Themen zur Computersicherheit

Teilnehmerauthentisierung

PD Dr. Reinhard Bündgen
buendgen@de.ibm.com

Authentisierung vs. Authentifizierung

- Authentisierung
  - „aktiv“
  - Subjekt authentisiert sich: beweist seine Identität
- Authentifizierung
  - „passiv“
  - Subjekt authentifiziert Objekt: erlangt Beweis über Identität des Objekts

Ziel der Authentisierung

- Ein Teilnehmer, der Zugang zu einem System anfordert muss seine Identität nachweisen
- Teilnehmer wird im System durch Teilnehmer-Identifikator (Mitgliedsnummer, Kontonummer, e-mail Adresse o.ä.) beschrieben
  - evtl. mit Attributen (wie Adresse, Geburtsdatum, Kreditkartennummer,…) versehen
- Die Verifikation, dass die Teilnehmerattribute wirklich zum Teilnehmer gehören
  - außerhalb des Systems, z.B. persönliche Anmeldung, Postident-Verfahren
  - Plausibilitätschecks: Rückfrage an e-mail Adresse, Konsistenz von Kreditkartendaten
  - elektronische Signaturen (ePA)

Richtung der Authentisierung

- Teilnehmer beweist dem System seine Identität
  - z.B login
- System beweist dem Teilnehmer seine Identität
  - z.B. die meisten HTTPS Server
- einseitige Authentisierung
  - z.B klassisches login
- beidseitige Authentisierung
  - z.B. ssh

Methoden der Authentisierung

- Besitz
  - z.B. Ausweis, Smartcard, Handy
  - → chipTAN, mTAN
- Wissen
  - Passwort, PIN, TAN-Liste
- (körperliche) Merkmale
  - Irismuster, Fingerabdruck, Retina, Handschrift, Stimme, Handgeometrie, Gesichtsgeometrie, DNA?,
- Lokation
  - GPS Daten, Konsole, Netzwerksegment
- Einfaktor-Authentisierung
- Mehrfaktor-Authentisierung (MFA)
- Vier-Augenprinzip

Passwortverfahren - Anforderungen

- Passwort: auch Kennwort, Passphrase, PIN
- Anforderungen
  - nur Teilnehmer darf Passwort kennen
  - System muss Passwort erkennen können
  - System muss unterschiedliche Passwörter unterscheiden können
- abgeleitete Anforderungen
  - Passwort darf nicht im Klartext gespeichert werden
  - Passwort darf nur verschlüsselt übertragen werden

Kryptographische Hashfunktionen

im folgenden gilt $\sum = \{0,1\}$

Definition Eine Funktion $h: \sum^* \rightarrow \sum^n$ heißt *Hashfunktion* wenn es einen effizienten Algorithmus gibt, der für jedes $m \in \sum^*$ den Wert h(m) berechnet.

Definition Eine Hashfunktion h heißt *Einweg-Hashfunktion*, wenn zu zufällig gewähltem $y \in \sum^n$ der Wert $h^{-1}(y)$ nicht effizient berechnet werden kann.

