

Passwörter sicher speichern

Inhaltsverzeichnis

- ✦ Warum der Vortrag? eHarmony, LinkedIn, last.fm, Sony, ...
- ✦ Geschichte vom Speichern der Passwörter
 - ✦ Klartext-Passwörter
 - ✦ Passwörter, gehasht
 - ✦ gebrochene Hash-Algorithmen
 - ✦ Rainbowtables
 - ✦ Kollisionen
 - ✦ KDF: key derivation function
- ✦ Empfehlungen für Technik und Passwörter

Der Erzähler

- ✦ Fabian Blechschmidt
- ✦ PHP seit 2004
- ✦ Freelancer seit 2008
- ✦ Magento seit 2011
- ✦ Certified Magento Developer
- ✦ spielt gerne, aktuell mit
 - ✦ Magento und Symfony2
 - ✦ Passwörter, Hashing, etc.



Warum der Vortrag?

eHarmony, LinkedIn, last.fm, Sony, ...

- LinkedIn, eHarmony (Dating-Portal) und last.fm (Streaming-Seite) gehackt
 - Beute: 8 Millionen Accounts
- *„I left a bunch of stuff running over night, and have about 50% of all the passwords cracked.“* (LinkedIn: 6,46 Mio Acc., MD5, ungesalzen)

▪ <http://erratassec.blogspot.de/2012/06/linkedin-vs-password-cracking.html>

Ausgangspunkt:

Wir werden gehackt - früher oder später

- ✦ Frage ist **NICHT**: Hat unsere Anwendung Sicherheitslücken?
- ✦ SONDERN: Wann werden sie gefunden?
- ✦ Und durch wen?

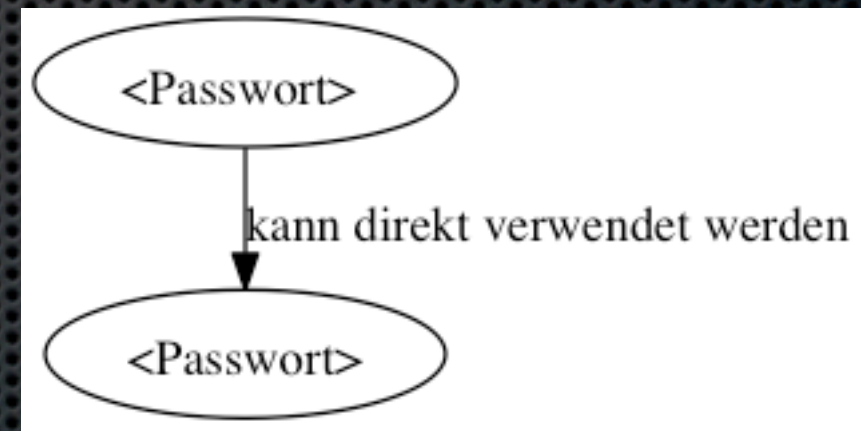
Klartextpasswörter

Klartextpasswörter

- ✦ Wie? Passwort direkt in die Datenbank
- ✦ Vorteil:
 - ✦ Benutzer kriegt SEIN Passwort wieder
- ✦ Nachteil:
 - ✦ Datenbank weg, ALLE Passwörter weg.

Klartextpasswörter - Probleme

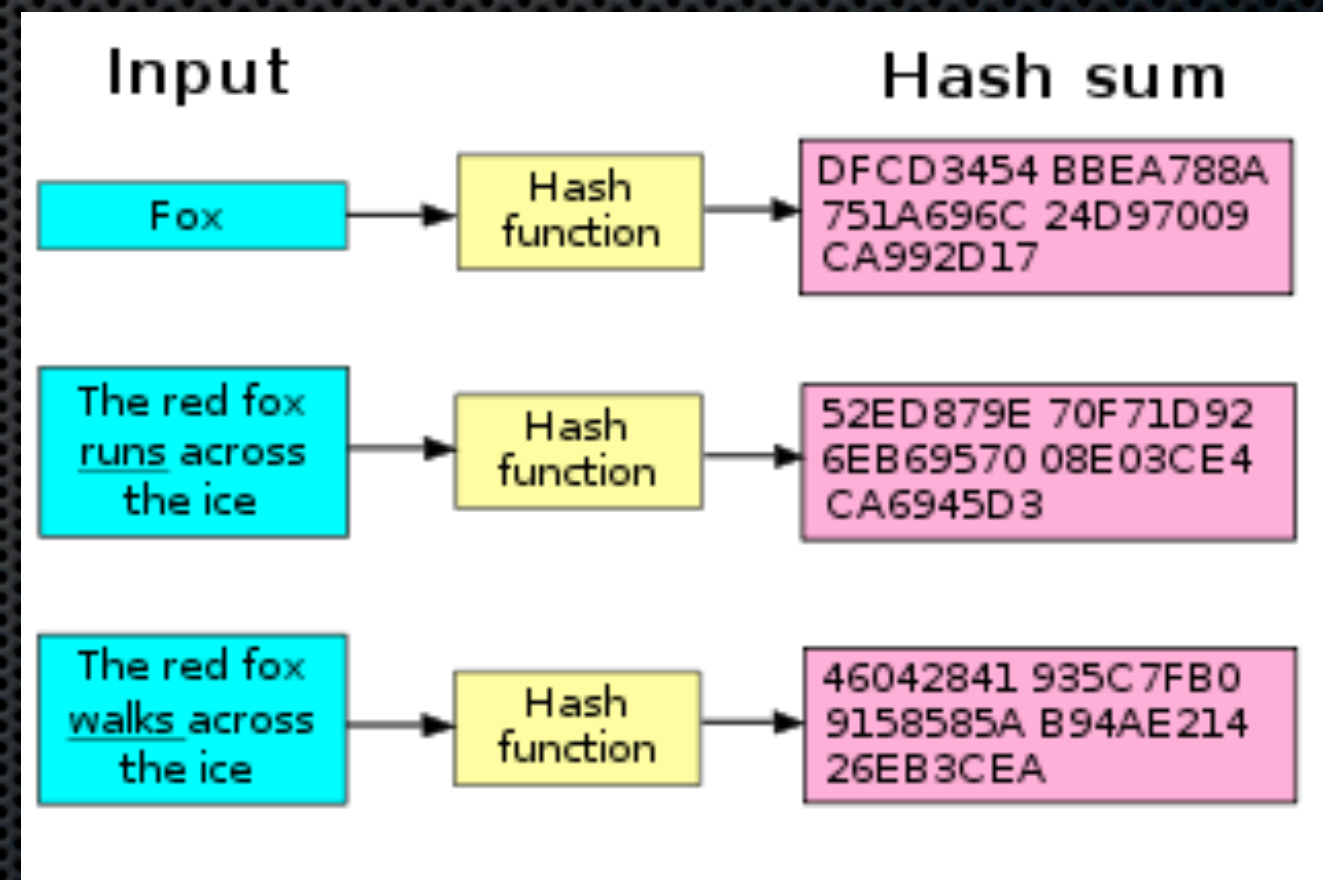
- ✦ Problem(e):
 - ✦ Zeit zum Cracken (Passwörter berechnen):
 - ✦ nicht nötig \Rightarrow so schnell wie die Platte lesen kann.



