Kryptografische Hash Funktionen

Dozent: Prof. Dr. Michael Eichberg

Kontakt: michael.eichberg@dhbw-mannheim.de

Basierend auf: *Cryptography and Network Security - Principles and Practice, 8th Edition, William Stallings*

Version: 1.0

Folien: **HTML:** https://delors.github.io/sec-hashfunktionen/folien.de.rst.html
PDF: https://delors.github.io/sec-hashfunktionen/folien.de.rst.html.pdf

Fehler auf Folien melden:
https://github.com/Delors/delors.github.io/issues

1. Hashfunktionen - Grundlagen

Prof. Dr. Michael Eichberg

Hashfunktionen

- Eine Hashfunktion H akzeptiert eine beliebig lange Nachricht M als Eingabe und gibt einen Wert fixer Größe zurück: $h = H(M)$.
- Wird oft zur Gewährleistung der Datenintegrität verwendet. Eine Änderung eines beliebigen Bits in M sollte mit hoher Wahrscheinlichkeit zu einer Änderung des Hashwerts h führen.
- Kryptographische Hashfunktionen werden für Sicherheitsanwendungen benötigt. Mögliche Anwendungen:
  - Authentifizierung von Nachrichten
  - Digitale Signaturen
  - Speicherung von Passwörtern

Beispiel: Berechnung von Hashwerten mittels MD5

```
md5("Hello") = 8b1a9953c4611296a827abf8c47804d7
md5("hello") = 5d41402abc4b2a76b9719d911017c592
md5("Dieses Passwort ist wirklich total sicher
      und falls Du es mir nicht glaubst, dann
      tippe es zweimal hintereinander blind
      fehlerfrei ein.")
            = 8fcf22b1f8327e3a005f0cba48dd44c8
```

Warnung

Die Verwendung von MD5 dient hier lediglich der Illustration. In realen Anwendung sollte MD5 nicht mehr verwendet werden.

Sicherheitsanforderungen an kryptografische Hashfunktion I

Variable Eingabegröße:

H kann auf einen Block beliebiger Größe angewendet werden.

Pseudozufälligkeit:

Die Ausgabe von H erfüllt die Standardtests für Pseudozufälligkeit.

Einweg Eigenschaft:

Es ist rechnerisch/praktisch nicht machbar für einen gegeben Hashwert h ein N zu finden so dass gilt: $H(N) = h$

(🇺🇸 *Preimage resistant; one-way property*)

Sicherheitsanforderungen an kryptografische Hashfunktion II

Schwache Kollisionsresistenz:

Es ist rechnerisch nicht machbar für eine gegebene Nachricht M eine Nachricht N zu finden so dass gilt: $M \neq N$ mit $H(M) = H(N)$

(🇺🇸 *Second preimage resistant; weak collision resistant*)

Starke Kollisionsresistenz:

Es ist rechnerisch unmöglich ein paar (N, M) zu finden so dass gilt: $H(M) = H(N)$.

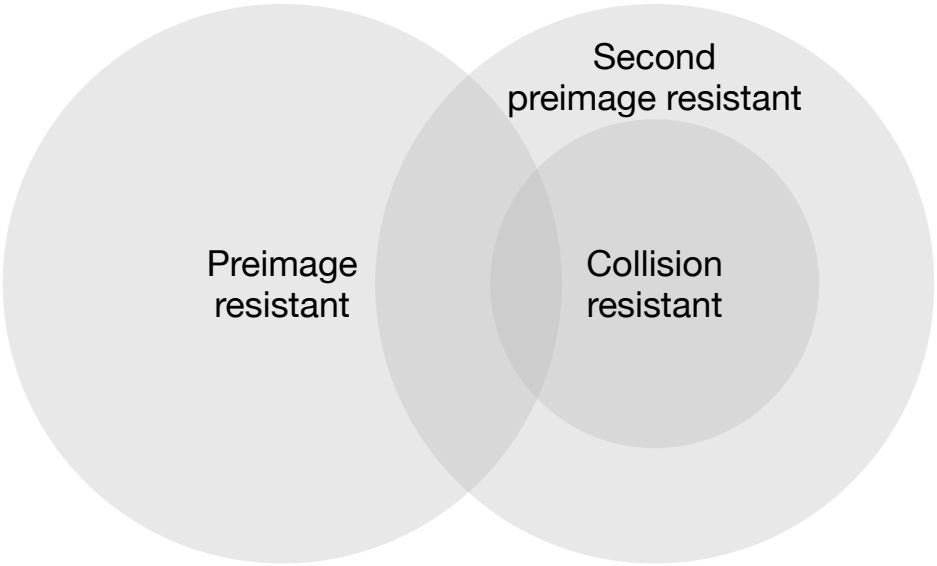
(🇺🇸 *Collision resistant; strong collision resistant*)

