Vorlesung Sicherheit

Dennis Hofheinz

ITI, KIT

26.06.2014

Überblick

Ergebnisse der Vorlesungsbefragung

- Sehr viele nützliche Kommentare, **vielen Dank!**
- Bewertung in Zahlen positiv
  - Bei „Anschaulichkeit“ und „Arbeitsaufwand“ große Streuung
- Gut gefallen hat:
  - Seitenkanal-VL, Verweis auf Forschung/Anwendung
  - Interaktion/Gelegenheit zu Fragen
  - Details: Fragen wiederholen, Zusammenfassungen, Doktor Meta

Ergebnisse der Vorlesungsbefragung

- Nicht gut gefallen hat:
  - „Pseudo-Formalismus“
  - Zuwenige konkrete Anwendungen/aktuelle Forschung
  - „Big Picture“ fehlt teilweise
  - Einige Grundlagen nicht wiederholt (Gruppen, Graphen)
  - Tafelbilder nicht digital verfügbar, sonst wenige Bilder
  - Übungen zu schwer (sind Klausuren auch so?)
  - Details: „manchmal Herumreden“, Verfahren ungenau/nicht ausführlich beschrieben, Reihenfolge Theorie $\rightarrow$ Praxis

Ergebnisse der Vorlesungsbefragung

- Feedback von mir:
  - Interaktion ist rege und gut (aber gerne mehr!)
  - Mehr eigene Wünsche einbringen (geeignetes Forum?)

Überblick

Überblick

Motivation

- **Szenario:** Nutzer U möchte sich auf Server S einloggen
- **Frage:** Was zeichnet U aus?
  - Üblich: U kennt Passwort $\mathtt{pw}$
  - Möglich: biometrische Merkmale (Fingerabdruck, Irisscan)
  - Etwas exotischer: Wissen/Fähigkeiten von U
- **Annahme zunächst:** U kennt Passwort $\mathtt{pw}$
- **Frage:** Wie könnte/sollte Überprüfung aussehen?

Erster Versuch

- Erster Versuch:

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathtt{pw}}$$

  (Server vergleicht $\mathtt{pw}$ mit für U gespeichertem Passwort)
- **Frage:** Warum nicht optimal?
- **Antwort:** Server kennt $\mathtt{pw}$
  - Problematisch, weil Server *alle* $\mathtt{pw}$ kennt
  - Server korrumpiert $\Rightarrow$ alle $\mathtt{pw}$ öffentlich
  - Heikel: Nutzer verwenden $\mathtt{pw}$ evtl. auch bei anderen Diensten

Zweiter Versuch

- Zweiter Versuch (schon angerissen bei Hashfunktionen):

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathrm{H}(\mathtt{pw})} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

- **Idee:** Server speichert nur $\mathrm{H}(\mathtt{pw})$ für Hashfunktion H
- **Frage:** Verbesserung?
  - Server muss nur $\mathrm{H}(\mathtt{pw})$ (und nicht $\mathtt{pw}$) speichern
  - Bringt keine Verbesserung, weil $\mathrm{H}(\mathtt{pw})$ schon hinreichend!
- Besser:

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

  (Server wendet H auf Nutzereingabe an und vergleicht)

Ziel

- **Ziel:** Das Verfahren

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

näher beleuchten (Angriffe, Verbesserungen, Implementierungen)

Überblick

Angriffsziel

- **Szenario:** Server korrumpiert, (Liste von) $\mathrm{H}(\mathtt{pw})$ bekannt
- **Angriffsziel:** Finde $\mathtt{pw}$
- Bei sicherer Hashfunktion H schwierig, H zu invertieren...
- ... wenn H-Eingabe $\mathtt{pw}$ gleichverteilt gewählt
- **Aber:** Passwörter üblicherweise nicht gleichverteilt

Häufige Passwörter

- **Passwort-Hitliste:** (englischsprachig, 2012, Quelle: SplashData)