Passwörter, gehasht

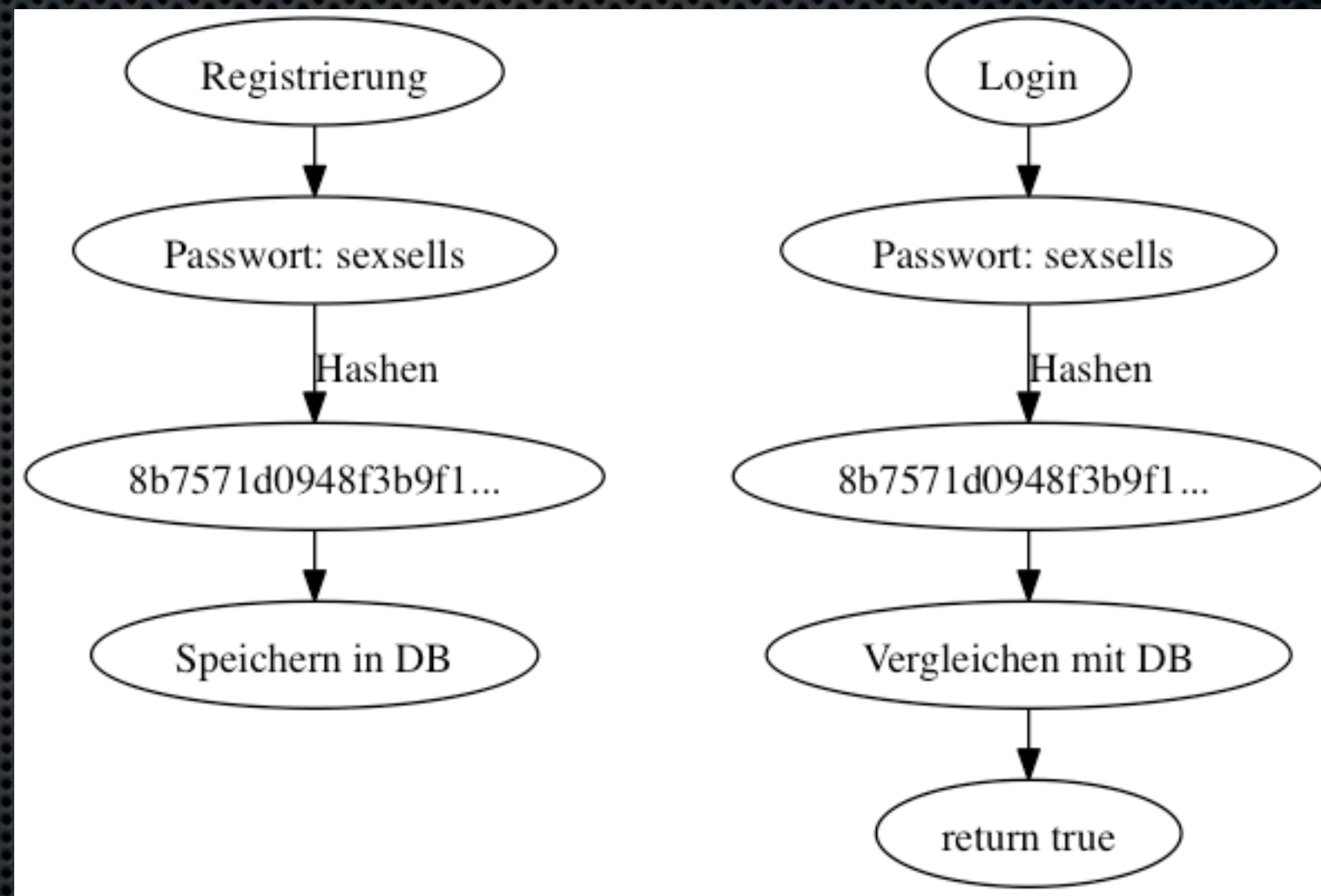
Hashing

- ✧ Einweg-Funktion
- ✧ leicht in eine Richtung,
schwer in die andere
 - ✧ z.B. Primfaktorzerlegung
vs. Multiplikation oder
Modulo-Berechnungen
- ✧ z.B. SHA2, SHA512,
Whirlpool



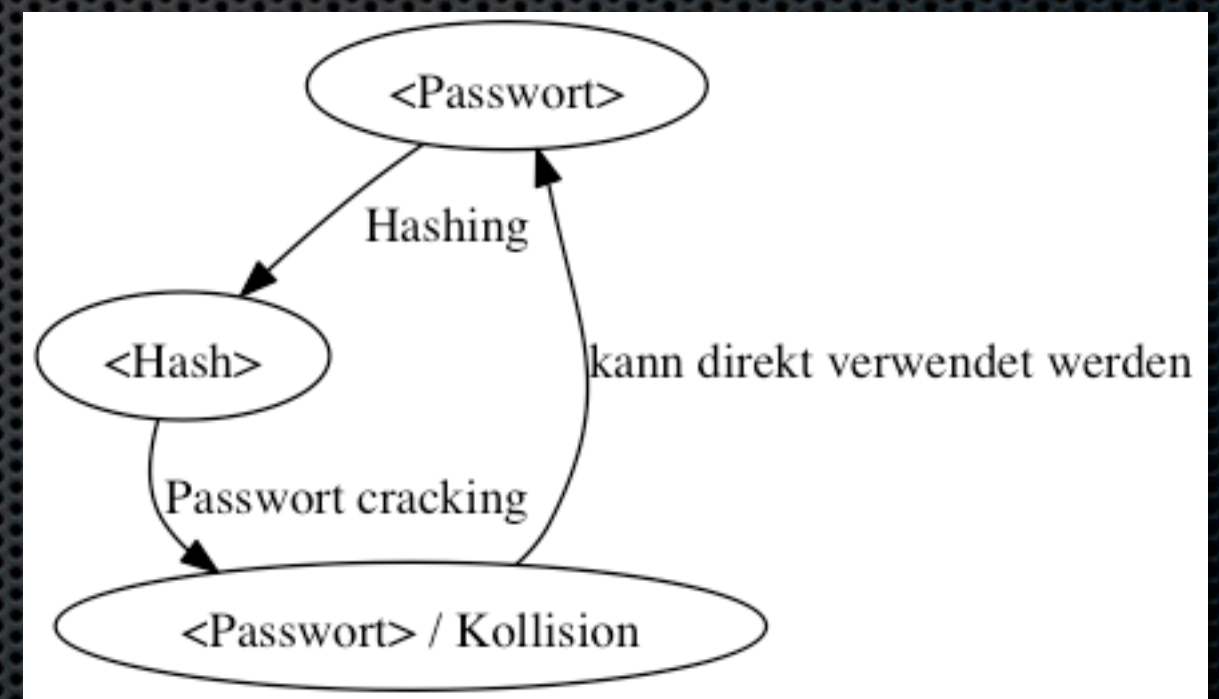
Passwort (gehasht)

- ✦ Passwörter
gehasht speichern
- ✦ Vorteil:
 - ✦ Passwort nicht
im Klartext
- ✦ Nachteil:
 - ✦ heute relativ
leicht zu knacken



Passwort (gehasht) - Probleme

- ✦ Klartext zum Hash finden ist schwer, aber:
 - (1) viele Hash-Algorithmen „broken“ (DES, MD5, SHA1, ...)
 - (2) man kann Passwörter/Kollisionen im Voraus berechnen (Rainbowtable)
 - (3) viel Rechenkraft vorhanden (GPU, Amazon, etc.)