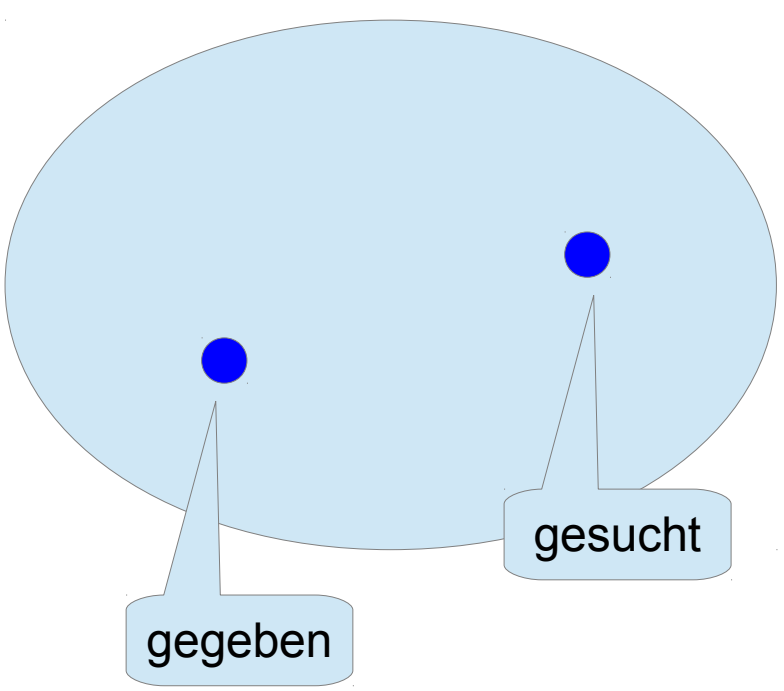
Definition Eine Hashfunktion h heißt *schwach kollisionsresistent (target collision resistent, 2nd pre-image resistent)* wenn es keinen effizienten Algorithmus gibt, der zu gegebenen $m \in \sum^*$ ein $m' \in \sum^*$ findet mit $m \neq m'$ und $h(m) = h(m')$.

Definition Eine Hashfunktion h heißt *stark kollisionsresistent* wenn es keinen effizienten Algorithmus gibt der $m \in \sum^*$ und $m' \in \sum^*$ findet mit $m \neq m'$ und $h(m) = h(m')$.

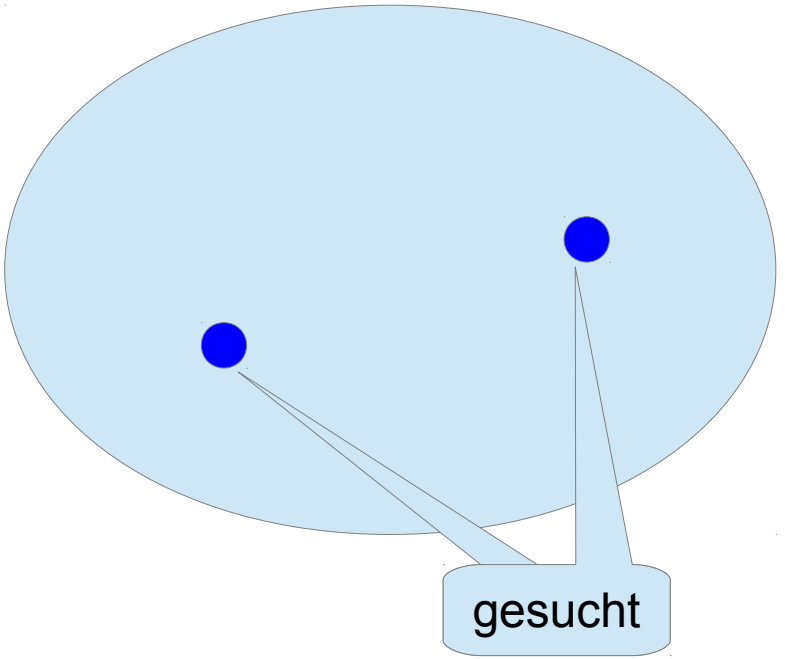
Definition Eine stark kollisionsresistente Einweg-Hashfunktion heißt *kryptographische Hashfunktion*.

Schwache und starke Kollisionsresistenz

schwach:
finde spezifische Kollision



stark:
finde irgendeine Kollision



Zum Nachdenken

- Ist jede schwach kollisionsresistente Hashfunktion stark kollisionsresistent?
  - Wenn ja, warum?
- Beschreibe ein nicht kollisionsresistente Einweg-Hashfunktion

Ideale kryptographische Hashfunktionen

- Charakterisierung (Ferguson et al): Eine ideale Hashfunktion verhält sich wie eine Zufallsabbildung.
- Eine Attacke auf eine Hashfunktion: eine nicht generische Methode, die einen Unterschied zu einer idealen Hashfunktion findet.
  - nicht-generisch: nutzt spezifische Eigenschaft der zu attackierenden Hashfunktion