10	`baseball`	5	`qwerty`
9	`111111`	4	`abc123`
8	`dragon`	3	`12345678`
7	`letmein`	2	`123456`
6	`monkey`	1	`password`

- **Kommen diese Passwörter oft vor?** (Quelle: Mark Burnett)
  - 91% nutzen eins der 1000 häufigsten Passwörter
  - 99,8% nutzen eins der 10000 häufigsten Passwörter

Überblick

Wörterbuchangriffe

- **Gegeben:** $\mathrm{H}(\mathtt{pw})$ und ein Wörterbuch
- **Ziel:** Finde $\mathtt{pw}$
- **Vorgehen:** Vergleiche $\mathrm{H}(\mathtt{pw})$ mit $\mathrm{H}(\mathtt{pw}')$ für alle Wörterbucheinträge $\mathtt{pw}'$
- **Problem:** Sehr langsam, wenn Wörterbuch riesig
- **Hier: einige Annahmen über Passwort**
  - Server erzwingt nichttriviale Passwörter
  - Passwort nicht trivial ratbar ($\Rightarrow$ Wörterbuch groß)

Wörterbuchangriff

- **Problem:** Wörterbuchangriff langsam
- **Lösung:** *Sortiere* Wörterbuch nach Hashwert:

$$\begin{array}{lcl} (\quad \mathrm{H}(\mathtt{pw}_1) & , & \mathtt{pw}_1 \quad) \\ (\quad \mathrm{H}(\mathtt{pw}_2) & , & \mathtt{pw}_2 \quad) \\ \quad \ldots & & \ldots \end{array}$$

  mit $\mathrm{H}(\mathtt{pw}_1) < \mathrm{H}(\mathtt{pw}_2) < \ldots$
- Sortieren einmal nötig, danach binäre Suche möglich
- Allerdings: hoher Speicherplatzbedarf, wenn Liste lang

Überblick

Wörterbuchangriff

- **Problem:** Wörterbuch zu groß
- **Idee:** Tausche Speicherplatz gegen Rechenzeit (Time-Memory-Tradeoff)
- **Genauer:** Verlagere Arbeit
  - Wörterbuch verkleinert
  - Fehlende Teile beim Angriff selektiv rekonstruiert
  - Schwierigkeit: nur *benötigte* Teile sollen rekonstruiert werden

Grundidee

- **Grundidee:** betrachte *Ketten* von Hash-Passwort-Paaren

$$(\mathrm{H}(\mathtt{pw}_{1,1}), \mathtt{pw}_{1,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,2}), \mathtt{pw}_{1,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,m}), \mathtt{pw}_{1,m})$$
$$(\mathrm{H}(\mathtt{pw}_{2,1}), \mathtt{pw}_{2,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{2,2}), \mathtt{pw}_{2,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{2,m}), \mathtt{pw}_{2,m})$$
$$\ldots$$
$$(\mathrm{H}(\mathtt{pw}_{n,1}), \mathtt{pw}_{n,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,2}), \mathtt{pw}_{n,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,m}), \mathtt{pw}_{n,m})$$

- Hier f Funktion, die nächstes Tupel aus vorherigem ableitet
- Ergibt matrixartige Anordnung, spart so noch keinen Platz
- Tatsächlich geht Sortierung verloren, Angriff unklar

Grundidee

- **Reparieren:** so funktioniert Angriff jetzt

$$(\mathrm{H}(\mathtt{pw}_{1,1}), \mathtt{pw}_{1,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,2}), \mathtt{pw}_{1,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,m}), \mathtt{pw}_{1,m})$$

$$\ldots$$

$$(\mathrm{H}(\mathtt{pw}_{n,1}), \mathtt{pw}_{n,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,2}), \mathtt{pw}_{n,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,m}), \mathtt{pw}_{n,m})$$