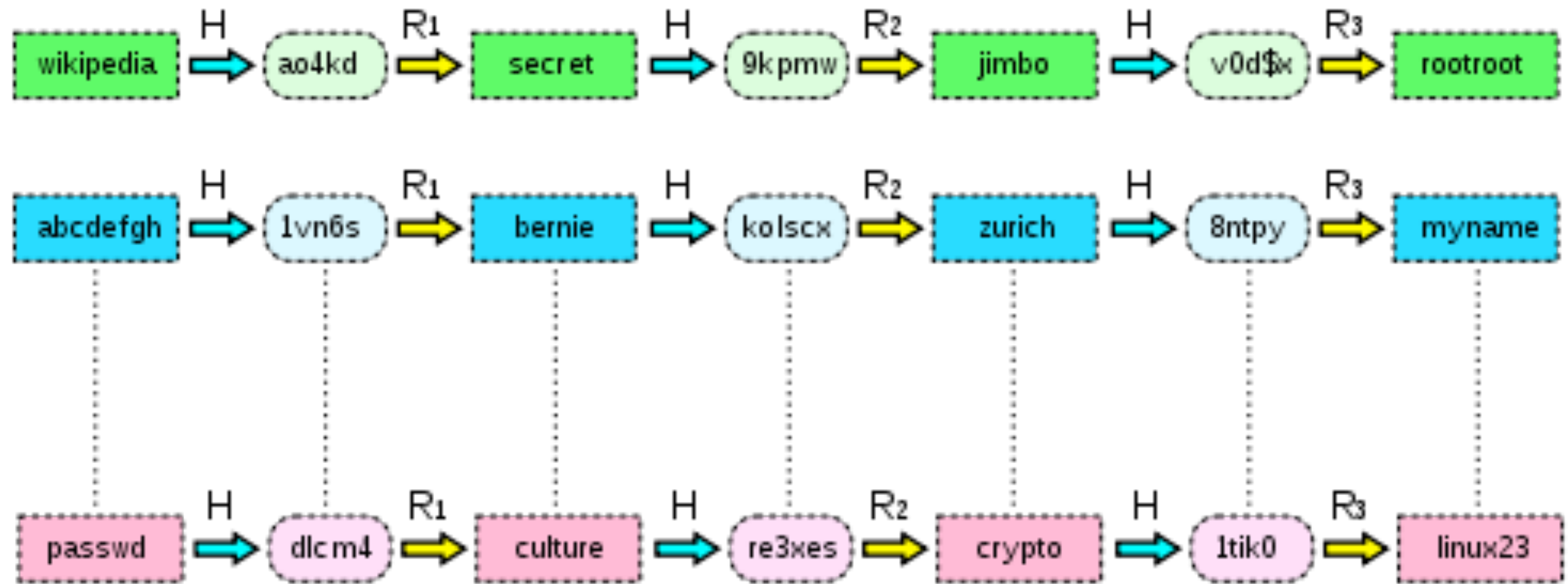


kaputte Hash-Algorithmen

broken algorithms

- ✦ Finger weg!
- ✦ DES (alte crypt() Impl.)
- ✦ MD2
- ✦ MD4
- ✦ MD5
- ✦ SHA1
- ✦ RC4 (WEP)