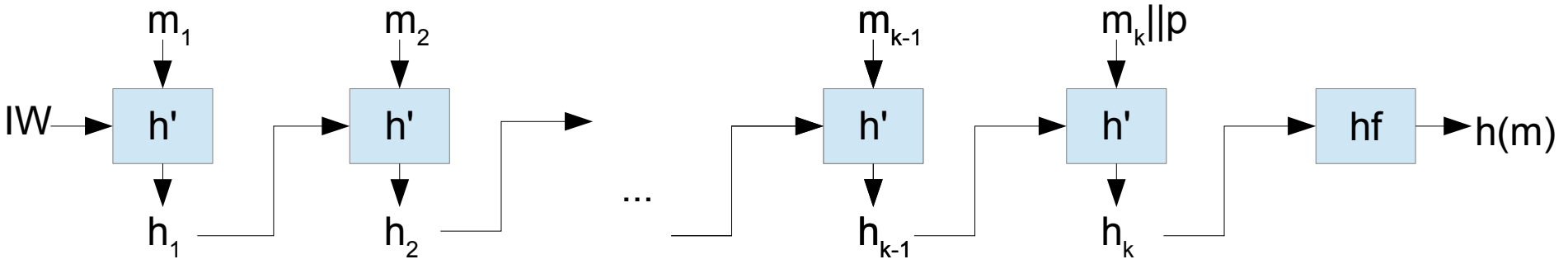
Das Geburtstagsparadox

- **Satz** Zu beliebiger Hashfunktion h: $\sum^* \rightarrow \sum^n$ gibt es einen Algorithmus, der zur Berechnung von $h^{-1}(y)$ für ein zufällig gewähltes $y \in \sum^n$ im Durchschnitt 2^{n-1} Hashoperationen benötigt
- **Satz** (*Geburtstagsparadoxon*) Zu beliebiger Hashfunktion h: $\sum^* \rightarrow \sum^n$ ist die Wahrscheinlichkeit P(n,k), dass eine Menge $M \in \sum^*$ mit $|M| = k \geq 1{,}2 \cdot 2^{n/2}$ zwei Elemente m, m' $\in \sum^*$ mit $m \neq m'$ und $h(m) = h(m')$ enthält, größer als 0,5.
- Beweis: $1-P(n,k)=\prod_{i=0}^{k-1}\frac{2^n-i}{2^n}=\prod_{i=1}^{k-1}1-\frac{i}{2^n}\leq\prod_{i=1}^{k-1}e^{\frac{-i}{2^n}}\leq e^{\frac{-(k-1)^2}{2\cdot 2^n}}\leq e^{\frac{-(k-1)^2}{2\cdot 2^n}}$ da $1-x\leq e^{-x}$

$$1-P(n,1,2\cdot 2^{n/2}+1)\leq e^{-\frac{2\cdot 2^n\cdot\ln 2}{2\cdot 2^n}}=e^{-\ln 2}=1/2$$

- **Geburtstagsangriff**:
  - Berechne die Hashwerte von $1{,}2 \cdot 2^{n/2}$ zufällig gewählten Elementen aus $\sum^*$.

Typische Struktur von Hashfunktionen



- Hashfunktion $h: \sum^* \rightarrow \sum^n$ wird iterativ mit elementarer Hashfunkton $h': \sum^k \times \sum^b \rightarrow \sum^k$ berechnet mit k >= n
- $hf: \sum^k \rightarrow \sum^n$ ist eine finale Filterfunktion
- Teile Nachricht m in Blöcke der Länge b auf:
  - $m = m_1 \,||\, m_2 \,||\, \ldots \,||\, m_{k-1} \,||\, m_k$ mit $|m_i| = b$ für $i<k$ und $|m_k| \leq b$
- sei $p \in \sum^*$ mit $|p| = b-|m_k|$ ein Padding
- sei $IW \subseteq \sum^k$ ein fester Initialisierungswert
- $h_0 = IW$
- $h_i = h'(h_{i-1}, m_i)$ für $1 \leq i \leq k-1$
- $h_k = h'(h_{k-1}, m_k||p)$
- $h(m) = hf(h_k)$

