

Password Probability Matrix

UnFUG

BlueC0re

28. Oktober 2010

Inhaltsverzeichnis

- 1 Einführung
 - Regeln
 - Passwortlängenvergleich
 - crypt
- 2 PPM
 - Methode
 - Idee
 - Arbeitsweise
 - Beispiel
- 3 Fragen
- 4 Quellen

Regeln

Zeichensatz

- 1 Amerikanischer Zeichensatz (ASCII)
- 2 Keine Sonderzeichen (nur $< \text{ASCII } 127$)
- 3 Nur druckbare Zeichen

⇒ 95 Zeichen (Leerzeichen (ASCII 32) bis Tilde (ASCII 126))

Algorithmus

crypt3 Hashverfahren [Beschreibung](#)

Passwordlängenvergleich

Passwort- länge	Passwordvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte

Passwordlängenvergleich

Passwort- länge	Passwordvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB
4	$95^4 \approx 8.15e7$	2.26 Stunden	310.8 MiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB
4	$95^4 \approx 8.15e7$	2.26 Stunden	310.8 MiB
5	$95^5 \approx 7.74e9$	9 Tage	28.8 GiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB
4	$95^4 \approx 8.15e7$	2.26 Stunden	310.8 MiB
5	$95^5 \approx 7.74e9$	9 Tage	28.8 GiB
6	$95^6 \approx 7.35e11$	2.33 Jahre	2.7 TiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB
4	$95^4 \approx 8.15e7$	2.26 Stunden	310.8 MiB
5	$95^5 \approx 7.74e9$	9 Tage	28.8 GiB
6	$95^6 \approx 7.35e11$	2.33 Jahre	2.7 TiB
7	$95^7 \approx 6.98e13$	221.29 Jahre	254 TiB

Passwörtervergleich

Passwortlänge	Passwortvarianten	Dauer (10k/s)	Speicher
1	$95^1 = 95$	< 1 Sekunde	380 Byte
2	$95^2 = 9025$	0.9 Sekunde	32 KiB
3	$95^3 \approx 8.57e5$	85.7 Sekunden	3.3 MiB
4	$95^4 \approx 8.15e7$	2.26 Stunden	310.8 MiB
5	$95^5 \approx 7.74e9$	9 Tage	28.8 GiB
6	$95^6 \approx 7.35e11$	2.33 Jahre	2.7 TiB
7	$95^7 \approx 6.98e13$	221.29 Jahre	254 TiB
8	$95^8 \approx 6.63e15$	21022 Jahre	23.6 PiB

crypt

Was ist crypt?

- dient auf Unix-Systemen zum hashen von Passwörtern
- verwendet verschiedene Algorithmen:
 - 1 DES
 - 2 MD5
 - 3 Blowfish
 - 4 SHA

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: crypt(password, salt)
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: `crypt(password, salt)`
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: crypt(password, salt)
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: crypt(password, salt)
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: crypt(password, salt)
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

GNU C Library crypt3

Perl/C crypt

- verwendet DES + salt zum hashen
- Syntax: `crypt(password, salt)`
- Passwort wird auf 8 Zeichen runtergebrochen

Password	Hash
<i>aabbccdd</i>	<i>jeoLtAdXVaviw</i>
<i>aabbccdde</i>	<i>jeoLtAdXVaviw</i>

- base64-Encoded \Rightarrow Zeichensatz von 64 Zeichen
- Zusätzlicher Speicherbedarf pro Passwort/Salt: 13 Byte
- Anzahl der Hashes: $64^{13} \approx 0.30e24$

PPM

Methode

- Time attack: Bruteforce (alle Möglichkeiten ausprobieren)
- Space attack: Hashlookup (Passwörter nachschlagen)
- time/space trade-off attack
 - kleinerer key-space
 - weniger Speicherplatzverbrauch

PPM

Methode

- Time attack: Bruteforce (alle Möglichkeiten ausprobieren)
- Space attack: Hashlookup (Passwörter nachschlagen)
- time/space trade-off attack
 - kleinerer key-space
 - weniger Speicherplatzverbrauch

PPM

Methode

- Time attack: Brute force (alle Möglichkeiten ausprobieren)
- Space attack: Hashlookup (Passwörter nachschlagen)
- time/space trade-off attack
 - kleinerer key-space
 - weniger Speicherplatzverbrauch

PPM

Methode

- Time attack: Bruteforce (alle Möglichkeiten ausprobieren)
- Space attack: Hashlookup (Passwörter nachschlagen)
- time/space trade-off attack
 - kleinerer key-space
 - weniger Speicherplatzverbrauch