Rainbowtables

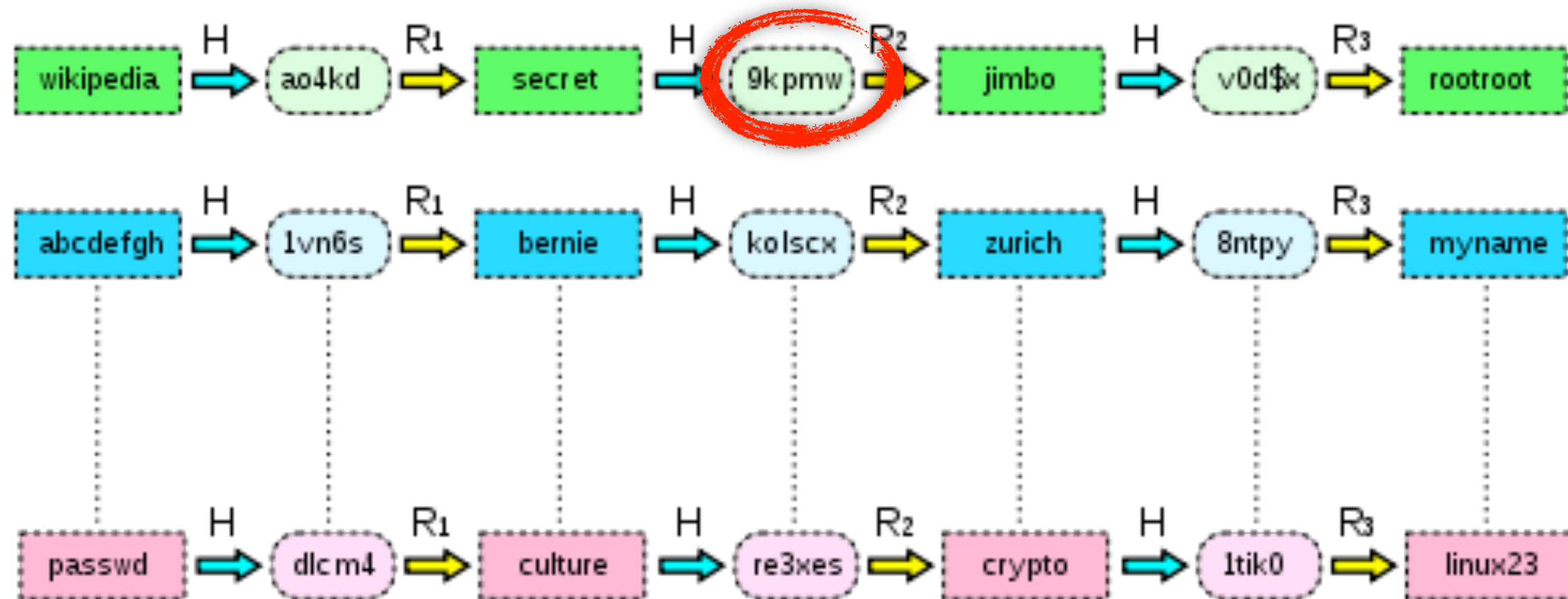


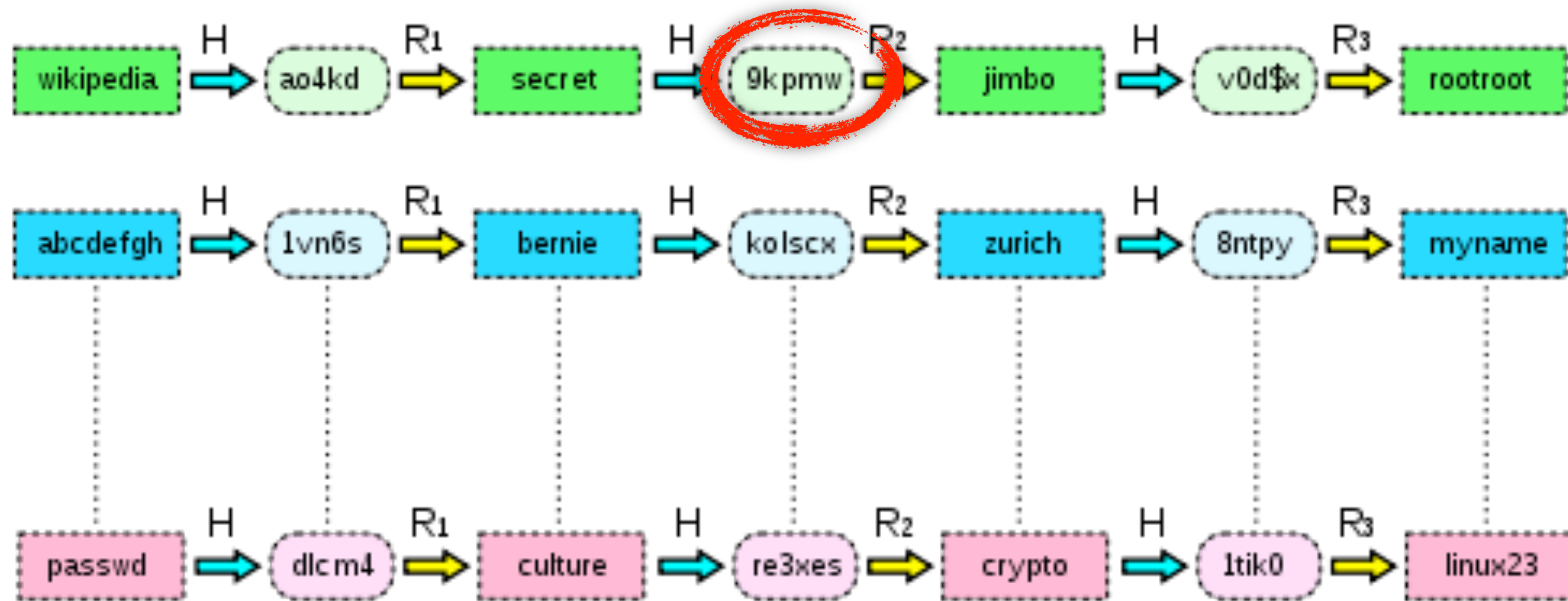
Rainbowtable

Startwert -> Hashing -> Hash -> Reduktion -> usw.

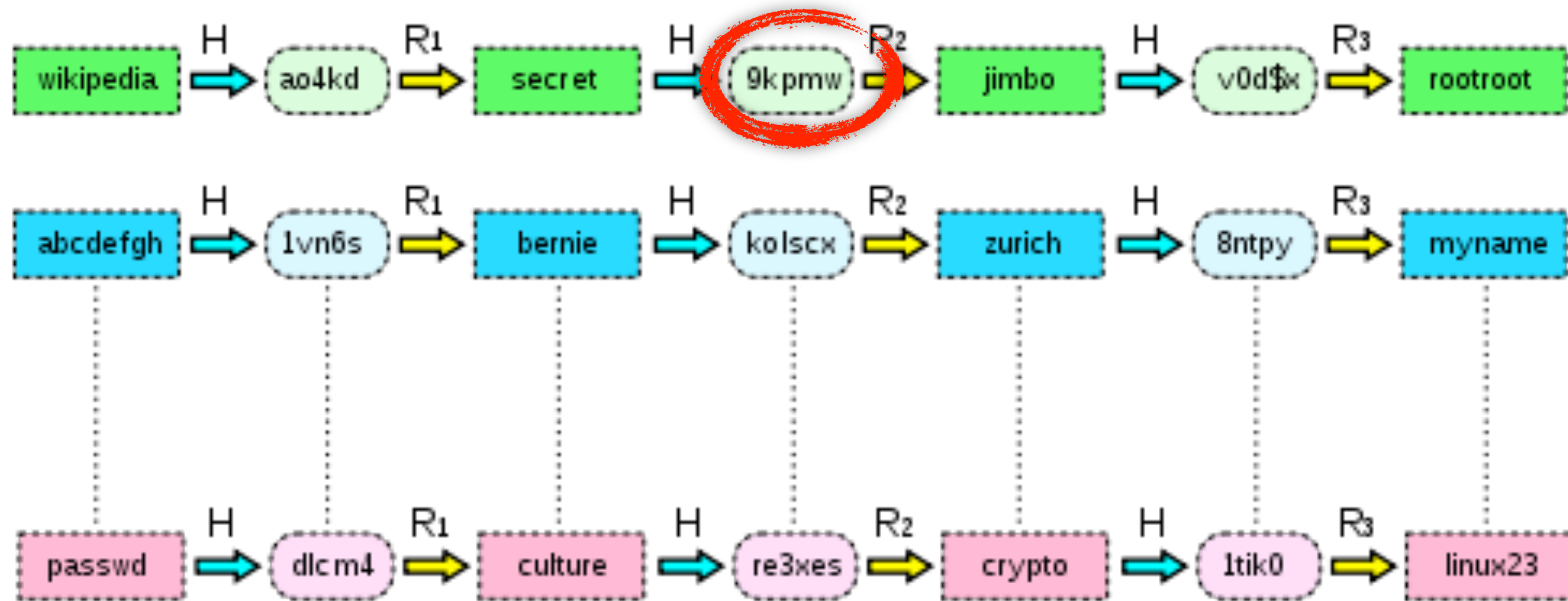
Rainbowtables - Nutzen

- Startwert, Reduktionsfunktion & Endwert werden gespeichert
- Hashes sollten sich nicht wiederholen!
- Benutzung:
 - gesucht Hash reduzieren, hashen, bis Endwert gefunden
 - Startwert nehmen, hashen, reduzieren, usw. bis Schlüssel gefunden