Hintergrund

Im Deutschen wird auch von Urbild-Angriffen gesprochen. In dem Fall ist *preimage resistance* (d.h. die Einweg Eigenschaft) gleichbedeutend damit, dass man nicht effektiv einen „Erstes-Urbild-Angriff" durchführen kann. Hierbei ist das Urbild die ursprüngliche Nachricht M, die *gehasht* wurde.

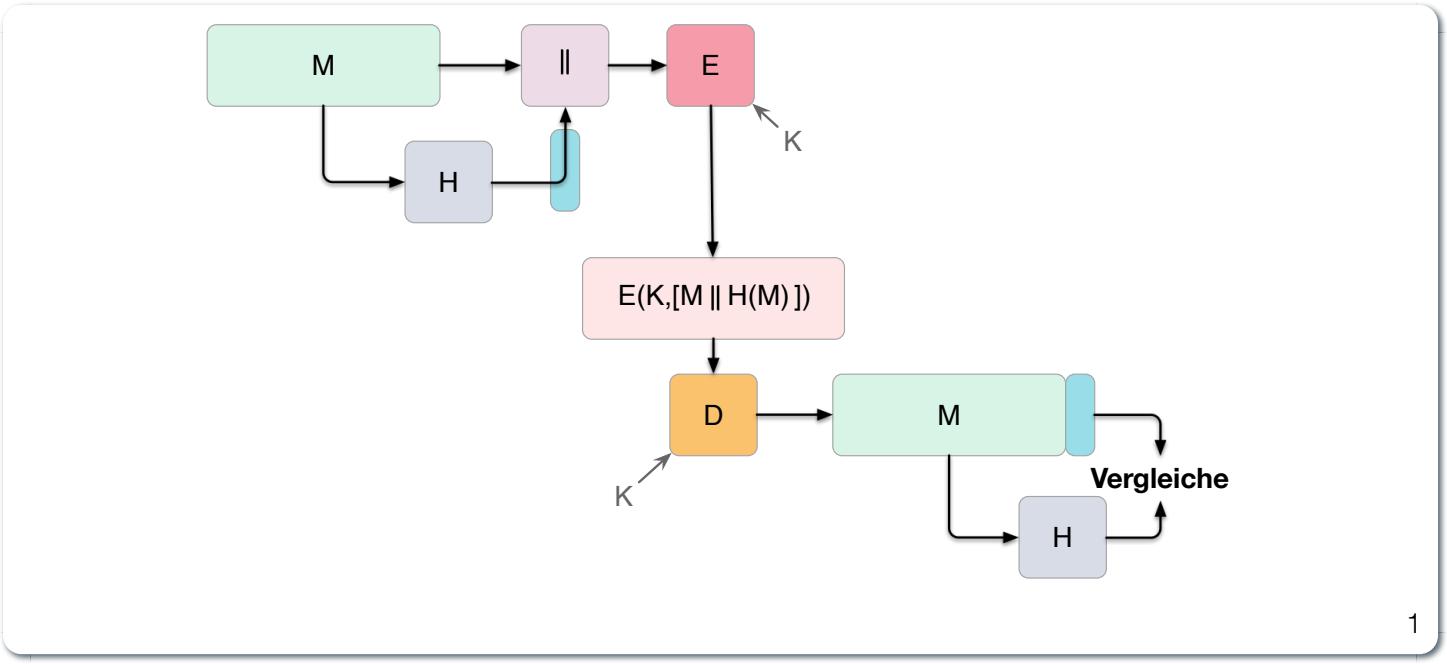
Second preimage resistance ist dann gleichbedeutend damit, dass man nicht effektiv einen „Zweites-Urbild-Angriff" durchführen kann. Es ist nicht möglich zu einer Nachricht M eine zweite Nachricht N (d.h. ein zweites Urbild) zu finden, die für eine gegebene Hashfunktion den gleich Hash aufweist.