Notation:
|| ist der Konkatenationsoperator

Einige wichtige Hashfunktionen

Name	Autor	Jahr	Block größe	Hashgrößen (bit)	Aufwand Kollision
MD4	Rivest	1990	521	128	1995: 2^{20} (heute 2?)
MD5	Rivest	1992	512	128	2009: 2^{16}
SHA-1	NSA	1995	512	160	2007: 2^{61}, 2017 erste Kollision mit 2 PDF Dateien
SHA-2	NIST	2002	512, 1024	224,256, 384,512 [1)]	
SHA-3 (Keccak)	Bertoni et al	2011	1600 - 2n	224 – 512	

Empfohlene sichere Hashlängen (www.keylength.com)
je nach Autor/Organisation

- für 2015: 156 – 224 bits (BSI 224 bits)
- für 2020: 163 – 384 bits (BSI 256 bits)
- für 2030: 176 – 384 bits
- für 2050: 203 – 512 bits

[1)] Hashgrößen passen zu 3DES, AES-128, AES-192, AES-256 Chiffren

Ein Maß für die Sicherheit

- die Anzahl der kryptographischen Operationen, die (im Durchschnitt) durchprobiert werden muss um eine kryptographische Funktion zu attackieren.
- Da diese Zahl sehr groß ist nimmt man ihren 2er-Logarithmus und nennt ihn *Sicherheitsniveau*.
- Die Masseinheit für das Sicherheitsniveau: Bits
- Heute wird meist ein Sicherheitsniveau von 128 Bits empfohlen.
- Das Sicherheitsniveau einer idealen Hashfunktion mit einem Hashwert der Länge n ist n/2.

Große Zahlen und Zeit

- 1 Zyklus auf 4GHz Rechner: $0.25 \cdot 10^{-9}s \approx 2^{-32}s$
- $1h = 3\,600s < 2^{12}s \approx 2^{44}$ Zyklen
- $1d = 86\,400s < 2^{17}s \approx 2^{49}$ Zyklen
- $1y = 31\,536\,000s \approx 2^{25}s \approx 2^{57}$ Zyklen

- Merke: $10^3 \approx 2^{10}$

Passwortverschlüsselungsverfahren - Methoden

- System hat Passwortdatenbank pdb: $T \rightarrow EP$, die jedem autorisierten Teilnehmer $t \in T$ ein verschlüsseltes Passwort in EP zuweist.
- Passwort $p_t \in P$ des Teilnehmers t wird mit Passwortverschlüsselungsfunktion ep: $P \rightarrow EP$ verschlüsselt und mit hinterlegten verschlüsselten verglichen: $ep(p_t) = pdb(t)$?
- Folgerung
  - ep^{-1}:$EP \rightarrow P$ mit $ep^{-1}(ep(p)) = p$ muss *praktisch nicht berechenbar* (also sehr aufwendig) sein → *Einwegfunktion*
  - ep muss jedoch mit vertretbarem Aufwand berechenbar sein

Passworthashfunktionen

- bekannte Funktion gemäß Kerkhoffs Prinzip
- Passwortverschlüsselung: kryptographische Hashfunktion
- dasselbe Passwort muss auf unterschiedlichen Systemen unterschiedliche Hashes ergeben: Salz (Pfeffer)
- sie muss schwer zu knacken sein: einstellbarer Kostenfaktor
  - Zeit: mehrfaches Iterieren einer zugrundeliegenden Hashfunktion
    - z.B. PBKDF2 aus PKCS#5 v2 oder bcrypt
  - Speicher
    - z.B. Argon2 Familie

PBKDF2

- Password Based Key Derivation Function 2
- PBKDF2: $\sum^* \times \sum^* \times \mathbf{N} \times \mathbf{N} \rightarrow \sum^*$ berechnet Passworthash (bzw den Schlüssel)
- F: $\sum^* \times \sum^* \times \mathbf{N} \times \mathbf{N} \rightarrow \sum^n$ berechnet Schlüsselteil der Länge n
- PRF: $\sum^* \times \sum^* \rightarrow \sum^n$ berechnet schlüsselabhängigen Hashwert einer Nachricht
- PBKDF2(pw,salt,iter,keylen) = $T_1||...||T_{m-1}||msb(T_m,r)$
- mit
  - T_j = F(pw,salt,iter,j)
  - m= $\lceil$ keylen/n $\rceil$; r = keylen – (m - 1)•n
  - F(p,s,it,j) =$U_1 \oplus ... \oplus U_{it}$ mit
    - U_1=PRF(pw, salt || INT(j))
    - U_i=PRF(pw,U_{i-1})

