Kapitel 12: Hashing, MAC und digitale Signaturen

Hash-Funktionen

- Es sei k ∈ K ein Schlüssel aus einem **Schlüsselraum** K ⊂ Z.
- Sei a ∈ A = {0, 1, . . . , m − 1} ein **Hash-Wert** aus der Menge A. (a ist Index in einer Tabelle mit m Plätzen).
- Die Hash-Werte kennzeichnen die **Adressen der Datensätze** in der **Hash-Tabelle**.
- Die Anzahl m der Elemente aus A ist in der Regel **sehr viel kleiner** als die mögliche Anzahl |K| von Schlüsseln aus K.
- Eine Hash-Funktion h ist eine **Abbildung** (Funktion)
- h:K→A, k ↦h(k), a=h(k).
- Das **Datenelement** mit dem **Schlüssel** k wird dann an der **Hash-Adresse** (also dem Index) a = h(k) gespeichert.

Hash-Funktionen: Allgemeines Prinzip

- Eine **Hash-Funktion** h erhält das zu speichernde Element k als Eingabe (Argument) und berechnet daraus einen Hash-Wert a = h(k), also eine Adresse.
- Hash-Werte sind **positive ganze Zahlen** aus einem gegebenen Intervall [0,M].
- Die **Zielmenge** (Menge aller Hash-Werte) einer Hash-Funktion ist deutlich kleiner (weniger mächtig) als die **Eingabemenge** (Menge aller Schlüsselwerte).

Ein Beispiel:

- *Eingabemenge K: Alle natürlichen Zahlen im Intervall [0,999] Zielmenge A: Alle natürlichen Zahlen im Intervall [0, 27] Hash-Funktion h: Berechne die Quersumme der Zahl*
- *h(5) = 5, h(14) = 5, h(93) = 12, etc.*

Eigenschaften von Hash-Funktionen

- Eine Hash-Funktion soll schnell und einfach zu berechnen sein.
- Eine Hash-Funktion streut möglichst gut auf die Elemente der Zielmenge (Hash-Adressen).
- Die Mächtigkeit der Zielmenge |A| (Anzahl der Hash-Adressen) ist im Allgemeinen wesentlich geringer als die der Eingabemenge |K|. Hash-Funktionen sind im Allgemeinen **nicht injektiv**, es können zwei unterschiedliche Schlüssel k1≠ k2 zu demselben Hash-Wert h(k1) = h(k2), also zum selben Feld in der Tabelle führen.
- Dieses Ereignis wird als Kollision bezeichnet.
    - In diesem Fall muss die Hashtabelle **mehrere** Werte an derselben Stelle aufnehmen.
- Um dieses Problem zu behandeln, existieren diverse Strategien zu Kollisionsauflösung.

Kollisionsauflösung durch geschlossenes Hashing

- Wenn ein Eintrag an eine schon belegte Hash-Adresse in der Hash-Tabelle kommen soll, wird stattdessen eine andere freie Stelle gesucht und belegt. Dabei werden drei Varianten unterschieden:

1) **Lineares Sondieren:**
   Bei einer Kollsion wird um ein konstantes Intervall verschoben und dort nach einer freien Stelle gesucht. Meistens wird die Intervallgröße auf 1 festgelegt. Folge: 1,2,3,4,...
2) **Quadratisches Sondieren**:
   Bei einer Kollision werden die Quadrate der Intervallgrößen benutzt. Folge: ~~12 = 1,22 = 4,32 = 9,42 = 16,...~~.
3) **Doppeltes Hashen**:
   Bei einer Kollision wird eine weitere Hash-Funktion benutzt, die dann das Intervall liefert .

- Problem: Beim Löschen eines Eintrags muss der Tabellenplatz speziell markiert werden, damit dahinter stehende Einträge noch gefunden werden können.