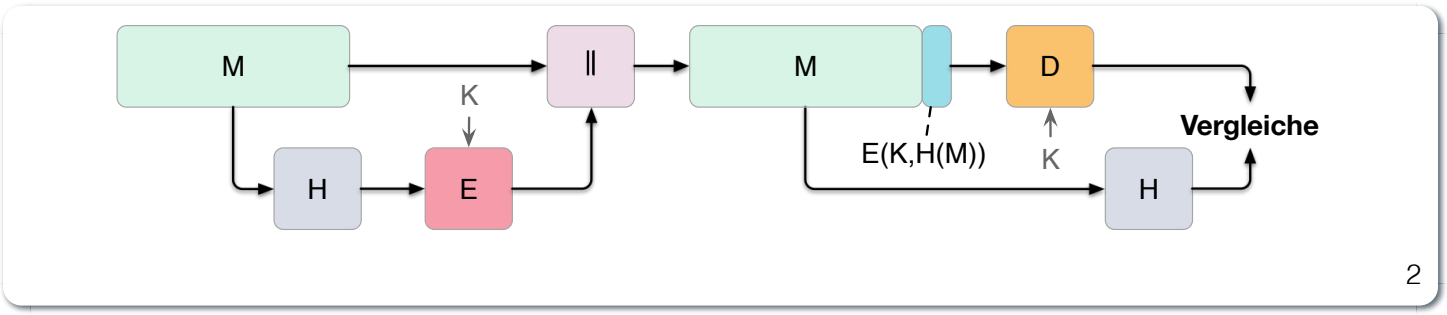
Beziehung zwischen den Sicherheitsanforderungen an Hashfunktionen