PPM

Methode

- Time attack: Brute force (alle Möglichkeiten ausprobieren)
- Space attack: Hashlookup (Passwörter nachschlagen)
- time/space trade-off attack
 - kleinerer key-space
 - weniger Speicherplatzverbrauch

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgröße: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgröße}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgröße: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgröße}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgrösse: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgrösse}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgrösse: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgrösse}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgröße: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgröße}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgrösse: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgrösse}}{8} \approx 141\text{MiB}$

PPM

Idee

- 3-dimensionale Matrix
 - x-Achse: Kombination aus 2 Passwortzeichen ($95^2 = 9025$)
 - y-Achse: 3-Zeichenchunk des Hashes ($64^2 * 4 = 16384$)
 - z-Achse: Vier chunks für jedes Passwortpaar ($4 * 2 = 8$)
 - Gesamtgrösse: 1182924800 Zellen
- Enthält nur einzelne Bits, Koordinaten liefern Zordnung von Hash und Passwort
- Speichergröße: $\frac{\text{Gesamtgrösse}}{8} \approx 141\text{MiB}$

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich: $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich:
 $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich:
 $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich:
 $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich:
 $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Arbeitsweise

- Basiert auf dem Schubfachprinzip: Stellt man k Objekte in n Fächer (mit $k > n$) dann enthält ein Fach min. 2 Objekte
- Für optimale Ergebnisse soll jeder Vektor zu 50% gesättigt sein \Rightarrow doppelt so viele Löcher wie Passwörter
- Benötigt: $\frac{95^4 * 2}{9025} = 18050$ Spalten
- Besitzen: $64^2 * 4 \approx 16000$ Spalten \Rightarrow 42% gesättigt
- Da zu jedem Hash 4 Vektoren existieren ergibt sich:
 $0.42^4 \approx 0.0311 = 3.11\%$
- Verringerung des key-spaces: $(9025 * 3.11\%)^2 \approx 280^2 = 78400$
 \Rightarrow dauert ca. 8s bei 10k Cracks/s (vgl 2.26 Stunden bei Bruteforce)

PPM

Beispiel

Passwort	Hash
test	jeHEAX1m66RV.
!J)h	jeHEA38vqlkkQ
".F+	jeHEA1Tbde5FE
"8,J	jeHEAnX8kQK3I
...	...

PPM

Erstellen

- 1 Passwort wird erstellt
- 2 Hash wird berechnet
- 3 Passwort wird paarweise gesplittet (z.B. te und st)
- 4 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 5 Koordinaten werden aus Paaren und Chunks gebildet

PPM

Erstellen

- 1 Passwort wird erstellt
- 2 Hash wird berechnet
- 3 Passwort wird paarweise gesplittet (z.B. te und st)
- 4 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 5 Koordinaten werden aus Paaren und Chunks gebildet

PPM

Erstellen

- 1 Passwort wird erstellt
- 2 Hash wird berechnet
- 3 Passwort wird paarweise gesplittet (z.B. te und st)
- 4 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 5 Koordinaten werden aus Paaren und Chunks gebildet