Kollisionauflösung durch offenes Hashing

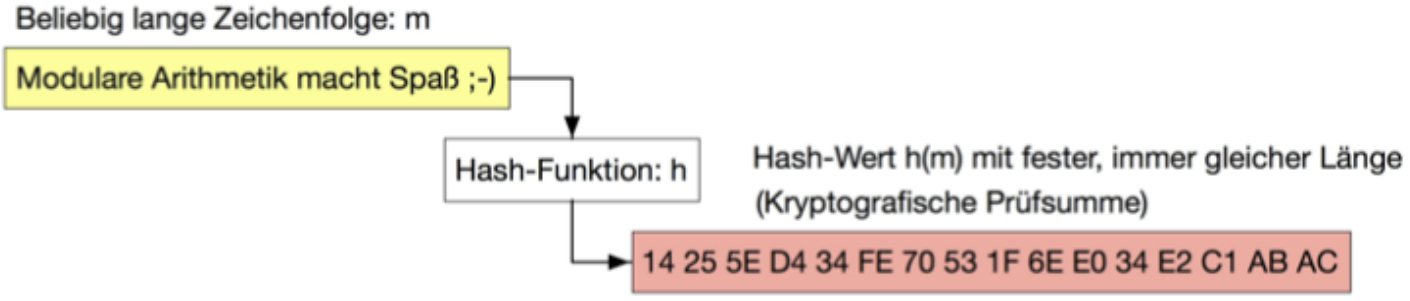
Statt der gesuchten Daten enthält die Hashtabelle hier Behälter (engl. buckets), die alle Daten mit gleicher Hash-Adresse aufnehmen.
Das kann eine einfache verkettete Liste sein, in der alle Datensätze hinter einander gehängt werden.
Suche nach einem Datensatz besteht aus zwei Schritten:

1) Berechne des Hashwert h des Datensatzes.
2) Durchsuche den Behälter an Position h der Tabelle.

Einfügen analog.
In der Praxis dürfen die Behälter nicht zu groß werden, denn sonst werden die Geschwindigkeitsvorteil der Hashtabelle verloren.

Kryptografische Hash-Funktionen

- Kryptografische Hash-Funktionen generieren aus beliebig langen Datensätzen eine Zeichenfolge mit einer festen Länge.
- Ein Datensatz kann ein Wort, ein Satz, ein längerer Text oder auch eine ganze Datei sein.
- Die erzeugte Zeichenfolge wird als **digitaler Fingerabdruck** (engl. fingerprint), **kryptografische Prüfsumme** oder auch **Message Digest (MD)** bezeichnet.



Anforderungen an kryptographische Hashfunktionen

- Kollisionsresistenz bedeutet, dass zwei unterschiedliche Zeichenfolgen m1≠ m2 nicht den gleichen Hash-Wert haben dürfen. ($h(m_1) \neq h(m_2)$).
- Da die Eingabemenge der Hash-Funktion größer ist als die Menge der möglichen Hash-Werte sind Kollisionen nicht grundsätzlich zu vermeiden. Anforderung daher: Die Wahrscheinlichkeit, dass zwei unterschiedliche Eingaben $m_1 \neq m_2$ denselben Hash-Wert haben ist praktisch gleich 0.
- Daraus folgt, dass die Menge der möglichen Hash-Werte ausreichend groß sein muß. (Bei nur 25 möglichen Hash-Werten gibt es bereits bei nur 26 verschiedenen Eingaben bereits sicher eine Kollision!)
- Heute werden Hash-Werte gewählt, die mindestens 256 Bit lang sind, so dass es 2^{256} viele verschiedene Hash-Werte gibt.
- Hash-Funktionswerte sollten wie zufällig aussehen und kleine Änderungen an der Eingabe sollen zu großen Änderungen an der Ausgabe führen.

Konkretisierung der „kryptographischen" Anforderungen

Schwache Kollisionsresistenz:

- Zu einer gegebenen Zeichenfolge m_1 ist es praktisch unmöglich, eine Zeichenfolge $m_2 \neq m_1$ zu finden, für die $h(m_1) = h(m_2)$ gilt.

Starke Kollisionsresistenz:

- Es ist praktisch unmöglich, zwei beliebige Zeichenfolgen m1 und m2 ($m_1 \neq m_2$) zu finden, für die $h(m_1) = h(m_2)$ gilt.