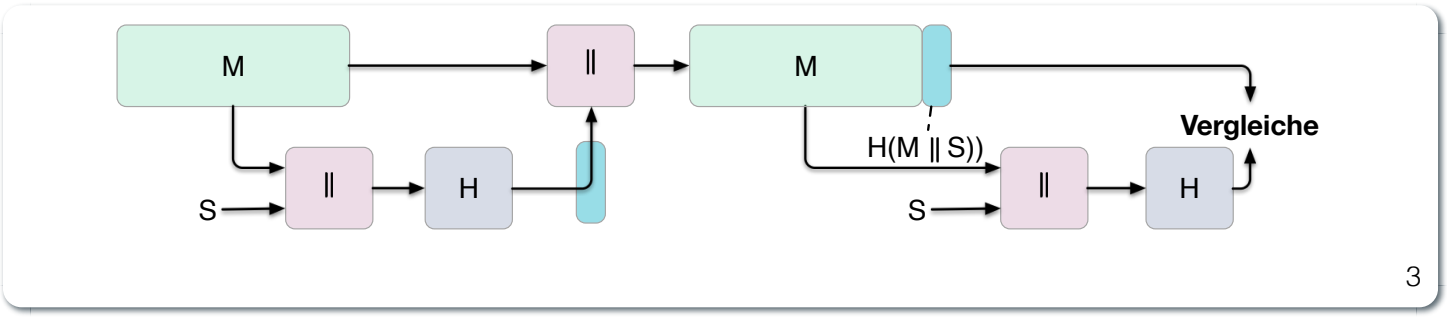


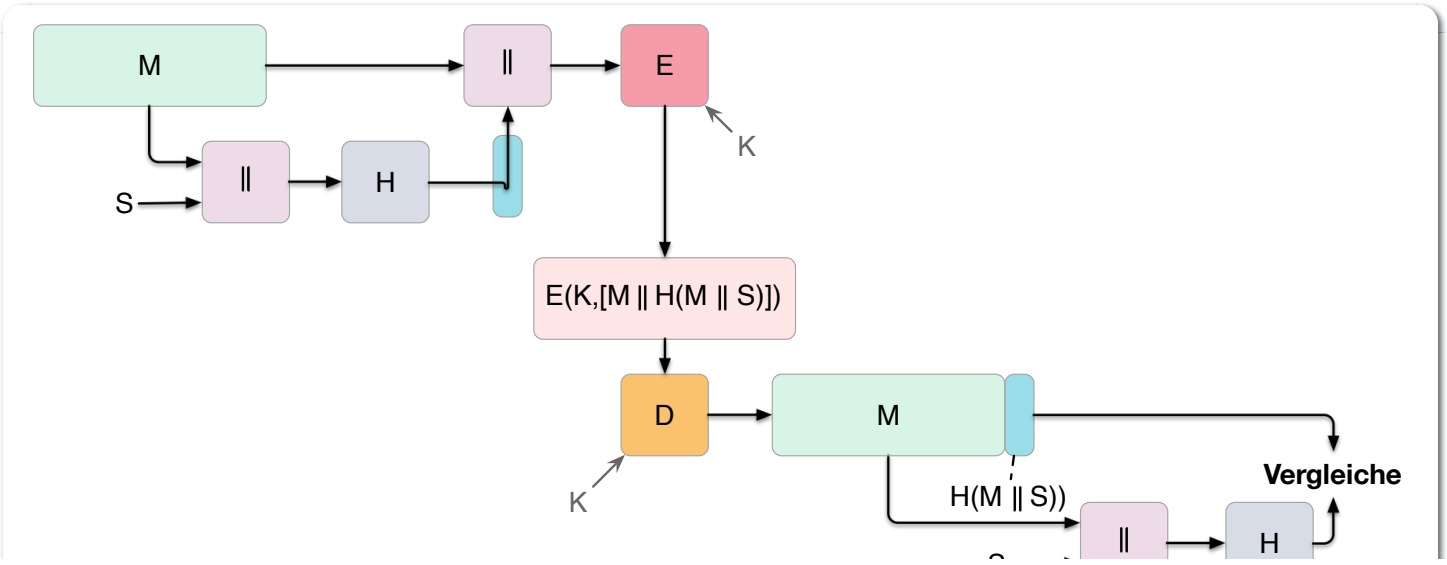
Nachrichtenauthentifizierung - vereinfacht

Nachrichten können auf verschiedene Weisen authentifiziert werden, so dass *Man-in-the-Middle-Angriffe* (MitM)[1] verhindert werden können.









4

[1] *Man* ist hier geschlechtsneutral zu verstehen.

8

Im ersten Szenario wird der Hash an die Nachricht angehängt und als ganzes verschlüsselt. Wir erhalten Vertraulichkeit und Authentizität.

Im zweiten Szenario wird der Hash der Nachricht berechnet und dann verschlüsselt. Der Empfänger kann den Hash berechnen und mit dem entschlüsselten Hash vergleichen. Wir erhalten Authentizität, aber keine Vertraulichkeit.

Im dritten Szenario wird an die Nachricht ein geteiltes Secret angehängt und alles zusammen gehasht. Die Nachricht wird dann mit dem Ergebnis der vorhergehenden Operation zusammen verschickt.