- Sortiere Liste initial nach Endpunkten $\mathrm{H}(\mathtt{pw}_{i,m})$
- Wähle f, das nur von Hashwert abhängt
- Angreifer kann nun gegebenes $\mathrm{H}(\mathtt{pw}^*)$ mit f weiterentwickeln:

$$\mathrm{H}(\mathtt{pw}^*) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_1^*), \mathtt{pw}_1^*) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_2^*), \mathtt{pw}_2^*) \xrightarrow{f} \ldots$$

. . . und in der Liste der Endpunkte (effizient binär!) suchen

Grundidee

- Angenommen, Angreifer findet gemeinsamen Endpunkt:

$$(\mathrm{H}(\mathtt{pw}_{i,1}), \mathtt{pw}_{i,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{i,2}), \mathtt{pw}_{i,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{i,m}), \mathtt{pw}_{i,m})$$
$$\mathrm{H}(\mathtt{pw}^*) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_1^*), \mathtt{pw}_1^*) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{m^*}^*), \mathtt{pw}_{m^*}^*)$$

  mit $\mathtt{pw}_{m^*}^* = \mathtt{pw}_{i,m}$ und $m^* \leq m$
- Angreifer geht i-te Kette in Liste von vorn durch
- Hoffnung: $(\mathrm{H}(\mathtt{pw}^*), \mathtt{pw}^*)$ taucht auf (dann $\mathtt{pw}^*$ gefunden)
- Problem bei f-Kollision

  (dann gemeinsames Kettenende bei unterschiedlichen Ketten möglich)

Wie die Liste komprimiert wird

- **Beobachtung:** nur Anfangs- und Endpunkte der Liste

$$(\mathrm{H}(\mathtt{pw}_{1,1}), \mathtt{pw}_{1,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,2}), \mathtt{pw}_{1,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{1,m}), \mathtt{pw}_{1,m})$$

$$\ldots$$

$$(\mathrm{H}(\mathtt{pw}_{n,1}), \mathtt{pw}_{n,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,2}), \mathtt{pw}_{n,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{n,m}), \mathtt{pw}_{n,m})$$

  benötigt (benötigter Rest kann on-the-fly generiert werden)

- **Konsequenz:** Liste kann auf

$$(\mathrm{H}(\mathtt{pw}_{1,1}), \mathtt{pw}_{1,1}) \quad , \quad (\mathrm{H}(\mathtt{pw}_{1,m}), \mathtt{pw}_{1,m})$$

$$\ldots$$

$$(\mathrm{H}(\mathtt{pw}_{n,1}), \mathtt{pw}_{n,1}) \quad , \quad (\mathrm{H}(\mathtt{pw}_{n,m}), \mathtt{pw}_{n,m})$$

  komprimiert werden (spart Faktor $m/2$ an Speicherplatz)

Anforderungen an f

- Funktion f sollte einige Anforderungen erfüllen
  - f sollte „sinnvolle“ Passwörter (mit Hashes) ausgeben
  - $f((\mathrm{H}(\mathtt{pw}), \mathtt{pw}))$ sollte nur von $\mathrm{H}(\mathtt{pw})$ abhängen
    (damit Angreifer f auf gegebenes $\mathrm{H}(\mathtt{pw}^*)$ anwenden kann)
  - f sollte „hinreichend“ kollisionsresistent sein
  - Insbesondere: f sollte Passwortraum „gut abdecken“
  - Spagat: f willkürlich genug, um Kollisionen zu vermeiden, aber nicht zu willkürlich, um häufige Passwörter zu finden
- Mögliches f:
  - f nimmt die ersten 40 Bits von $\mathrm{H}(\mathtt{pw})$, „arrangiert“ diese in eine sinnvolle Buchstaben-Sonderzeichen-Zahlenkombination $\mathtt{pw}'$ und gibt $(\mathrm{H}(\mathtt{pw}'), \mathtt{pw}')$ aus