Einwegfunktion:

- Zu einem gegebenen Hash-Wert a ist es praktisch unmöglich, eine Zeichenfolge m zu finden, für die h(m) = a gilt.

Geburtstagsparadoxon

Gesucht ist die Wahrscheinlichkeit p(x), dass von n Personen mindestens 2 am selben Tag geboren sind (das Jahr wird nicht berücksichtigt). Die Berechnung erfolgt leichter über die **Gegenwahrscheinlichkeit**: Wir fragen nach der Wahrscheinlichkeit, dass **alle** n **Personen** an einem **anderen Tag** geboren sind. Dazu verwenden wir den Multiplikationssatz. Die Wahrscheinlichkeit, dass die **erste** Person an **irgendeinem** der 365 Tage geboren ist ist:

- 1 = 356/356

Die Wahrscheinlichkeit, dass die zweite Person an einem anderen Tag als die erste Person geboren ist:

- 364/ 365

Die Wahrscheinlichkeit, dass die dritte Person an einem anderen Tag als die erste und die zweite Person geboren ist:

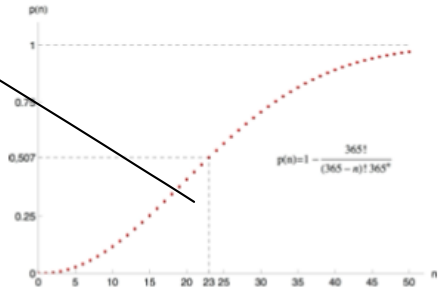
- 363/365

Die Wahrscheinlichkeit, dass die k-te Person an einem anderen Tag als die k – 1 anderen Personen geboren ist ist:

- (365−k +1)/365

Daher ist die gesuchte Wahrscheinlichkeit p(n):

$$p(n) = 1 - \prod_{k=1}^{n} \frac{365 - k + 1}{365} = 1 - \frac{365!}{(365 - n)!\, 365^n}$$



Die Wahrscheinlichkeit dafür, dass unter n = 23 Personen zwei am selben Tag geboren sind beträgt: p(23) = 0.507.

Hash-Algorithmen: MD5

- Das Verfahren heißt Message-Digest-Algorithm 5 (MD5).
- Maximale Nachrichten-Länge: 264 Bytes.
- MD5 erzeugt aus einer Nachricht einen Hash-Wert mit der Länge von 128 Bit. 1996 fand Hans Dobbertin (BSI) eine Kollision für zwei unterschiedliche Nachrichten.
- Es handelte sich um eine echte Kollision, mit zwei speziell präparierten Nachrichten, die sich unterscheiden, aber dennoch denselben Hashwert ergeben (allerdings bei einer etwas modifizierten MD5-Variante).
- MD5 gilt inzwischen als nicht mehr als sicher, da es mit überschaubarem Aufwand möglich ist, unterschiedliche Nachrichten zu erzeugen, die den gleichen MD5-Hashwert aufweisen.

Hash-Algorithmen: SHA-Familie

Die Familie heißt Secure Hash Algorithm und der Name bekommt eine Nummer angehängt.
SHA-1 wurde vom National Institute of Standards and Technology (NIST) zusammen mit der National Security Agency (NSA) im Jahr 1995 entwickelt und standardisiert (RFC 3174).
Die maximale Länge der Eingabe ist auf $2^{64} - 1$ Byte begrenzt.
SHA-1 erzeugt aus beliebigen Daten einen Hash-Wert von 160-Bit Länge.
Die SHA-2-Familie (SHA-224, SHA-256, SHA-384, SHA-512) arbeitet nach demselben Prinzip wie SHA-1, liefert aber längere Hash-Werte.
SHA-3 wurde vom (NIST) im Jahr 2015 als Nachfolger von SHA-2 standardisiert.
SHA-3 arbeitet nach einem anderen Prinzip (genannt sponge construction) und erzeugt aus beliebigen Daten einen Hash-Wert von 256-Bit Länge.