Notation:
N sind die natürlichen Zahlen
$\oplus$ ist der bitweise xor Operator
msb(m,r) r signifikanteste Bits von m

Wie werden Passwörter geknackt (1)

- online Angriff
  - Passwort raten + Versuch einzuloggen
- offline Angriff
  - Gegeben ein verschlüsseltes Passwort suche Urbild zu bekannter Passworthashfunktion
  - Gegeben eine Liste verschlüsselter Passwörter suche Urbild zu einem der verschlüsselten Passwörter

Gegenmaßnahmen

- online Angriff:
  - erlaube nur endliche Zahl von Fehlversuchen
  - forciere nach einem Fehlversuch eine Pause, Pausen werden mit jedem Fehlversuch größer
  - jailing: vorspielen eines gelungenen Einbruchs
  - CAPTCHAs (Completely Automated Public Turing Test to tell Computers and Humans Apart): verhindern automatischer Angriffe
- offline Angriff
  - Passworthashes (/etc/shadow) dürfen nur autorisierten Lesern zugänglich sein
  - langsame Passworthashfunktionen
  - (systemspezifisch) pfeffern
  - (nutzerspezifisch) salzen

Wie werden Passwörter geknackt (2)

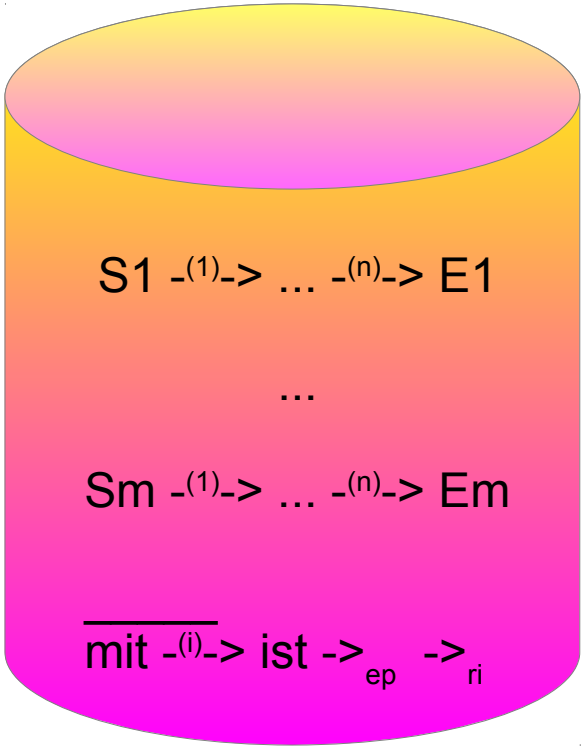
- Katalog von vorberechneten Hash-Urbildern (rainbow tables)
- Passwortkataloge
  - beliebte Passwörter (z.B. 12345678, letmein, asdfghj)
  - Wörterbücher verschiedener Sprachen
  - aussprechbare Buchstabenketten
  - jemals geknackte Passwörter
- Variationen von Katalogelementen
  - Prefixe, Suffixe,
  - Ersetzung Buchstaben durch Ziffern, Sonderzeichen (z.B. i → 1, s → $, at → @, for → 4)
  - Transformationen: vorwärts, rückwärts
- Indexierung von Festplatten
- Nutzung persönlicher Informationen
  - Phishing

Rainbow Tables (1)