Überblick

Rainbow Tables

- Problem komprimierter Listen: „Überlappung“ von Ketten

$$(\mathrm{H}(\mathtt{pw}_{i,1}), \mathtt{pw}_{i,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{i,2}), \mathtt{pw}_{i,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{i,m}), \mathtt{pw}_{i,m})$$
$$(\mathrm{H}(\mathtt{pw}_{j,1}), \mathtt{pw}_{j,1}) \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{j,2}), \mathtt{pw}_{j,2}) \xrightarrow{f} \ldots \xrightarrow{f} (\mathrm{H}(\mathtt{pw}_{j,m}), \mathtt{pw}_{j,m})$$

- Sobald $\mathtt{pw}_{i,u} = \mathtt{pw}_{j,v}$ (bel. u, v) ist, sind Restzeilen gleich
- Viele Überlappungen $\Rightarrow$ schlechte Abdeckung der Liste
- Andererseits sollte f nicht „zu zufällig“ sein
- **(Mögliche) Lösung:** benutze verschiedene f_u (u = Spalte)

Rainbow Tables

- **(Mögliche) Lösung:** benutze verschiedene f_u ($u =$ Spalte)

$$
\begin{aligned}
&(\mathrm{H}(\mathtt{pw}_{1,1}), \mathtt{pw}_{1,1}) \xrightarrow{f_1} (\mathrm{H}(\mathtt{pw}_{1,2}), \mathtt{pw}_{1,2}) \xrightarrow{f_2} \ldots \xrightarrow{f_{m-1}} (\mathrm{H}(\mathtt{pw}_{1,m}), \mathtt{pw}_{1,m}) \\
&(\mathrm{H}(\mathtt{pw}_{2,1}), \mathtt{pw}_{2,1}) \xrightarrow{f_1} (\mathrm{H}(\mathtt{pw}_{2,2}), \mathtt{pw}_{2,2}) \xrightarrow{f_2} \ldots \xrightarrow{f_{m-1}} (\mathrm{H}(\mathtt{pw}_{2,m}), \mathtt{pw}_{2,m}) \\
&\qquad\qquad\qquad\qquad \ldots \\
&(\mathrm{H}(\mathtt{pw}_{n,1}), \mathtt{pw}_{n,1}) \xrightarrow{f_1} (\mathrm{H}(\mathtt{pw}_{n,2}), \mathtt{pw}_{n,2}) \xrightarrow{f_2} \ldots \xrightarrow{f_{m-1}} (\mathrm{H}(\mathtt{pw}_{n,m}), \mathtt{pw}_{n,m})
\end{aligned}
$$

- **Effekt:** $\mathtt{pw}_{i,u} = \mathtt{pw}_{j,v}$ zieht nur bei $u = v$ Kollision der „Restzeilen“ nach sich $\Rightarrow$ größere Abdeckung der Liste
- **Namensgebung:** f_j und j-te Spalte der Liste haben „Farbe“ j
- **Nachteil:** Angriff muss alle Farben von $\mathrm{H}(\mathtt{pw}^*)$ durchprobieren

Überblick

Gesalzene Hashwerte

- Einfache (und seit langem bekannte) Gegenmaßnahme:

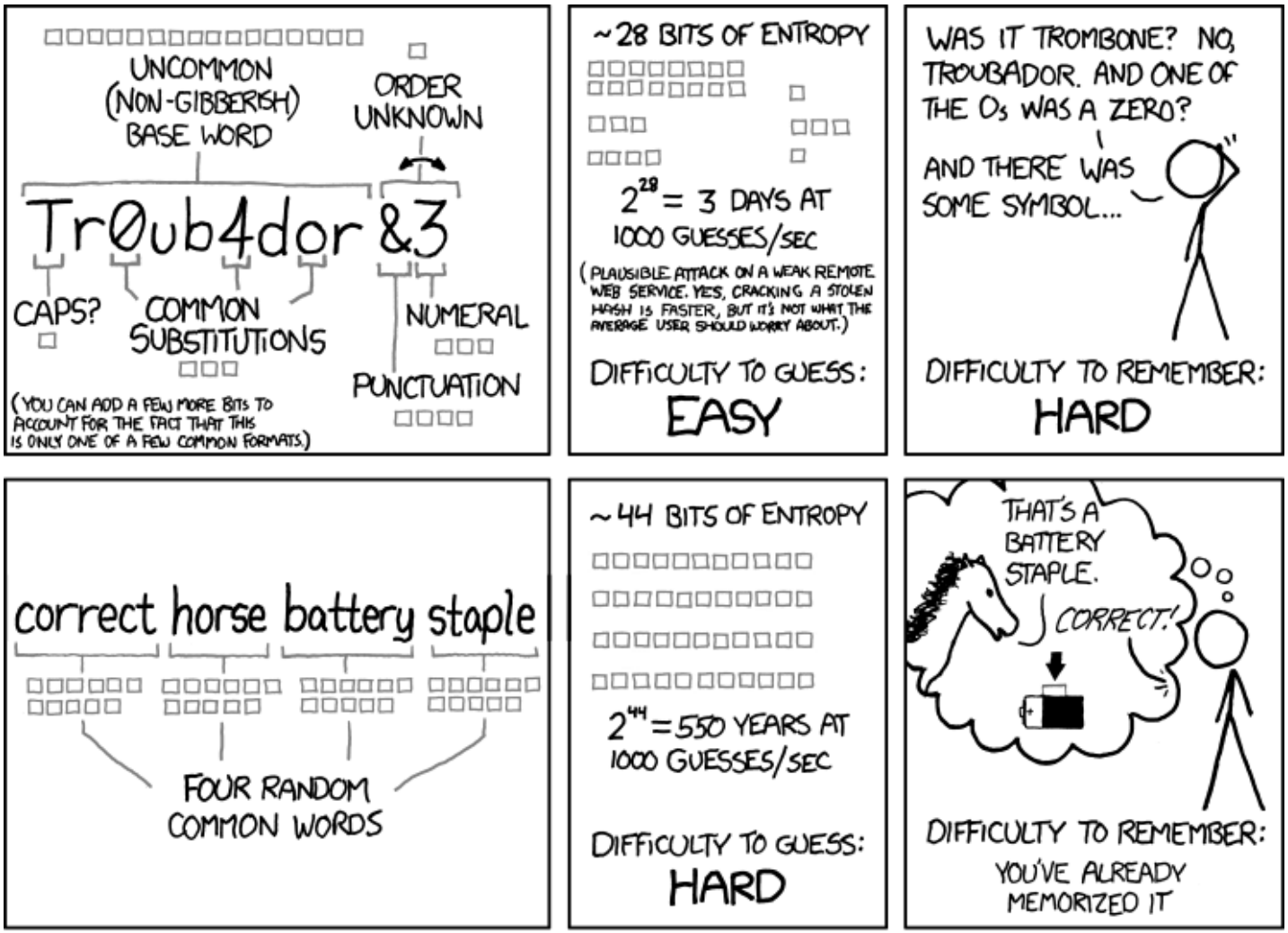
$$\mathrm{U}_{\texttt{pw}} \xrightarrow{\texttt{pw}} \mathrm{S}_{(\texttt{salt},\mathrm{H}(\texttt{salt},\texttt{pw}))}$$

  mit Zufallszahl `salt` „gesalzener“ Hashwert
- „Individualisiert“ `pw`, verhindert Wörterbuchangriffe (einschließlich Angriffen mittels Rainbow Tables)

Key Strengthening

- Weitere einfache Gegenmaßnahme: Key Strengthening
- **Idee:** erschwere H-Auswertung (für Angreifer *und* Server)
- Beispiel: $\mathrm{H}'(\mathtt{pw}) = \underbrace{\mathrm{H}(\mathrm{H}(\ldots\mathrm{H}}_{\text{1000 Mal}}(\mathtt{pw})\ldots))$
- Oder: $\mathrm{H}'(\mathtt{pw}) = \mathrm{H}(\mathtt{pw}, i)$ für kleinstes $i \in \mathbb{N}$, so dass $\mathrm{H}(\mathtt{pw}, i)$ mit 20 Nullen beginnt (benötigt Durchprobieren $\approx 2^{20}$ vieler i)
- Verlangsamt Angriff, verlangsamt aber auch Authentifikation
- Benutzt zum Beispiel von Truecrypt

Bessere Passwortwahl (?)