Im letzten Szenario werden alle Ansätze

Hinweis

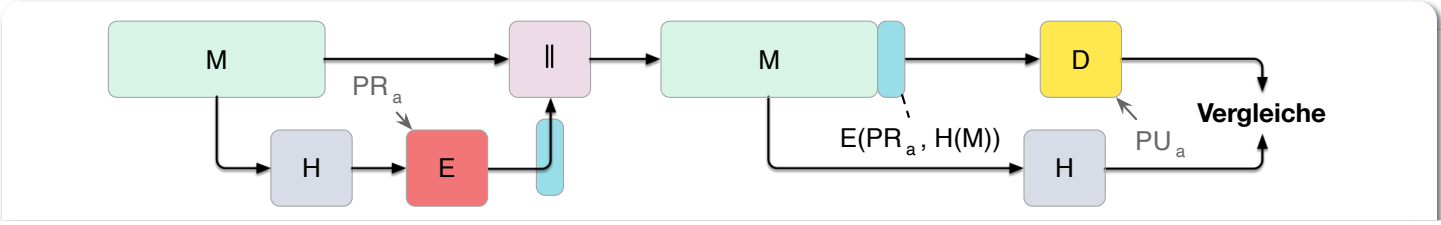
Bei *Man-in-the-Middle-Angriffen* handelt es sich um einen Fachbegriff und häufig wird zum Beispiel Eve oder Mallory verwendet, um die Person zu bezeichnen, die den Angriff durchführt. Gelegentlich wird auch *Adversary-in-the-Middle* oder *Person-in-the-Middle* verwendet.

Message-Digests

Im allgemeinen Sprachgebrauch wird auch von *Message Digests* gesprochen.

Digitale Signaturen - vereinfacht

Digitale Signaturen dienen dem Nachweis der Authentizität einer Nachricht und der Integrität der Nachricht. Jeder, der einen öffentlichen Schlüssel hat, kann die Signatur überprüfen, aber nur der Besitzer des privaten Schlüssels kann die Signatur erstellen.



Anforderungen an die Resistenz von Hashfunktionen

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + Digitale Signaturen	✓	✓	✓
Einbruchserkennung und Viruserkennung		✓	
Hash + Symmetrische Verschlüsselung			
Passwortspeicherung	✓		
MAC	✓	✓	✓

Einbruchserkennung und Viruserkennung - Hintergrund

Bei der Einbruchserkennung und Viruserkennung ist *second preimage* Resistenz erforderlich. Andernfalls könnte ein Angreifer seine Malware so schreiben, ass diese einen Hash wie eine vorhandene gutartige Software hat und so verhindern, dass die Malware auf eine schwarze Liste gesetzt werde kann, ohne den Kollateralschaden, dass auch die gutartige Software fälschlicherweise als Malware erkannt wird.

Aufwand eines Kollisionsangriffs

Ein Kollisionsangriff erfordert weniger Aufwand als ein *preimage* oder ein *second preimage* Angriff.

Dies wird durch das Geburtstagsparadoxon erklärt. Wählt man Zufallsvariablen aus einer Gleichverteilung im Bereich von 0 bis $N-1$, so übersteigt die Wahrscheinlichkeit, dass ein sich wiederholendes Element gefunden wird, nach $\sqrt{N}$ Auswahlen $0,5$. Wenn wir also für einen m-Bit-Hashwert Datenblöcke zufällig auswählen, können wir erwarten, zwei Datenblöcke innerhalb von $\sqrt{2^m} = 2^{m/2}$ Versuchen zu finden.

Beispiel

Es ist relativ einfach, ähnliche Meldungen zu erstellen. Wenn ein Text 8 Stellen hat, an denen ein Wort mit einem anderen ausgetauscht werden kann, dann hat man bereits 2^8 verschiedene Texte.

Es ist relativ trivial(1), vergleichbare(2) Nachrichten(3) zu schreiben(4). Wenn ein Text 8 Stellen hat, an denen ein Ausdruck(5) mit einem vergleichbaren (6) ausgetauscht werden kann, dann erhält(7) man bereits 2^8 verschiedene Dokumente(8).

