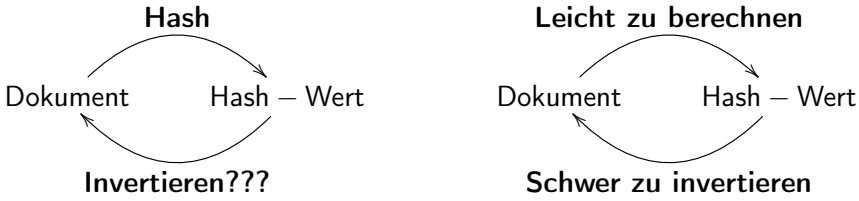

Abb. 1: Eine erste Motivation für Hash-Funktionen.

3 wichtige Eigenschaften kryptographischer Hash Funktionen

Kollisionsresistent

Es ist schwierig, **zwei verschiedene Dokumente** d_1, d_2 zu finden, die den **gleichen Hashwert** haben: $h(d_1) = h(d_2)$.

Schwer invertierbar (pre-image resistant)

Ist ein Wert w gegeben, dann ist es schwierig, ein Dokument d mit dem Hash-Wert w zu finden: $h(d) = w$.

Schwer zweite Urbilder zu finden (second pre-image resistant)

Ist ein Dokument d_1 gegeben, dann ist es schwer, ein weiteres Dokument d_2 mit dem selben Hash-Wert zu finden: $h(d_1) = h(d_2)$.

Typische Konstruktionsweise

n im Bereich 128–1024.

Lawinen-Prinzip: Wenn man ein Bit im Input-Dokument ändert, dann

- ändert sich typischerweise rund die Hälfte aller Hashwert-Bits – und
- ist es sehr schwer, vorherzusagen, welche Bits sich ändern.

Merkle-Damgard Konstruktion:

- Nehme an, Dokument bestehe aus k Blöcken der Länge n: $d = B_1 B_2 \dots B_k$
- Wähle Initialisierungsblock H_0
- Wähle Lawinen-artige Kompressionsfunktion $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.
- Verkette: $H_1 = f(H_0 \cdot B_1)$, $H_2 = f(H_1 \cdot B_2)$, $H_3 = f(H_2 \cdot B_3)$
- Letzter Wert ist dann Hash-Wert des Dokuments.

Beachte: $\cdot$ bedeutet **hier** die Konkatenation (das Hintereinanderschreiben) von Strings und bei RSA bedeutete es die Modulo-Multiplikation. Das sind unglücklicherweise weit verbreitete Überladungen von Bezeichnern.

Beispiele kryptographischer Hash-Funktionen

MD5:	128 Bit Wertebereich, gilt weithin als unsicher.
RIPEMD-128:	128 Bit, Kollisionen bekannt.
RIPEMD-320:	320 Bit, gilt noch als sicher.
SHA-0:	160 Bit Wertebereich, Kollisionen bekannt.
SHA-1:	160 Bit Wertebereich, Schwachstellen bekannt.
SHA-2:	Varianten mit 256 und 512 Bit und weitere.
SHA-3:	Varianten mit 256 und 512 Bit, neu seit 2015.

SHA-2 gilt als sicher gegen Angriffe durch Quanten-Computer,
erlaubt aber Angriffe durch Längenerweiterung.

SHA-3 verhindert Angriffe durch Längenerweiterung,
die Sicherheit gegen Quanten-Computer ist noch unbekannt.

Die **genauen** mathematischen Konstruktionen erscheinen konzeptuell zunächst einfach, sind im technischen Detail oft aber recht komplex ("Bit-Fummelei").

2. Anwendungen von Hash-Funktionen

Ziele: Wo werden Hash-Funktionen nun angewendet? Und sind Angriffe gegen solche Anwendungen bekannt?

Anwendung in der Integritäts-Sicherung einer Nachricht (1)

Situation: Integrität, aber nicht unbedingt Vertraulichkeit.

- Alice sendet eine Nachricht an Bob.
- **Schutzziel Integrität:** Angreifer soll Nachricht nicht unbemerkt verändern können.
- **Kein Schutzziel Vertraulichkeit:** Jeder darf die Nachricht lesen.