Einsatz von Hashing: Kryptografischer Fingerabdruck
Beispiel: SHA-2-256
$ echo "Maja" | shasum -a 256
12907cc009887a4e893c4b9b14df0397416f528f04949a68d27b9213e4e64912
$ echo "Mara" | shasum -a 256
daff34f5f78a3092086a173eca2b2c72e0b5ef29ba8d84a98f654bf51ed9e6f0

Message Authentication Codes (MAC)

Einsatzgebiete/Aufgaben:

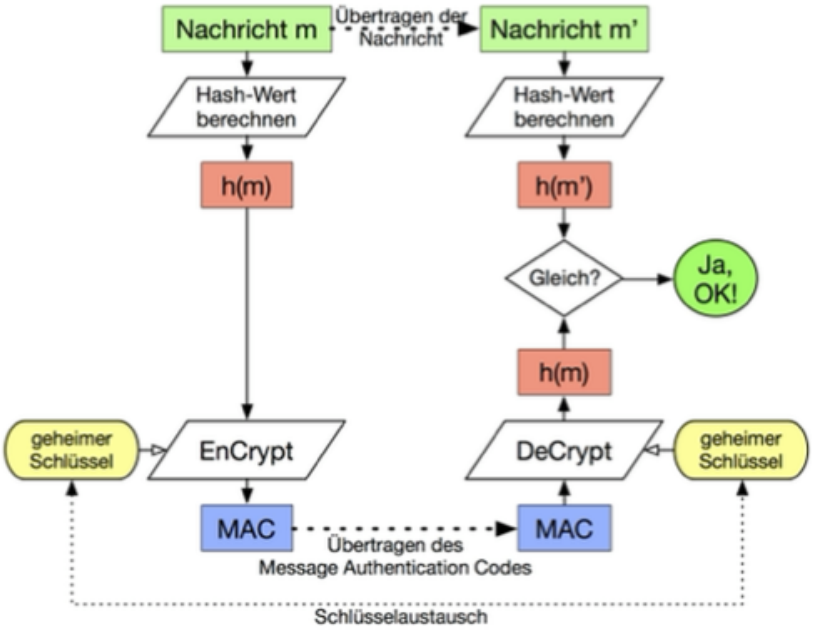
- Überprüfung der **Integrität** der Daten.
- Gewissheit über den **Ursprung** von Daten (mit Einschränkungen).

Problem: Eine kryptographische Prüfsumme kann jede*r berechnen

Lösung: Lasse zusätzlich ein Geheimnis in die Berechnung eingehen, welches nur Sender*in und Empfänger*in bekannt sind.

Beispiel: Keyed-Hash Message Authentication Code (HMAC)

- standardisiert in RFC 2104
- wird verwendet in IPsec, TLS, SSH, etc.



- **Schutzziel**: Die Integrität von Daten
- MAC's sichern der **Integrität** von Daten
  - Wird die Nachricht bei der Übertragung verändert, so kann der Empfänger dies bemerken.
  - Ohne Kenntnis des geheimen Schlüssels kann kein Angreifer eine Nachricht unbemerkt verändern
- Message Authentication Codes können **keine Authentizität** sichern!
  - Der Absender kann bestreiten, dass die Nachricht von ihm ist, denn der Empfänger könnte sie auch selbst erstellt haben.
- **Non repudiation** (Nichtabstreitbarkeit (Zurückweisung) der Urheberschaft): Eine stattgefundene Kommunikation kann von den beteiligten Parteien im Nachhinein nicht mehr abgestritten werden.

Prinzip von digitalen Signaturen

Einsatzgebiete:

- Überprüfung der **Integrität** der Daten.
- Sicherstellung der **Authentizität** des Absenders.

Problem: Nur genau eine Person soll eine digitale Signatur (DS) erstellen können.