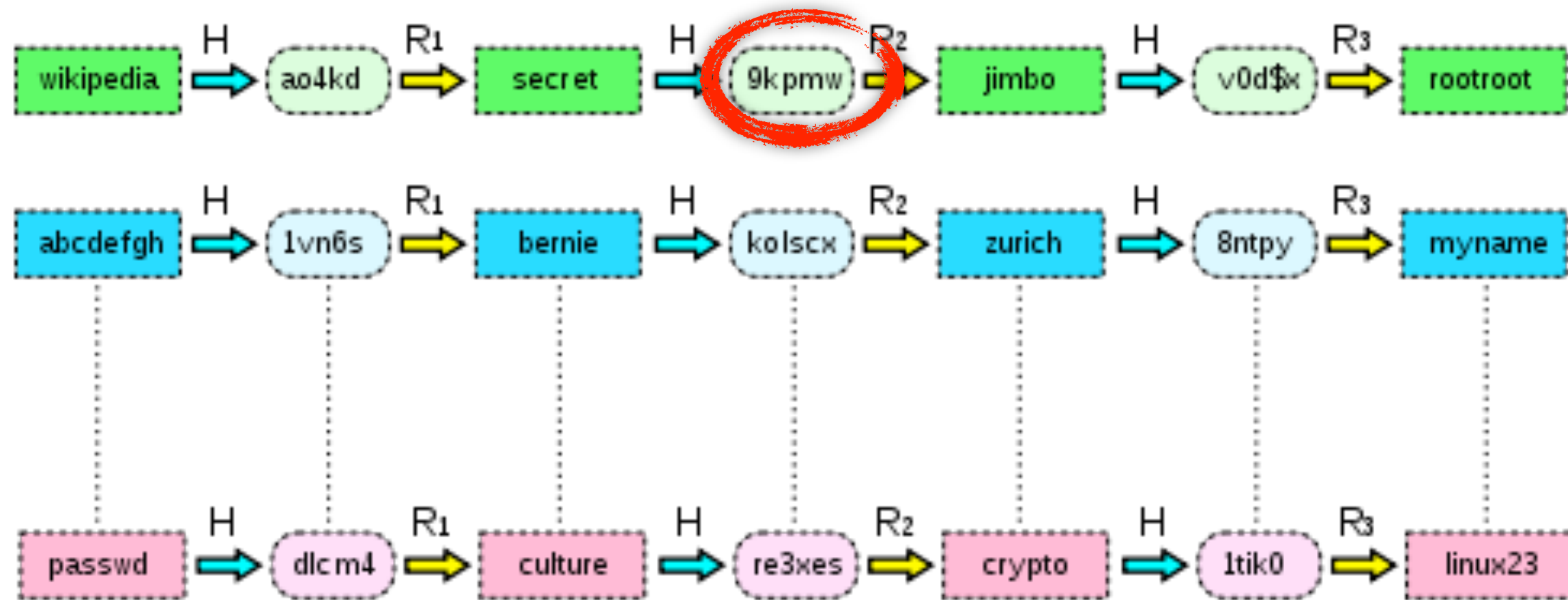


gesuchter Hash: 9kpmw



gesuchter Hash: 9kpmw

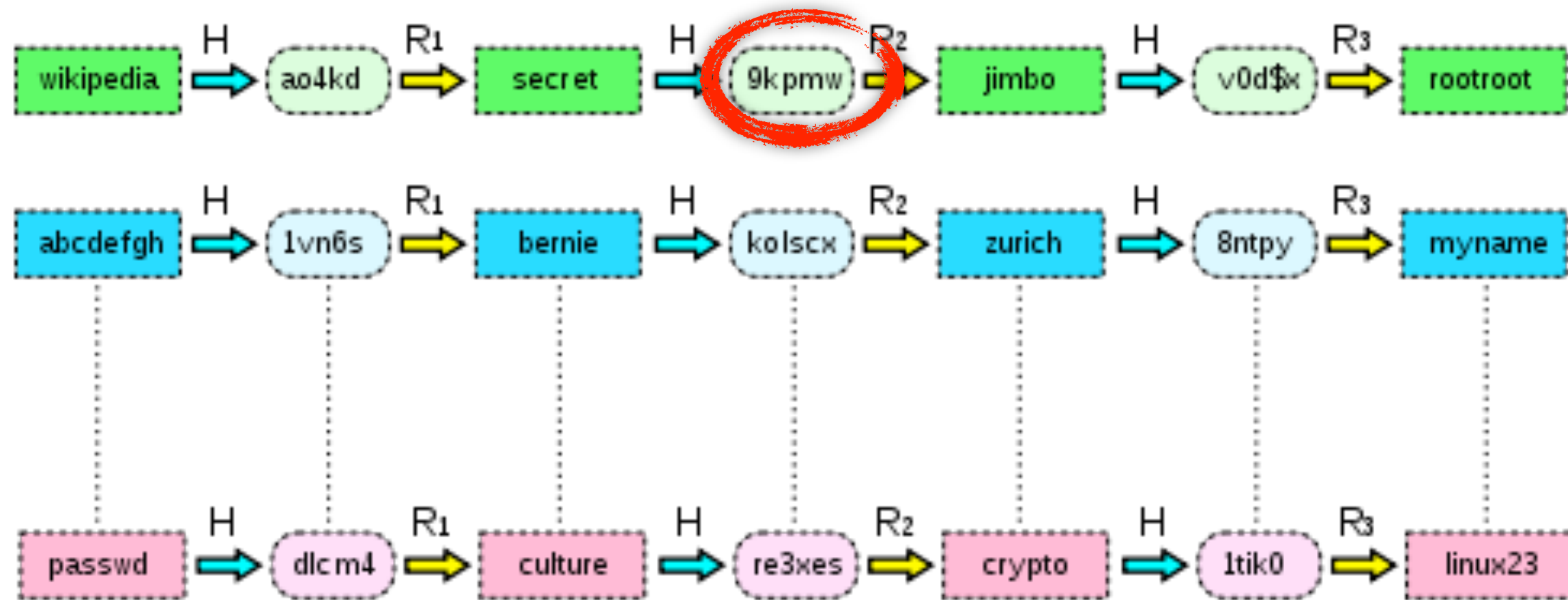
9kpmw → R2 → jimbo → H → v0d\$x → R3 → rootroot