PPM

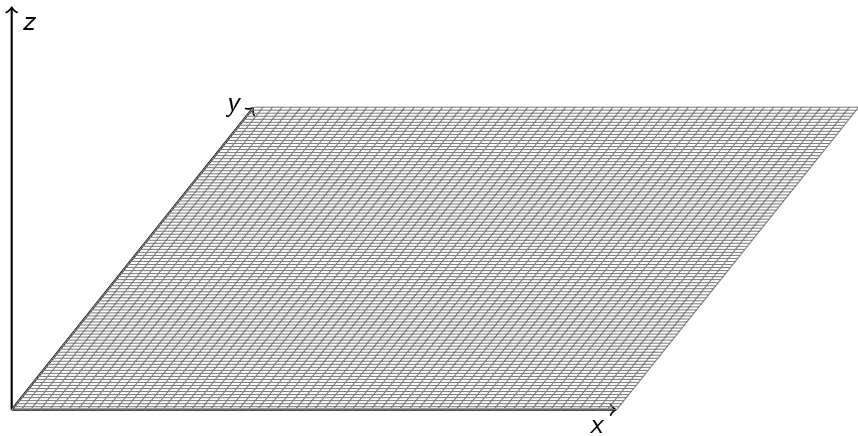
Erstellen

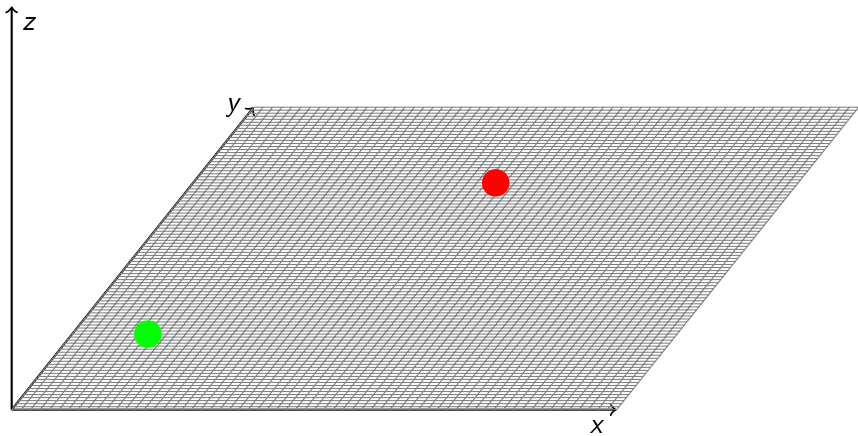
- 1 Passwort wird erstellt
- 2 Hash wird berechnet
- 3 Passwort wird paarweise gesplittet (z.B. te und st)
- 4 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 5 Koordinaten werden aus Paaren und Chunks gebildet