Effizienzanforderungen an kryptografische Hashfunktionen

Effizienz bei der Verwendung für Signaturen und zur Authentifizierung:

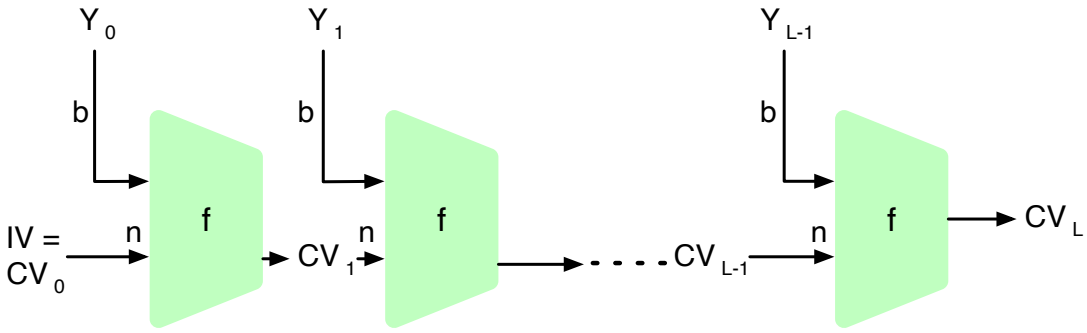
Bei der Verwendung zur Nachrichtenauthentifizierung und für digitale Signaturen ist $H(N)$ für jedes beliebige N relativ einfach zu berechnen. Dies soll sowohl Hardware- als auch Softwareimplementierungen ermöglichen.

vs.

Brute-Force-Angriffe auf Passwörter erschweren:

Bei der Verwendung für das Hashing von Passwörtern soll es schwierig sein den Hash effizient zu berechnen, selbst auf spezialisierter Hardware (GPUs, ASICs).

Struktur eines sicheren Hash-Codes



IV = Initialer Wert (Algorithmus-abhängig)

CV_i = Verkettungsvariable

Y_i = ier Eingabeblock

f = Kompressions-funktion

n = Länge des Blocks

L = Anzahl der Eingabeblöcke

b = Länge des Eingabeblocks

Übung

XOR als Hashfunktion

Warum ist eine einfache „Hash-Funktion”, die einen 256-Bit-Hash-Wert berechnet, indem sie ein XOR über alle Blöcke einer Nachricht durchführt, im Allgemeinen ungeeignet?

XOR als Hashfunktion

Warum ist eine einfache „Hash-Funktion”, die einen 256-Bit-Hash-Wert berechnet, indem sie ein XOR über alle Blöcke einer Nachricht durchführt, im Allgemeinen ungeeignet?

Übung

Bewertung der Sicherheit

- Eine Nachricht M bestehe aus N 64-bit Blöcken: $X_1, \ldots, X_n$.
- Der Hashcode H(M) ist ein simpler XOR über alle Blöcke:
  $H(M) = h = X_1 \oplus X_2 \oplus \ldots \oplus X_n$.
- h wird als der X_{N+1} Block an die Nachricht angehängt und danach wird unter Verwendung des CBC Modus die Nachricht inkl. des Hashcodes verschlüsselt ($C = Y_1, \ldots, Y_{N+1}$).
- Gegen welche Art von Manipulation ist diese Konstruktion *nicht* sicher?

  Studieren Sie ggf. noch einmal den CBC Modus.

Bewertung der Sicherheit

- Eine Nachricht M bestehe aus N 64-bit Blöcken: $X_1, \ldots, X_n$.
- Der Hashcode H(M) ist ein simpler XOR über alle Blöcke: $H(M) = h = X_1 \oplus X_2 \oplus \ldots \oplus X_n$.
- h wird als der X_{N+1} Block an die Nachricht angehängt und danach wird unter Verwendung des CBC Modus die Nachricht inkl. des Hashcodes verschlüsselt ($C = Y_1, \ldots, Y_{N+1}$).
- Gegen welche Art von Manipulation ist diese Konstruktion *nicht* sicher?

  Studieren Sie ggf. noch einmal den CBC Modus.