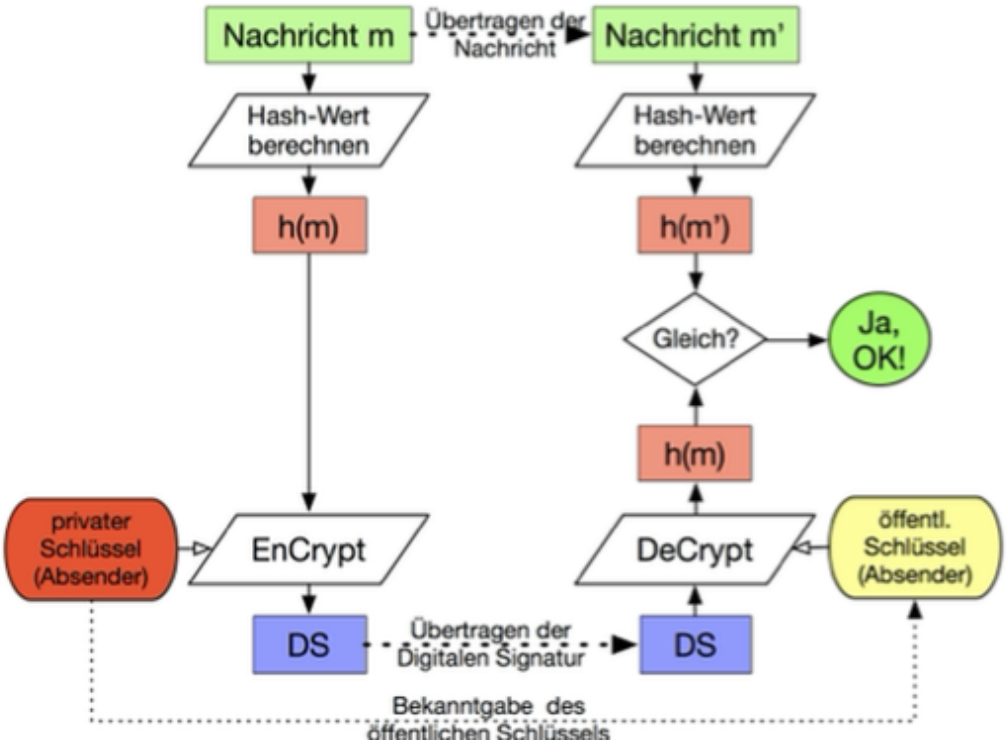
Berechnungsidee: Benutze zur Berechnung der DS ein Geheimnis, das nur der „Unterschreiber“ kennt.

Beispiele:

- Nehme eine mit dem privaten Schlüssel des Unterschreibers asymmetrisch verschlüsselte kryptographische Prüfsumme als Digitale Signatur.
- DS(x) = Crypt(privaterSchlüssel,h(x))

Überprüfung der digitalen Signatur:

- Benutze den öffentlichen Schlüssel des Unterschreibers.



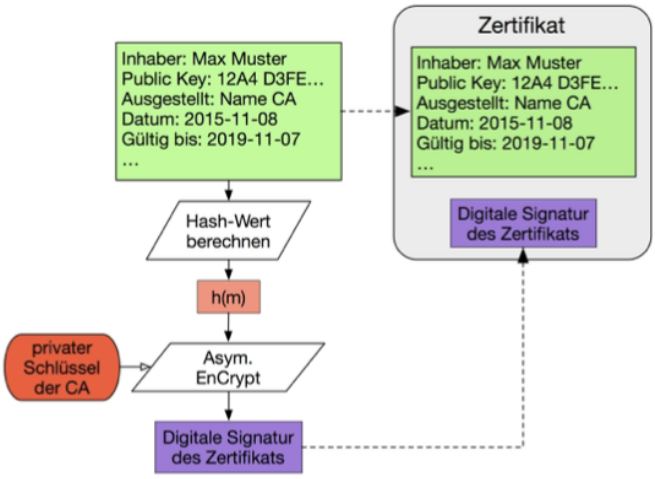
Eigenschaften von digitalen Signaturen

- **Erreichte Schutzziel**(e):
  - **Integrität** und
  - **Authentizität** von Daten (Nachrichten).
- Warum ist die Integrität gesichert?
  - Nur wenn der Empfänger **denselben Hash-Wert** berechnet, wie der verschlüsselt übertragene Hash-Wert, dann ist die Nachricht unverändert.
  - Ohne Kenntnis des **privaten Schlüssels** des **Absenders** kann kein Angreifer eine Nachricht unbemerkt **verändern**.
- Warum ist die Authentizität gesichert?
  - Die Nachricht kann nur von der Person stammen, die den privaten Schlüssel kennt. Niemand sonst könnte die kryptographische Prüfsumme so verschlüsseln.
- **Noch offenes Problem:**
  - Woher weiß man, dass ein öffentlicher Schlüssel tatsächlich zu einer bestimmten Person gehört?