gesuchter Hash: 9kpmw

9kpmw → R2 → jimbo → H → v0d\$x → R3 → rootroot

rootroot → Start → wikipedia

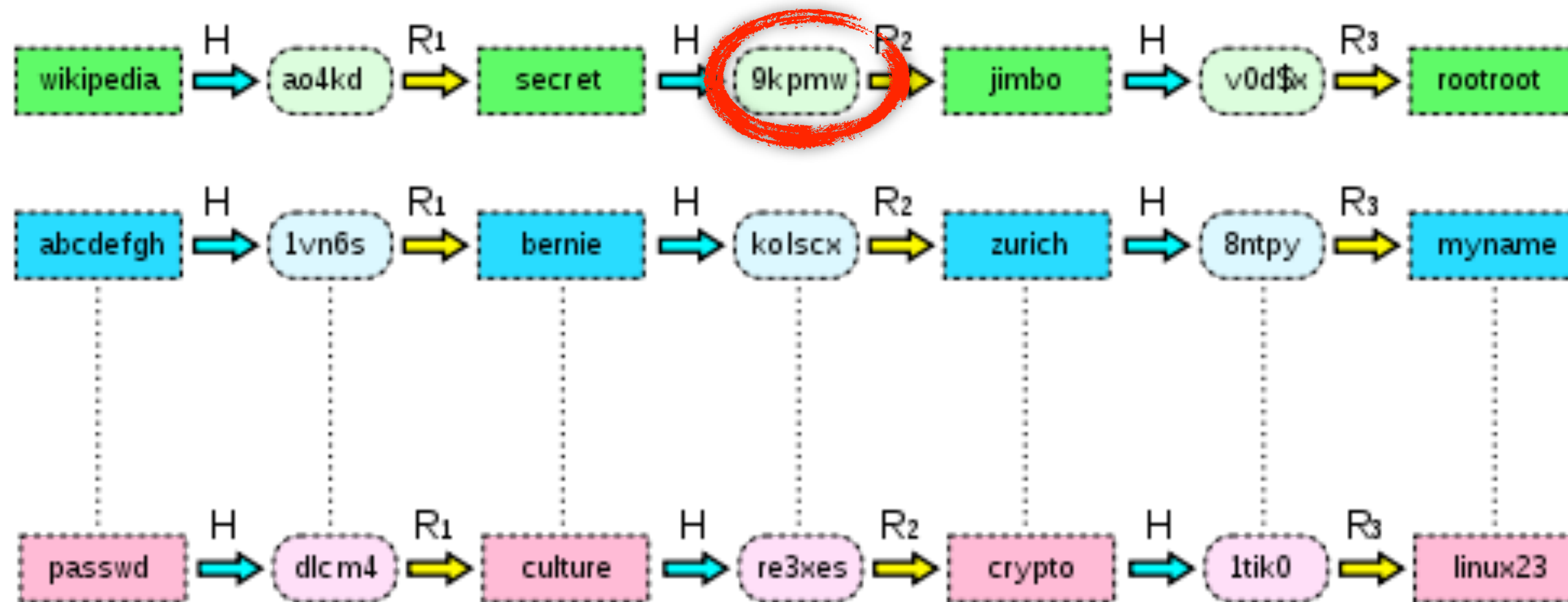


gesuchter Hash: 9kpmw

9kpmw → R2 → jimbo → H → v0d\$x → R3 → rootroot

rootroot → Start → wikipedia

wikipedia → H → ao4kd → R1 → secret → 9kpmw



gesuchter Hash: 9kpmw

9kpmw → R2 → jimbo → H → v0d\$x → R3 → rootroot

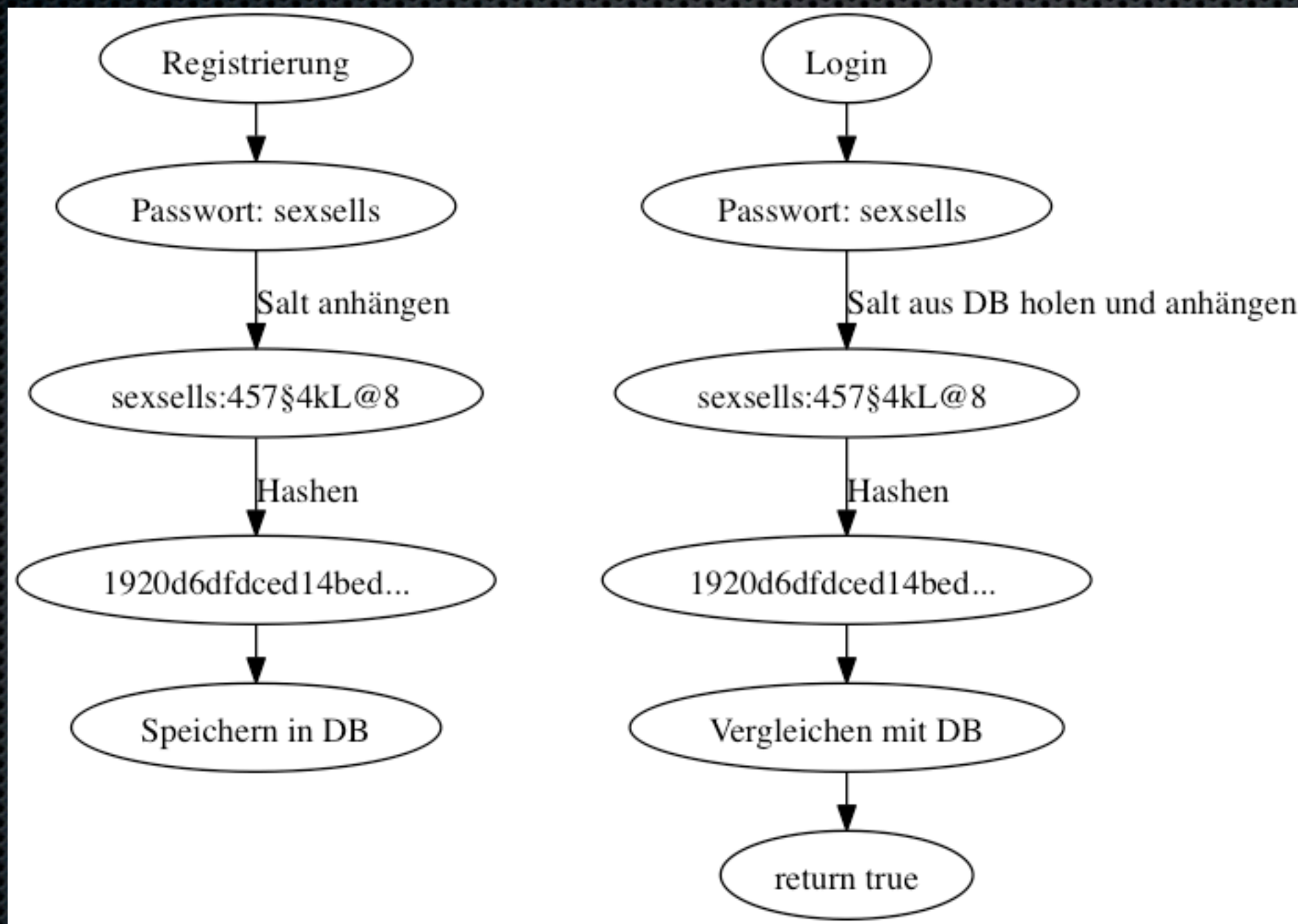
rootroot → Start → wikipedia

wikipedia → H → ao4kd → R1 → secret → 9kpmw

GEFUNDEN: **9kpmw**

Problem: Rainbowtables & Kollisionen