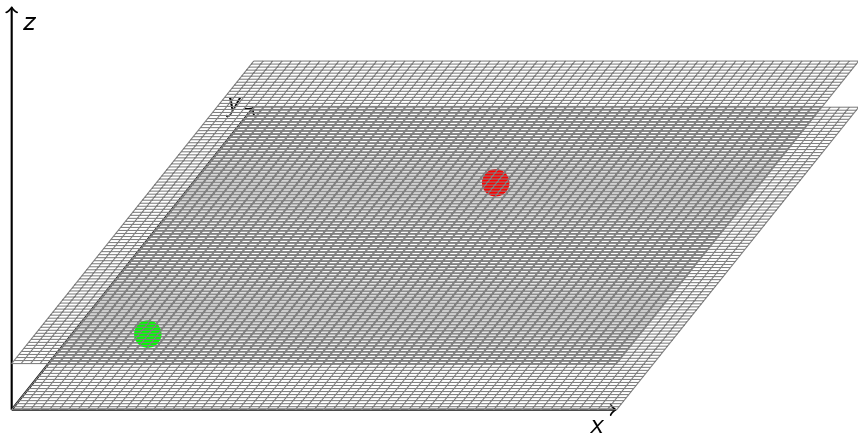
PPM

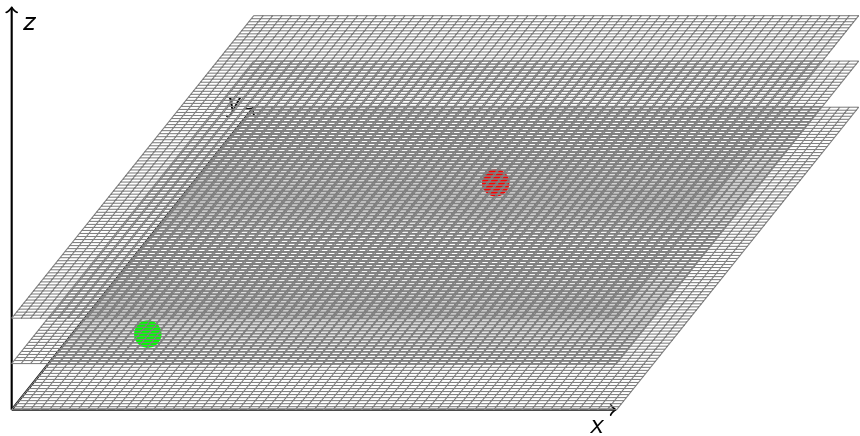
Erstellen

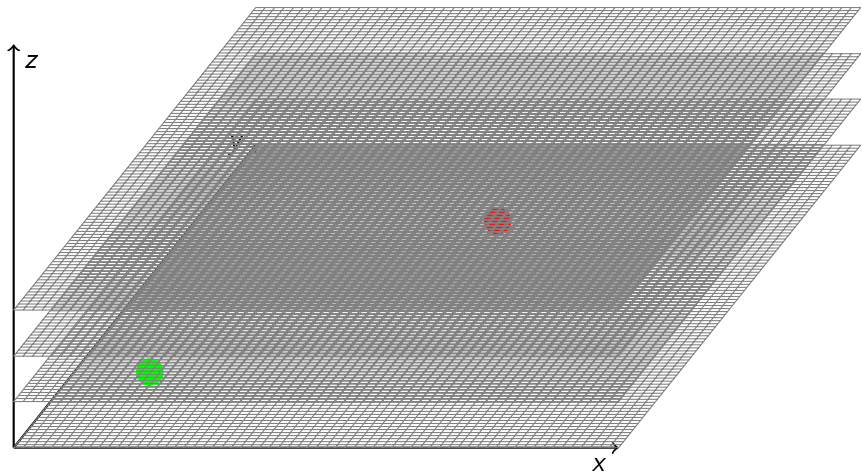
- 1 Passwort wird erstellt
- 2 Hash wird berechnet
- 3 Passwort wird paarweise gesplittet (z.B. te und st)
- 4 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 5 Koordinaten werden aus Paaren und Chunks gebildet