Irrelevanz von Second-Preimage-Resistenz und Kollisionssicherheit

Warum sind *Second-Preimage-Resistenz* und Kollisionssicherheit von nachgeordneter Relevanz, wenn der Hash-Algorithmus zum Hashing von Passwörtern verwendet wird?

Irrelevanz von Second-Preimage-Resistenz und Kollisionssicherheit

Warum sind *Second-Preimage-Resistenz* und Kollisionssicherheit von nachgeordneter Relevanz, wenn der Hash-Algorithmus zum Hashing von Passwörtern verwendet wird?

2. *Message Authentication Codes* (MACs)

Prof. Dr. Michael Eichberg

Hinweis

Message Authentication Codes könnte ins Deutsche mit Nachrichtenauthentifizierungscodes übersetzt werden, dies ist aber nicht üblich.

Im allgemeinen Sprachgebrauch wird von *MAC*s gesprochen.

HMAC (Hash-based Message Authentication Code)

Auch als *keyed-hash message authentication code* bezeichnet.

$$
\begin{aligned}
HMAC(K, m) &= H((K' \oplus opad) || H((K' \oplus ipad) || m)) \\
K' &= \begin{cases} H(K) & \text{falls K größer als die Blockgröße ist} \\ K & \text{andernfalls} \end{cases}
\end{aligned}
$$

H is eine kryptografische Hashfunktion.

m ist die Nachricht.

K ist der geheime Schlüssel (*Secret Key*).

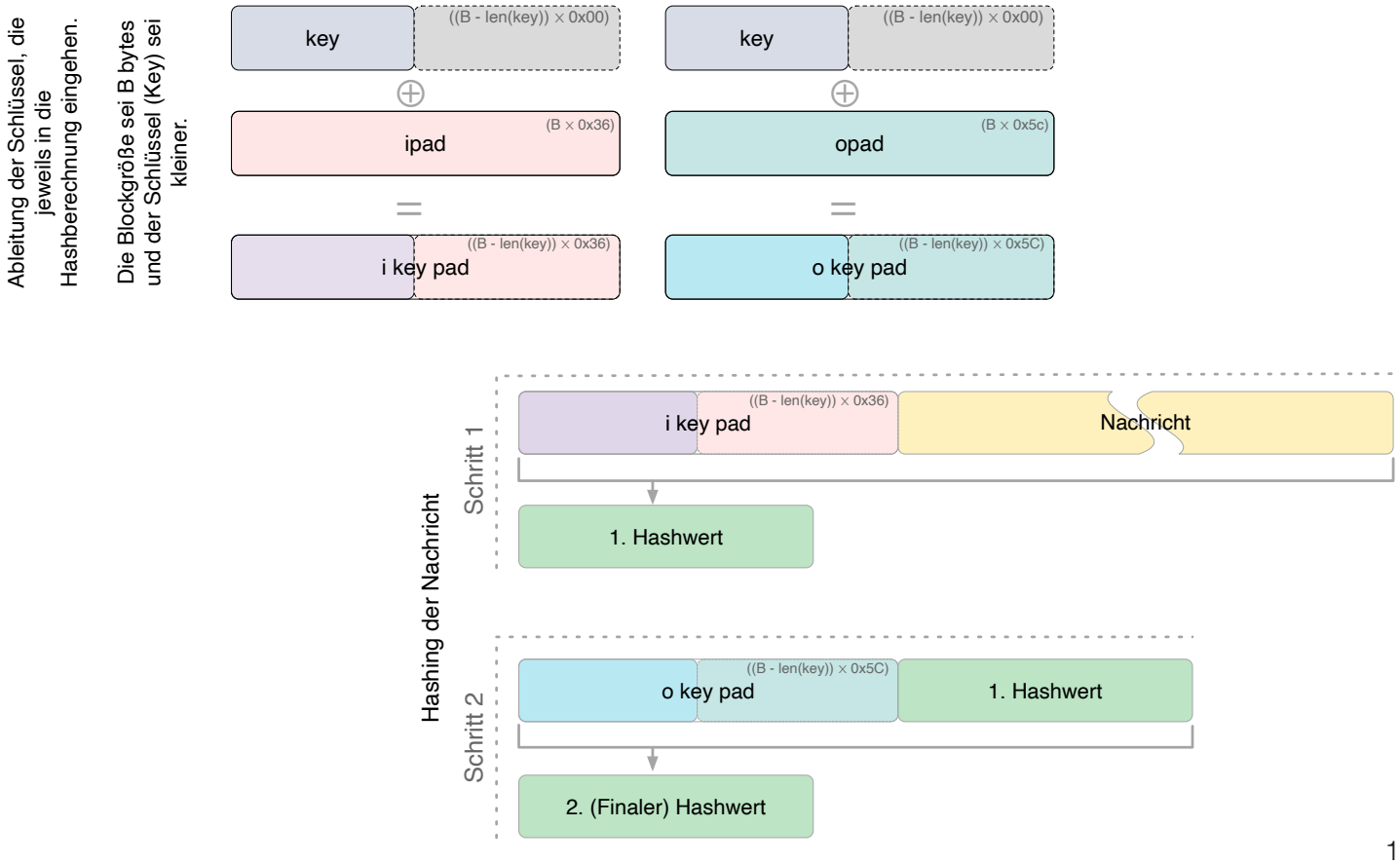
K' ist vom Schlüssel K abgeleiteter Schlüssel mit Blockgröße (ggf. *padded* oder *gehasht*).