- Ziel: erstelle vorberechnete Tabelle von Passwörtern und Passworthashes
- Problem: Suchraum zu groß,
  - z.B. alle alphanumerische Passwörter der Länge 8: ca 1 PByte
- Lösung: kompakte Darstellung der Vorberechnung.
  - Speichere wenige vorberechnete Werte
  - auf Kosten teuererer Nachschlagefunktionen
  - Hashketten: StartPW $\rightarrow h_1 \rightarrow h_2 \rightarrow \ldots \rightarrow h_n$
  - speichere pro Hashkette nur StartPW und Kettenendwert h_n
  - $h_1, \ldots, h_{n-1}$ müssen jeweils berechnet werden
  - Passwortsuche für gegebenen Hashwert:
    - berechne Hashketten bis gespeicherter Kettenendwert gefunden,
    - Dann verfolge Hashkette ab gespeichertem StartPW

Rainbow Tables (2)

- Passwort Hashfunktion ep: P → EP
- Reduktionsfunktion r: EP → P
- einfache Hashkette:
  - $p_0 \rightarrow_{ep} h_0 \rightarrow_r p_1 \rightarrow_{ep} h_1 \rightarrow_r p_2 \rightarrow_{ep} \ldots \rightarrow_r p_n$
  - erlaubt Kollisionen: mehrere Hashketten haben das gleiche Ende
- rainbow table Hashketten
  - unterschiedliche Reduktionensfunktionen ri: EP → P für i □ {0,…,n}
  - $p_0 \rightarrow_{ep} h_0 \rightarrow_{r0} p_1 \rightarrow_{ep} h_1 \rightarrow_{r1} p_2 \rightarrow_{ep} \ldots \rightarrow_{r\,n} p_n$
  - Suche Passwort zu Passworthash h:
    - für i von 0 bis n:
      - berechne Teilhashkette ab Position i mit $h_i = h$
      - falls Endwert der berechneten Teilhashkette gleich einem Endwert einer vorberechneten Teilhashkette ist: verfolge diese vorberechnete Kette und bestimme p_i



Suche: für alle i
h -(i)-> ... -(n)-> E*

if E* = Ek then
Sk -(1)-> ... -(i-1)-> P

Zum Nachdenken

- Welche Angriffe werden von rainbow tables unterstützt?
- Welche Komponenten von PBKDF2 schützen vor dem Einsatz von rainbow tables?
- Welche Eigenschaften zeichnen eine gute Reduktionsfunktion aus?

Gute Passwörter

- Lang!
- Großer Zeichensatz: Buchstaben (groß und klein), Zahlen Sonderzeichen
- echte Zufalls Wörter ($62^8 > 10^{14}$)
  - Kontospezifisch verziert
- Anfangsbuchstaben von *persönlichen* Phrasen
  - Schneier: "This little piggy went to market" -> "tlpWENT2m"
  - keine Shakespearezitiate!
- Best Practices gemäß NIST SP 800-63-3
  - Passphrasen statt komplexer Passwörter
  - kein Ablauf der Passwortgültigkeit
    - solange PW nicht kompromittiert
  - nutze Passwort-Manager
- Passwortwiederherstellungsfragen:
  - Antworten dürfen nicht unsicherer sein als Passwort → lügen!