Lösung: Salting

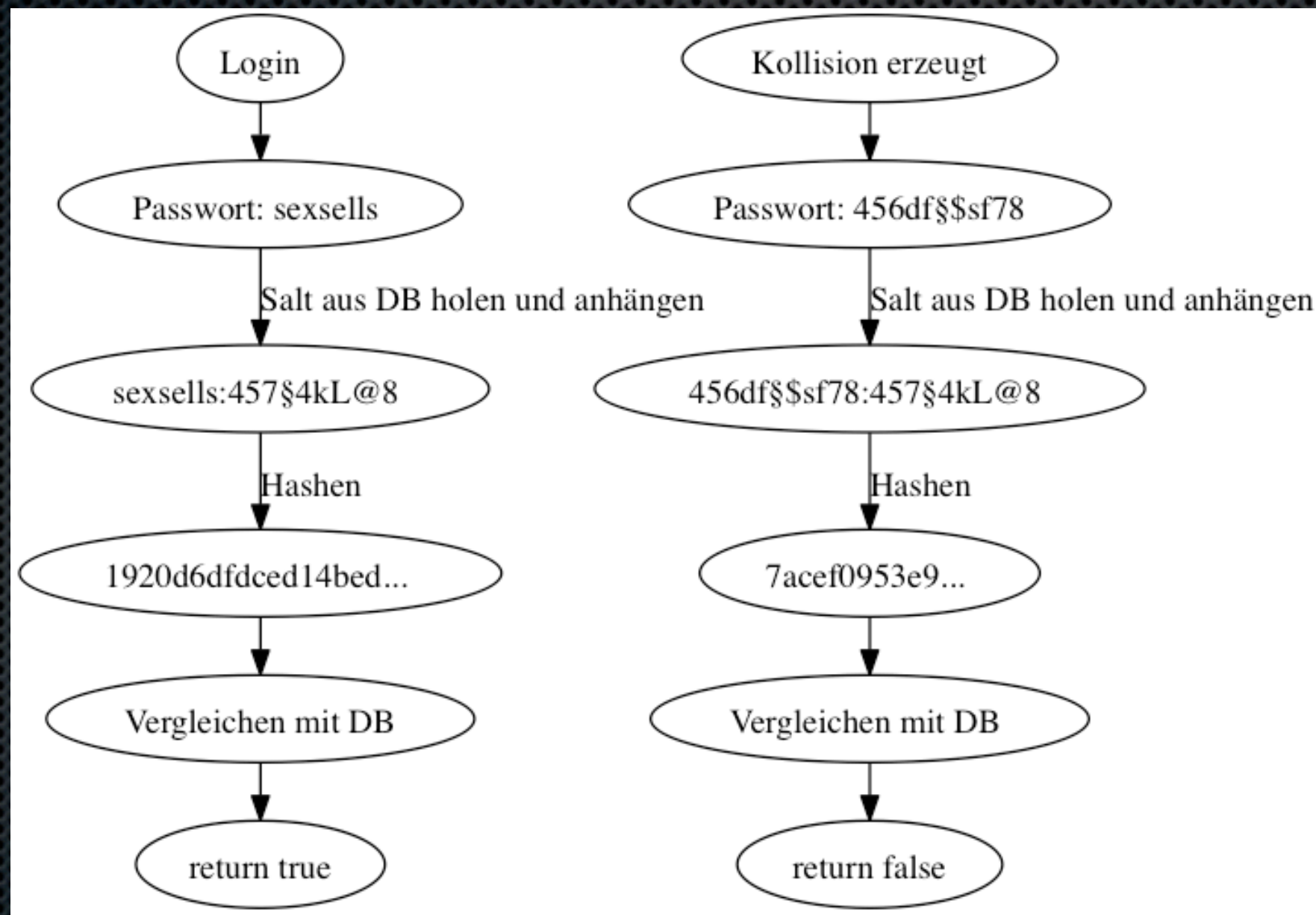


Hilft gegen Rainbowtable

- ✦ Rainbowtables existieren für ungesalzene Hashes
- ✦ gesalzener Hash benötigt eigene Rainbowtable
 - ✦ Präfix/Suffix ist festgelegt
- ✦ Tabelle wird größer (viel größer)
 - ✦ Ziel ist Kosten&Aufwand erhöhen, nicht verhindern!

Hilft gegen Kollision

" $H(\text{sexsells}) == H(456df\$\$sf78)$ "

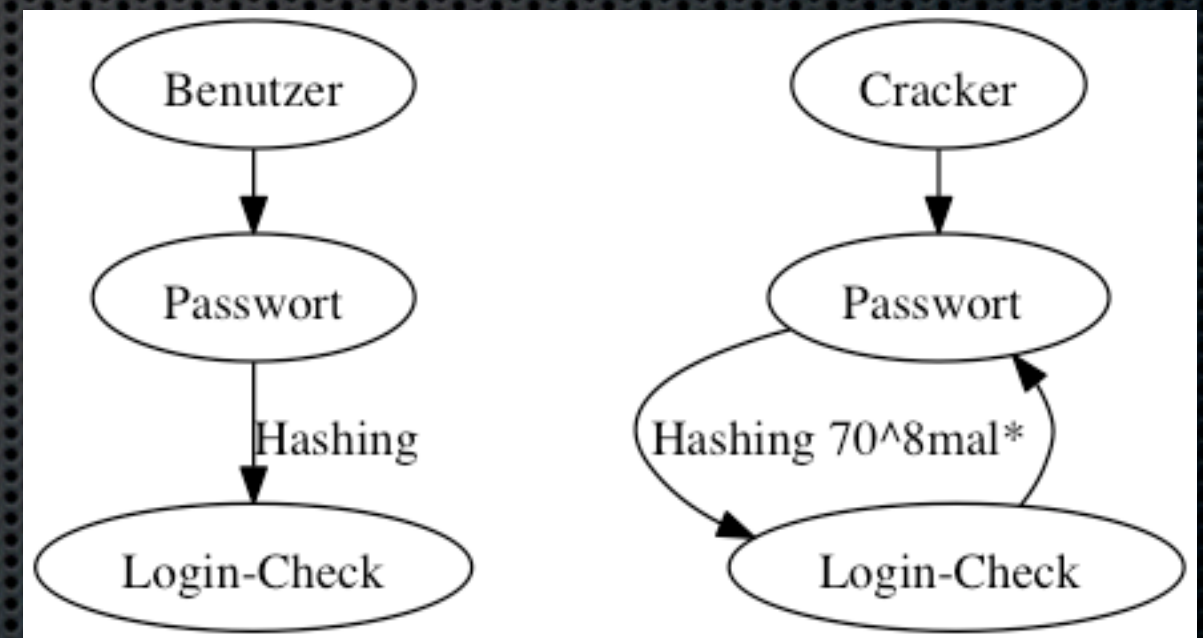


KDF: key derivation function

Hashes lassen sich „zu schnell“ berechnen

- ✦ pro Login einmal Hashen
- ✦ pro Passwort cracken:

576.480.100.000.000
 $\approx 5,8 \cdot 10^{13}$ mal hashen



* 70^8 [A-Za-z0-9#+-_.,;:]{8}

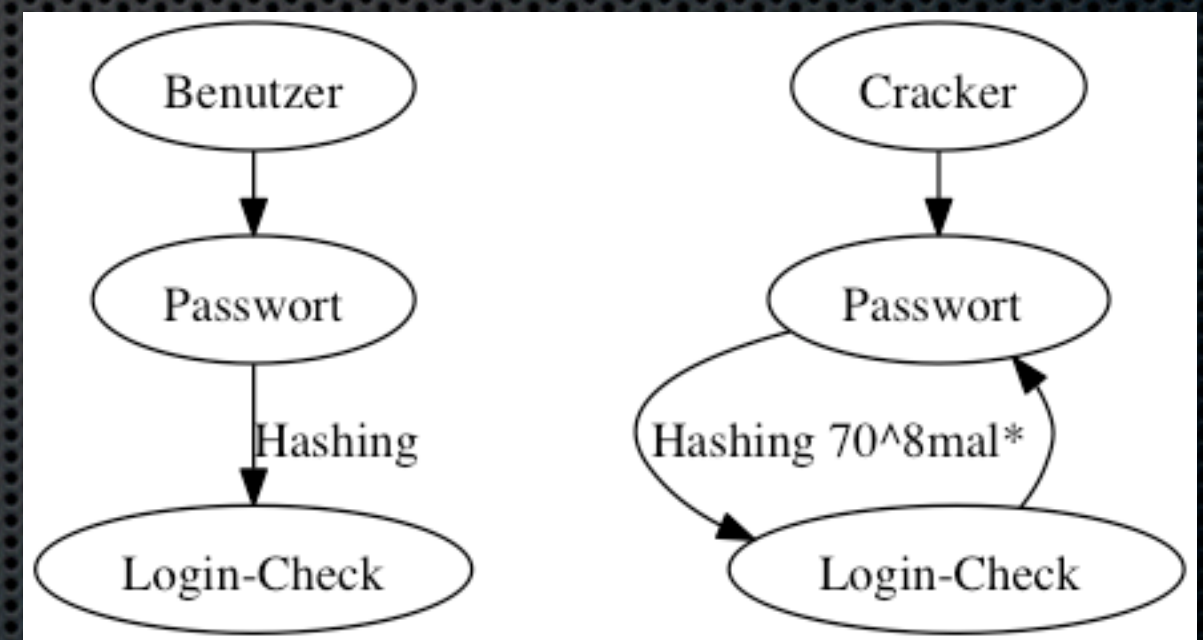
KDF - mehr Aufwand für alle

- ✦ viele hundert mal hashen:
- ✦ $n = 10^4$; $sres = \text{„Beliebige Zeichenkette“}$;
for($i = 0; i \leq n; i++$){
 $sres = \text{sha2}(sres)$;
}
return $sres$

Hashes lassen sich „zu schnell“ berechnen

- ✦ pro Login 10.000mal Hashen
- ✦ pro Passwort cracken:

5.764.801.000.000.000.000
 $\approx 5,8 \cdot 10^{17}$ mal hashen



* 70^8 [A-Za-z0-9#+-_.,;:]{8}

* 10^4 mal KDF


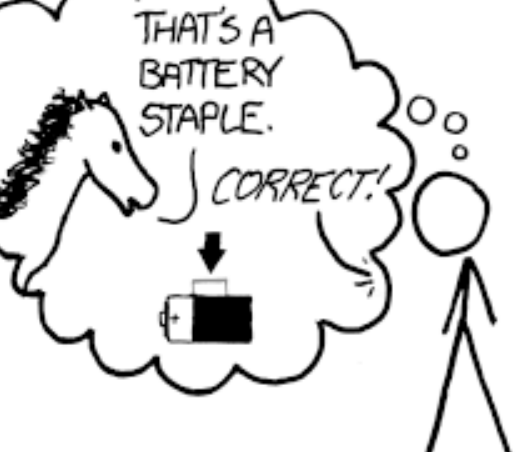
Key derivation function

- abstrakt
- Beispiel (WPA2 (PBKDF2))
- **Hashfunktion**
 - **sehr oft** auf ein **Eingabestring** und ein **Salt** angewendet.
- HMAC-SHA1
- 4096
- WLAN-Schlüssel
- SSID

technische Empfehlung

- ✦ key derivation function, z.B.
 - ✦ PBKDF2-HMAC-SHA256, $c = 86000$
 - ✦ bcrypt, $\text{cost} = 11$
 - ✦ scrypt, $N = 2^{14}, r = 8, p = 1$
- ✦ langer Salt, z.B.
 - ✦ `md5(username)`
 - ✦ `md5(time())`

Passwörter: Sätze statt Sonderzeichen

<p>□□□□□□□□□□□□□□□□</p> <p>UNCOMMON (NON-GIBBERISH) BASE WORD</p> <p>ORDER UNKNOWN</p> <p>Tr0ub4dor &3</p> <p>CAPS? □</p> <p>COMMON SUBSTITUTIONS □□□</p> <p>NUMERAL □□□</p> <p>PUNCTUATION □□□□</p> <p>(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)</p>	<p>~28 BITS OF ENTROPY</p> <p>□□□□□□□□ □□□□□□□□ □□□ □□□□</p> <p>$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$</p> <p>(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)</p> <p>DIFFICULTY TO GUESS: EASY</p>	<p>WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?</p> <p>AND THERE WAS SOME SYMBOL...</p>  <p>DIFFICULTY TO REMEMBER: HARD</p>
<p>correct horse battery staple</p> <p>□□□□□□ □□□□□□ □□□□□□ □□□□□□ □□□□□□ □□□□□□ □□□□□□ □□□□□□</p> <p>FOUR RANDOM COMMON WORDS</p>	<p>~44 BITS OF ENTROPY</p> <p>□□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□ □□□□□□□□□□</p> <p>$2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$</p> <p>DIFFICULTY TO GUESS: HARD</p>	<p>THAT'S A BATTERY STAPLE.</p> <p>CORRECT!</p>  <p>DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT</p>

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED
EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS
TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Quellen

- <http://mashable.com/2012/06/15/hard-to-hack-password/>
- <http://erratasec.blogspot.de/2012/06/linkedin-vs-password-cracking.html>
- <http://blog.zoller.lu/2012/06/storing-password-securely-hashses-salts.html>
- http://de.wikipedia.org/wiki/Rainbow_Table
- <http://en.wikipedia.org/wiki/PBKDF2>
- <http://www.openwall.com/presentations/PHDays2012-Password-Security/>
- <http://en.wikipedia.org/wiki/Bcrypt>
- <http://www.mscs.dal.ca/~selinger/md5collision/>

Bilder

- Hashfunction: http://de.wikipedia.org/w/index.php?title=Datei:Hash_function_long.svg&filetimestamp=20051202013811
- Comic: <http://xkcd.com/936/>