|| ist die Konkatenation.

$\oplus$ ist die XOR Operation.

$opad$ ist das äußere Padding bestehend aus Wiederholungen von 0x5c in Blockgröße.

$ipad$ ist das innere Padding bestehend aus Wiederholungen von 0x36 in Blockgröße.



Padding und Hashing

Im Rahmen der Speicherung von Passwörtern und *Secret Keys* ist die Verwendung von Padding Operationen bzw. das Hashing von Passwörtern, um Eingaben in einer wohl-definierten Länge zu bekommen, üblich. Neben dem hier gesehenen Padding, bei dem 0x00 Werte angefügt werden, ist zum Beispiel auch das einfache Wiederholen des ursprünglichen Wertes, bis man auf die notwendige Länge kommt, ein Ansatz.

Diese Art Padding darf jedoch nicht verwechselt werden mit dem Padding, dass ggf. im Rahmen der Verschlüsselung von Nachrichten notwendig ist, um diese ggf. auf eine bestimmte Blockgröße zu bringen (zum Beispiel bei ECB bzw. CBC Block Mode Operations.)

HMAC Berechnung in Python

Implementierung

```
import hashlib
pwd = b"MyPassword"
stretched_pwd = pwd + (64-len(pwd)) * b"\x00"
ikeypad = bytes(map(lambda x : x ^ 0x36 , stretched_pwd)) # xor with ipad
okeypad = bytes(map(lambda x : x ^ 0x5c , stretched_pwd)) # xor with opad
hash1 = hashlib.sha256(ikeypad+b"JustASalt"+b"\x00\x00\x00\x01").digest()
hmac  = hashlib.sha256(okeypad+hash1).digest()
```

Ausführung

```
hmac =
b'h\x88\xc2\xb6X\xb7\xcb\x9c\x90\xc2R...
  \x16\x87\x87\x0e\xad\xa1\xe1:9\xca'
```


HMAC ist auch direkt als Bibliotheksfunktion verfügbar.

```
import hashlib
import hmac

hash_hmac = hmac.new(
    b"MyPassword",
    b"JustASalt"+b"\x00\x00\x00\x01",
    hashlib.sha256).digest()

hash_hmac =
    b'h\x88\xc2\xb6X\xb7\xcb\x9c\x90\xc2R...
      \x16\x87\x87\x0e\xad\xa1\xe1:9\xca'
```

GCM - Glaois Counter Mode

TODO