Beispiele gefundener Passwörter

ilovemySister31

windermere2313

k1araj0hns0n

Qbesancon321

gonefishing1125

Apr!l221973

ilovetofunot

allineedislove

Sh1a-labe0uf

tmdmmj17

all of the lights

iloveyousomuch

qeadzcwrsfxv1331

@Yourmom69

DG091101%

BandGeek2014

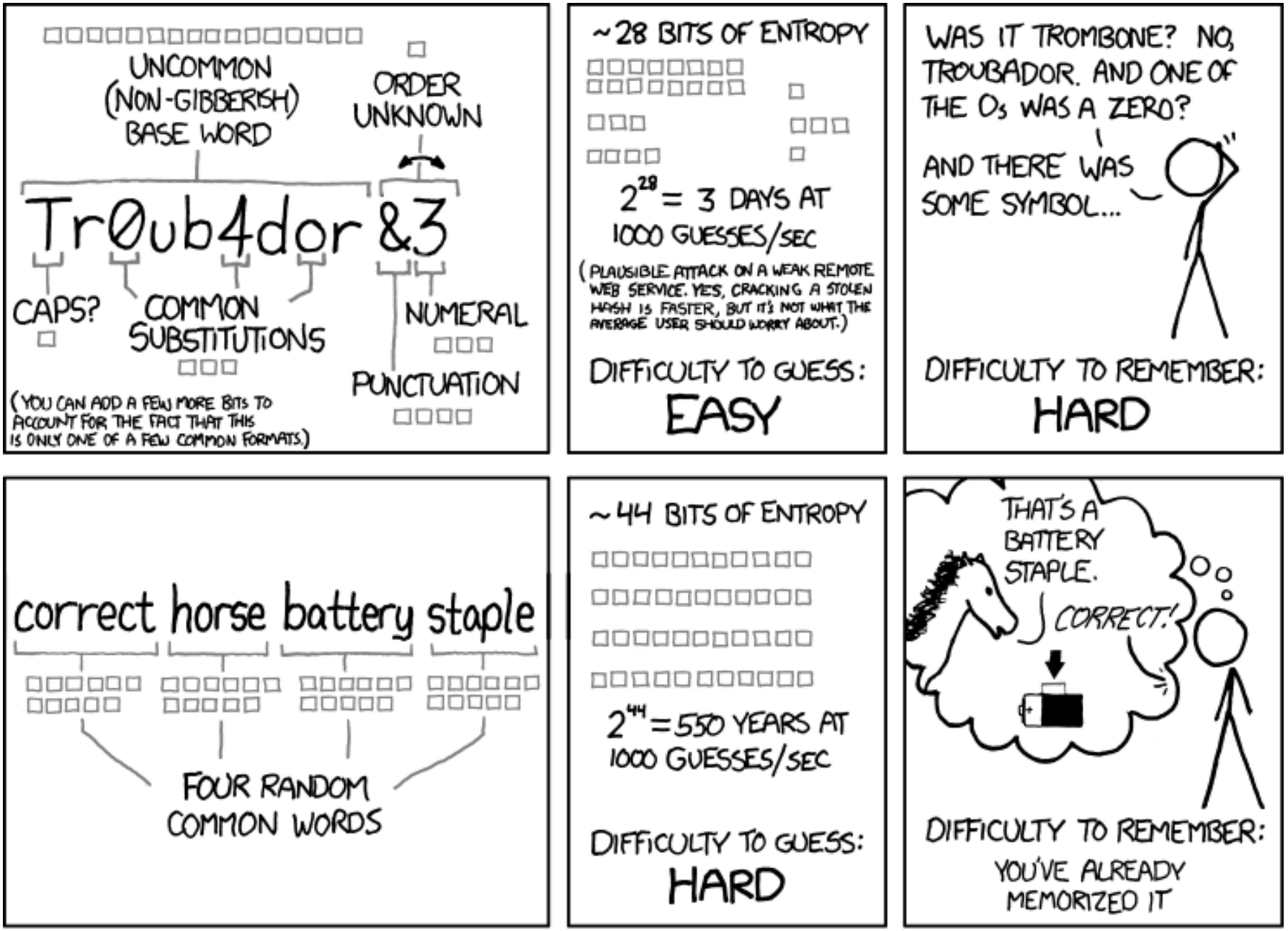
Philippians4:13

Philippians4:6-7

i hate hackers

Schneier newsletter 03/2014: „Last year, Ars Technica gave three experts a 16,000-entry encrypted password file, and asked them to break as many as possible. The winner got 90% of them, the loser 62% -- in a few hours.“