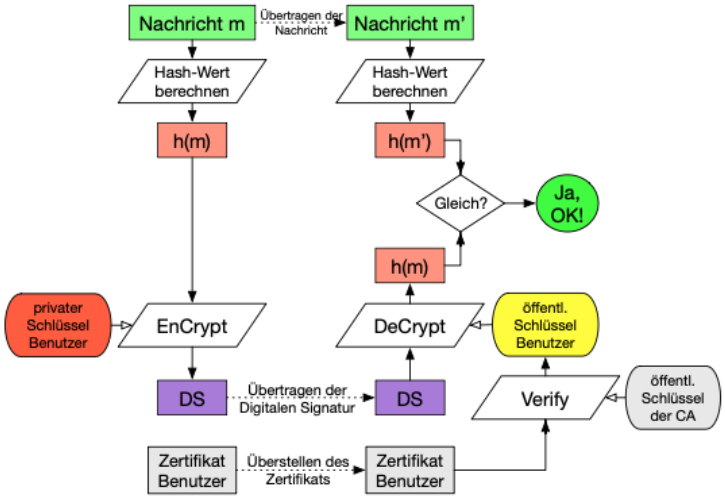
Prinzip und Idee von Zertifikaten

- **Zertifikate** sollen eine dem Personalausweis vergleichbare Funktion besitzen. Sie erfüllen also die folgenden Aufgaben:
  - Die Herstellung einer Verbindung zwischen einer Person und einem Namen (Feststellen der Identität).
  - Dabei soll ein Zertifikat sehr schwer zu fälschen sein.
- Der **Aussteller** eines Zertifikats muss eine vertrauenswürdige Instanz sein.
- Ein **Zertifikat** ist letztlich nichts anderes als ein signierter Datensatz mit den folgenden Einträgen:
  - Name des Inhabers (plus evtl. Telefon, Email, usw.)
  - Öffentlicher Schlüssel des Inhabers
  - Seriennummer
  - Gültigkeitsdauer
  - Name der Zertifizierungsstelle (Certification Authority (CA))

Aussehen eines Zertifikates



Ablauf beim Einsatz von Zertifikaten



Eigenschaften von Zertifikaten

- Mit **Zertifikaten** kann die **Integrität** von **Nachrichten** sichergestellt werden.
- Mit **Zertifikaten** kann die **Authentizität** von **Nachrichten** sichergestellt werden:
  - Der **Empfänger** kann **überprüfen**, ob der **Absender** der Nachricht mit dem **Namen des Inhabers** des **Zertifikats** übereinstimmt.
- **Problem**:
  - Wie wird der öffentlichen Schlüssel der Zertifizierungsstelle erlangt?
- **Lösung**:
  - Die Zertifizierungsstelle besitzt selbst auch ein Zertifikat!

Zertifizierungsstellen

- Es gibt eine **Wurzel-Zertifizierungsstelle**, die die **Zertifikate** für die **Zertifizierungsstellen** erstellt.
- In Deutschland ist das die **Bundesnetzagentur** www.bundesnetzagentur.de
- Das **Signaturgesetz** legt fest, welche **Anforderungen** eine **Zertifizierungsstelle erfüllen** muss
  - Technische Anforderungen (z. B. Sichere Systeme, geschützte Gebäude, usw.).
  - Organisatorische Anforderungen (z. B. Vier-Augen-Prinzip).
- Die Bundesnetzagentur hat folgenden Vertrauensdiensteanbieter die Qualitfikation erteilt, qualifizierte Zertifikate für elektronische Signaturen zu erstellen: Bundesagentur für Arbeit, Bundesnotarkammer, Deutsche Post AG, D-Trust GmbH, Deutsche Telekom AG, DGN Deutsches Gesundheitsnetz Service GmbH und medisign GmbH.