Beachte:

- Digitale Unterschrift würde das leisten, würde aber – unerwünscht – mehr leisten.
- Denn: Jeder könnte Alice das Schreiben der Nachricht nachweisen.

Lösung: Sogenannter MAC (**m**essage **a**uthentication **c**ode)

- Alice und Bob einigen sich auf eine pre-image resistente Hash-Funktion h.
- Alice und Bob vereinbaren ein geheimes Präfix p.
- Alice sendet das Dokument d und den Hash-Wert $m = h(p \cdot d)$.
- Bob erhält das Dokument d' und den Hash-Wert m'.
- Bob prüft, ob $h(p \cdot d') = m'$.

Anwendung in der Integritäts-Sicherung einer Nachricht (2)

Angriff:

- Mallory verändert d zu $\tilde{d}$. Wie kann sie nun m anpassen?
- Gegeben h, d, $h(p \cdot d)$ und $\tilde{d}$. Gesucht ist $h(p \cdot \tilde{d})$ (ohne Kenntnis von p).

Analyse:

- Da h pre-image resistent, ist es schwer, zu y ein x zu finden mit $y = h(x)$.
- Idee: Zu y und d ist es dann auch schwer, ein p zu finden mit $y = h(p \cdot d)$ und dieses p braucht man vermutlich?! Also dürfte ein Angriff schwer sein?!

Achtung: Das stimmt, so formuliert, leider nicht.

Anwendung in der Integritäts-Sicherung einer Nachricht (3)

Beispiel 1: Wähle $y = h(1 \cdot d)$, dann muß man nicht lange durchprobieren.

Beispiel 2: Wähle $y = h(1111111111111111111 \cdot d)$, dann muß man nicht lange durchprobieren, wenn man weiß, daß Alice solche Präfixe wählt.

Beispiel 3: Wähle $y = h(10100010111010111101010001111010100001111101010100001 \cdot d)$, falls h eine solche Hash-Funktion ist, daß bei diesem spezifischen Präfix p_0 immer $h(p_0 \cdot d) = h(d)$ ist. Eine solche Eigenschaft ist zwar unwahrscheinlich, aber nicht ausgeschlossen und folgt auch nicht streng mathematisch!

Problem: Die allgemeine Bedingung der pre-image Resistenz genügt nicht.

Fazit 1: Es sind Angriffe auf diesen Mechanismus denkbar.

Fazit 2: Wir müssen eigentlich die Definition der Hash-Funktion nachrüsten!
Da mathematisch aufwendig, tun wird das nur in fortgeschritteneren Veranstaltungen

Längen-Erweiterungs-Angriff

Viele etablierte Hash-Funktionen (SHA-1, SHA-2, MD-5) nutzen die Merkle-Damgard Konstruktion.
Diese Konstruktion erlaubt es einem Angreifer

1. das Dokument durch ein Suffix s zu erweitern: $d \mapsto \tilde{d} = d \cdot s$ und dann
2. den Hash-Wert $h(\tilde{d})$ des längeren Dokuments aus dem Suffix s und dem abgefangenen Hash-Wert $h(d)$ des kürzeren Dokuments zu berechnen.

Sei, als einfaches Beispiel, $d = B_1 \cdot B_2$.

Dann ist nach Merkle-Damgard $h(d) = f(f(H_0 \cdot B_1) \cdot B_2)$

Angreifer erweitert d zu $\tilde{d} = B_1 \cdot B_2 \cdot B$.

Angreifer kennt $h(d)$, also auch $f(f(H_0 \cdot B_1) \cdot B_2)$ (das ist $h(d)$).

Angreifer kennt aus den Standardisierungs-Dokumenten h und daher auch f.

Angreifer berechnet daraus $h(\tilde{d}) = f(\underbrace{f(f(H_0 \cdot B_1) \cdot B_2)}_{\text{Dem Angreifer bekannt}} \cdot B)$

Abwehr des Längen-Erweiterungs-Angriffs

Lösung 1: Bessere Hash-Funktion nutzen.