(xkcd.com)

Überblick

Parallelisierte Brute-Force-Angriffe

- **Idee:** suche systematisch „plausible“ Passwörter ab
- Lange Zeit praktisch aufwändiger als z.B. Rainbow Tables
- **Aber:** bei geschickter Aufteilung des Passwortraums parallelisierbar
  - Insbesondere interessant für Graphics Processing Units (GPUs)
  - Viele kleine *programmierbare* Einheiten auf GPU, die einfache Aufgaben (z.B. H-Auswertung) übernehmen können
  - Mittlerweile (2013) effizienter als Rainbow Tables
- **Frage:** welche Gegenmaßnahmen (Salt, Key Strengthening, „bessere Passwörter“) helfen hier?

Überblick

Implementierungen

- Windows (seit 1980er, $\leq$ ME) verwendet „LM-Hashes“ $\mathrm{H}(\mathtt{pw})$

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

  mit DES-basierter Hashfunktion H:
  - $\mathtt{pw}$ muss 14 Zeichen lang sein (ggf. aufgefüllt/abgeschnitten)
  - Kleinbuchstaben in $\mathtt{pw}$ durch Großbuchstaben ersetzt
  - $\mathtt{pw}$ wird in 7-Zeichen-Blöcke $\mathtt{pw}_1, \mathtt{pw}_2$ aufgeteilt, dann:

$$\mathrm{H}(\mathtt{pw}) := (\,\mathrm{Enc}(\mathtt{pw}_1, \mathtt{KGS!@\#\$\%}),\ \mathrm{Enc}(\mathtt{pw}_2, \mathtt{KGS!@\#\$\%})\,)$$

  - Erster Teil hängt nur von $\mathtt{pw}_1$ ab, zweiter nur von $\mathtt{pw}_2$
- **Frage:** Was könnte Angreifer tun, der nur $\mathrm{H}(\mathtt{pw})$ kennt?

Implementierungen

- Windows ($\leq$ XP) unterstützt und speichert noch LM-Hashes
- Ab Windows Vista LM-Hashes standardmäßig deaktiviert
- Neuere Windows-Versionen verwenden NTLM oder Kerberos
  - NTLM auch problematisch (nutzt schwache Verschlüsselung)
- Unix-artige Systeme (OS X, Linux, *BSD) verwenden seit etwa 1980 gesalzene Hashes:

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{(\mathtt{salt},\mathrm{H}(\mathtt{salt},\mathtt{pw}))}$$

- Hashfunktion H variiert (SHA-512, MD5, DES-basiert, . . .)
- **Probleme:** frühe Versionen beschränkt auf 8 Zeichen, Probleme mit Nicht-ASCII-`pw`

Überblick

Motivation

- Für manche Anwendungen (E-Banking) Passwörter nicht sicher genug
  - Kommunikation zwischen U und S unsicher (Internetcafe)
  - Wichtigstes Ziel: Replay-Angriffe vermeiden
- **Möglich:** *interaktive* Authentifikation zwischen U und S
- **Problem:** U allein nicht sehr berechnungsmächtig
- **Möglich:** U nutzt sichere Hardware (Chipkarte/Lesegerät)

1. $\mathrm{U} \xrightarrow{\texttt{pw}/\mathrm{PIN}} \mathrm{HW}_{K,\texttt{pw}} \xleftarrow{R} \mathrm{S}_K$
2. $\mathrm{U} \quad\quad\quad\quad \mathrm{HW}_{K,\texttt{pw}} \xrightarrow{\sigma := \mathrm{Sig}(K,R)} \mathrm{S}_K$

- Manchmal ohne pw/PIN (Nutzer setzt nur Chipkarte ein)