https://imgs.xkcd.com/comics/password_strength.png

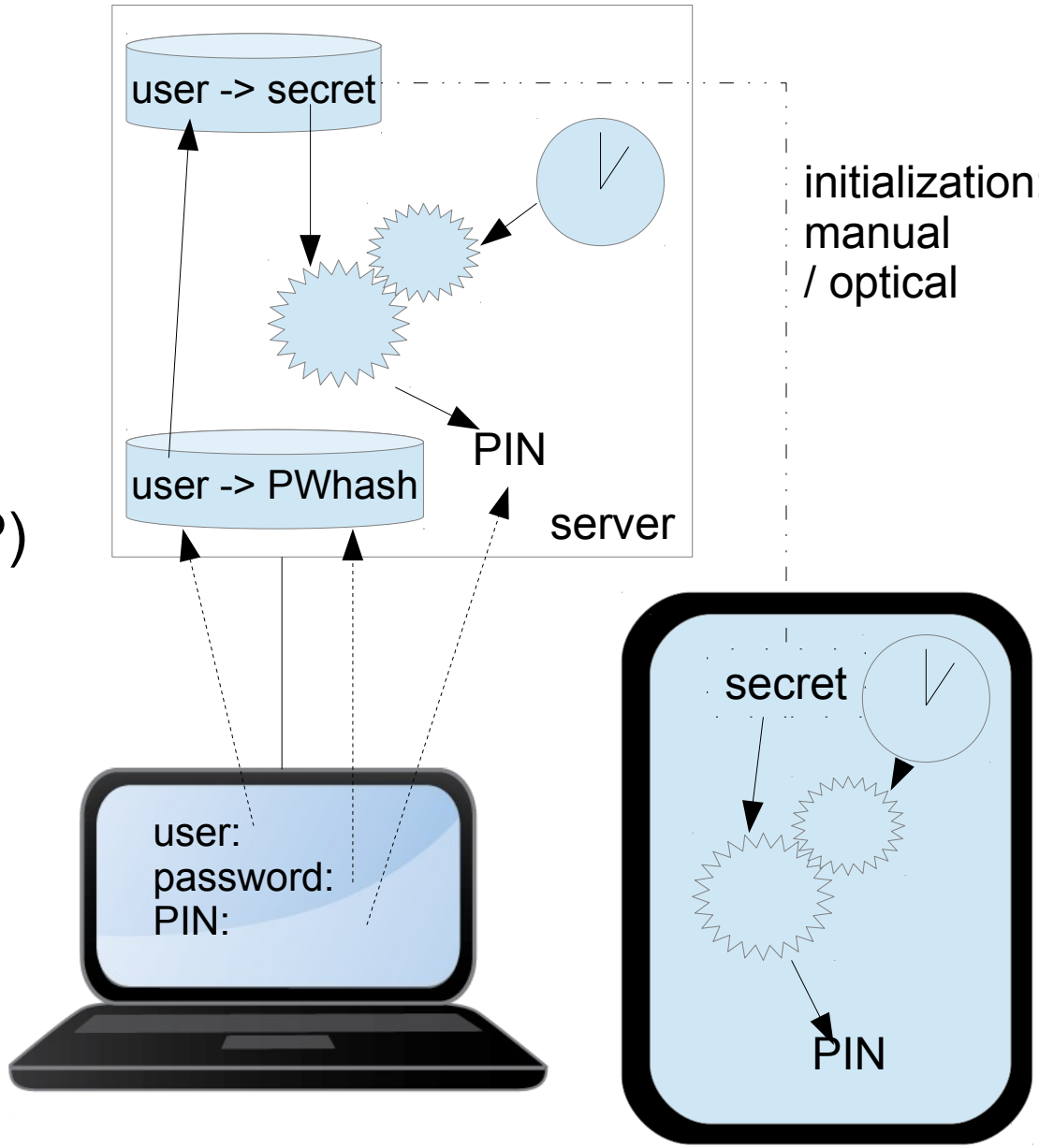


THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Multi-Faktor-Authentisierung (MFA)

Beispiel: Google Authenticator

- App basiert
  - app teilt Teilnehmer-geheimnis mit Google-Authentication-Server
- 2FA Protokoll (OATH-TOTP)
- Einmal-Passwörter
- abhängig von Teilnehmer-geheimnis und Zeit
- nur 30sec gültig



Aufgaben

- Richten Sie zu einem System Ihrer Wahl einen ssh Login mit beidseitiger Authentifizierung ein.
- Finden Sie heraus wie auf Ihrem System die Passwörter „verschlüsselt“ werden.
- Beschreiben Sie für eine reale Anwendung mit MFA, wie die MFA abläuft. Welche Faktoren werden genutzt?
- Welche Eigenschaften hat PBKDF2?