- Eine Hash-Funktion verwenden, die nicht nach Merkle-Damgard konstruiert ist.
- Beispiel: SHA-3.

Lösung 2: Besseren Mechanismus zur Berechnung des MAC nutzen.

- Beispiel: Der sog. HMAC mit verschachtelter Hash-Funktion: $h(k_1 \cdot h(k_2 \cdot x))$

Fazit: Kryptographie erfordert

- die Kenntnis des laufenden Fachdialogs – oder
- die Nutzung von Resultaten aus dem laufenden Fachdialog.

"Rant": Unternehmen, die "eigene" Kryptographie ohne tiefere Fachkenntnis anbieten, gibt es am Markt leider immer noch (sehr / zu) viele.

Weitere Anwendungen

Digitale Unterschriften:

- Unterschreibe Hash-Wert $h(d)$ des Dokuments und nicht das Dokument d selber.
- **Grund 1:** Blocklänge bei asymmetrischen Algorithmen eingeschränkt.
- **Grund 2:** Zusätzliche Sicherheit (etwa: gegen Homomorphie-Eigenschaften).

Kryptographische Bindung:
Alles, was bestätigt werden soll, in eine große Hash-Funktion "reinstecken".
Beispiel: Wenn Partner $h(\text{Kontonummer} \cdot \text{Betrag})$ kennt, dann kann Angreifer zwar nicht die Kontonummer oder den Betrag unbemerkt ändern, aber das Überweisungsdatum oder die Zahlungsreferenz.

Anhang

Übersicht

Verzeichnis aller Abbildungen

Rechtliche Hinweise

Die hier angebotenen Inhalte unterliegen deutschem Urheberrecht. Inhalte Dritter werden unter Nennung der Rechtsgrundlage ihrer Nutzung und der geltenden Lizenzbestimmungen hier angeführt. Auf das Literaturverzeichnis wird verwiesen. Das Zitatrecht in dem für wissenschaftliche Werke üblichen Ausmaß wird beansprucht. Wenn Sie eine Urheberrechtsverletzung erkennen, so bitten wir um Hinweis an den auf der Titelseite genannten Autor und werden entsprechende Inhalte sofort entfernen oder fehlende Rechtsnennungen nachholen. Bei Produkt- und Firmennamen können Markenrechte Dritter bestehen. Verweise und Verlinkungen wurden zum Zeitpunkt des Setzens der Verweise überprüft; sie dienen der Information des Lesers. Der Autor macht sich die Inhalte, auch in der Form, wie sie zum Zeitpunkt des Setzens des Verweises vorlagen, nicht zu eigen und kann diese nicht laufend auf Veränderungen überprüfen.

Die Bereitstellung der Inhalte erfolgt hier zur persönlichen Information des Lesers. Eine Haftung für mittelbare oder unmittelbare Schäden wird im maximal rechtlich zulässigen Ausmaß ausgeschlossen, mit Ausnahme von Vorsatz und grober Fahrlässigkeit. Eine Garantie für den Fortbestand dieses Informationsangebots wird nicht gegeben.

Die Anfertigung einer persönlichen Sicherungskopie für die private, nicht gewerbliche und nicht öffentliche Nutzung ist zulässig, sofern sie nicht von einer offensichtlich rechtswidrig hergestellten oder zugänglich gemachten Vorlage stammt.

Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen,
zitieren Sie es bitte wie folgt:

Clemens H. Cap: Hash-Funktionen und Fingerprints. Electronic document. https://iuk.one/1010-1015
3. 1. 2021.

Bibtex Information: https://iuk.one/1010-1015.bib

```
@misc{doc:1010-1015,
   author       = {Clemens H. Cap},
   title        = {Hash-Funktionen und Fingerprints},
   year         = {2021},
   month        = {1},
   howpublished = {Electronic document},
   url          = {https://iuk.one/1010-1015}
}
```

Typographic Information:
Typeset on January 3, 2021
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

Verzeichnis aller Folien

Legende:
Fortsetzungsseite
Seite ohne Überschrift
Bildseite