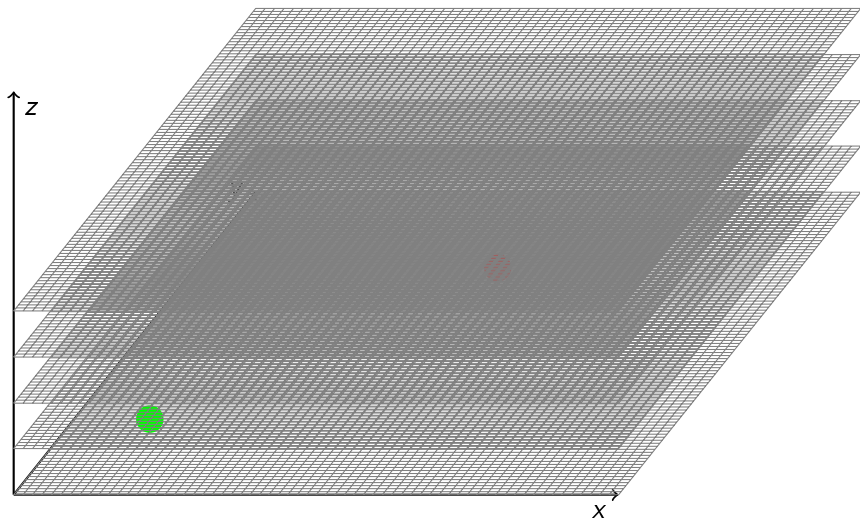


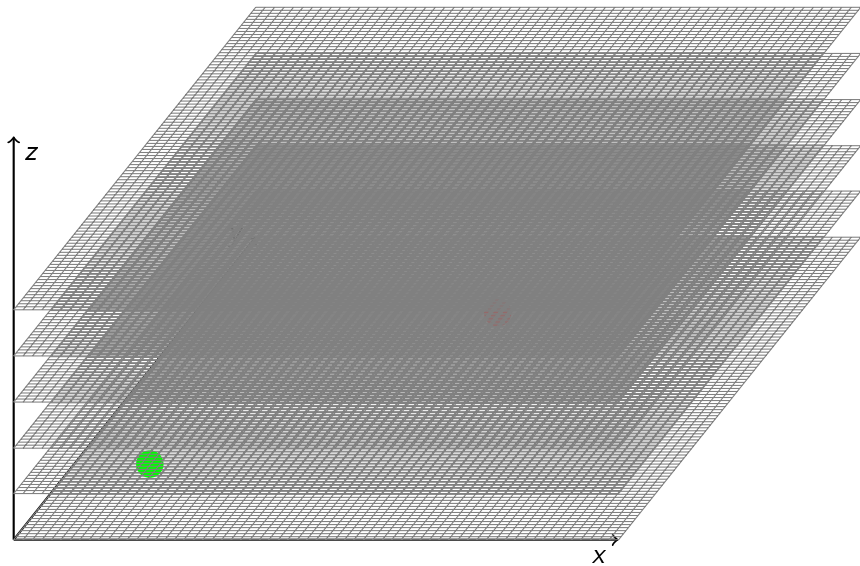


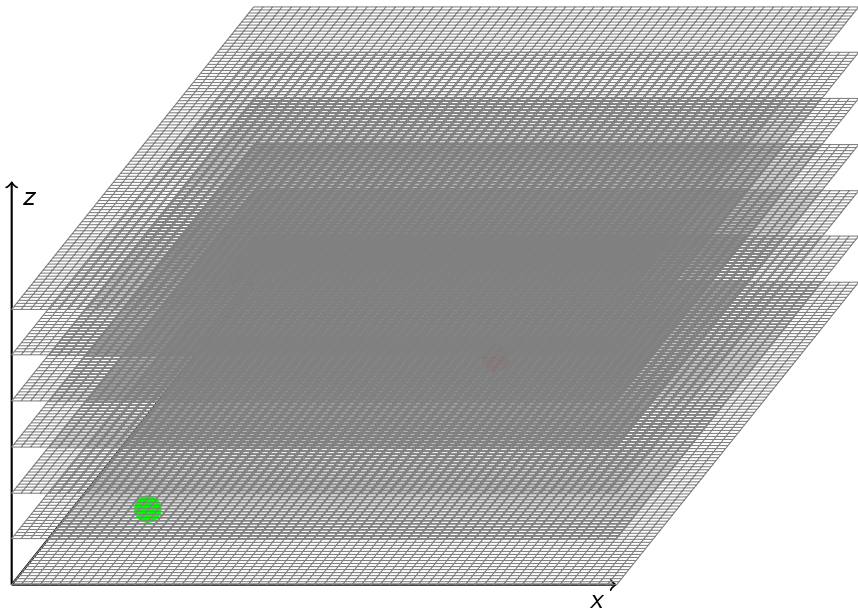


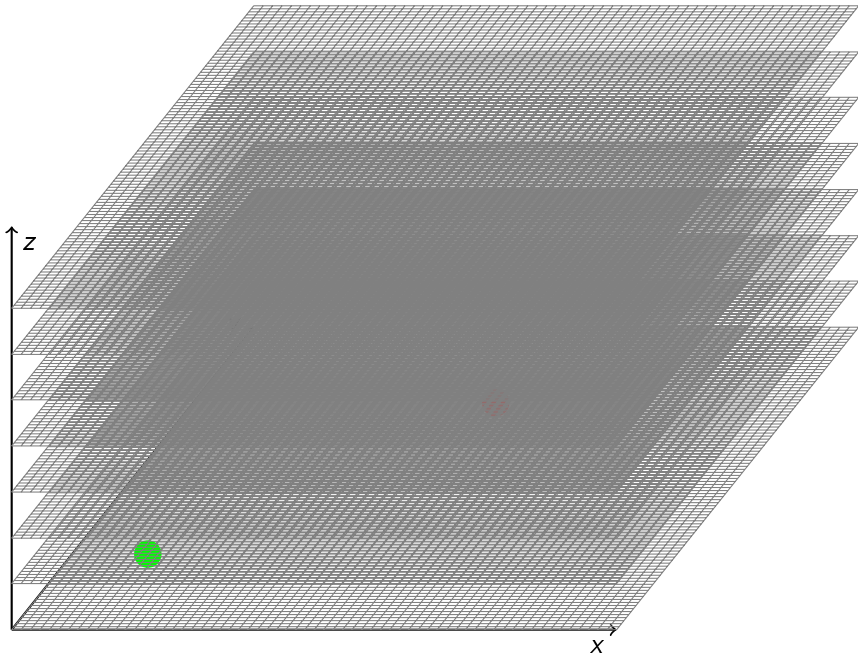












PPM

Cracken

- 1 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 2 Passende Zellen werden geladen
- 3 Passwörter werden aus den Koordinaten gebildet
- 4 Entstandene Wordlist wird gecrackt

PPM

Cracken

- 1 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 2 Passende Zellen werden geladen
- 3 Passwörter werden aus den Koordinaten gebildet
- 4 Entstandene Wordlist wird gecrackt

PPM

Cracken

- 1 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 2 Passende Zellen werden geladen
- 3 Passwörter werden aus den Koordinaten gebildet
- 4 Entstandene Wordlist wird gecrackt

PPM

Cracken

- 1 Hash wird chunkweise gesplittet (z.B. HEA, AX1, 1m6, 66R)
- 2 Passende Zellen werden geladen
- 3 Passwörter werden aus den Koordinaten gebildet
- 4 Entstandene Wordlist wird gecrackt

PPM

Live-Demo

```
> crack jeOM2RPzBwS.Y
```

```
1. 2 characters:      Saturation  
sing length = 3768   41.750693%  
dual length = 1577   17.473684%  
triple length = 651  7.213296%  
quad length = 286    3.168975%
```

```
2. 2 characters:      Saturation  
sing length = 3788   41.972299%  
dual length = 1596   17.684211%  
triple length = 706  7.822715%  
quad length = 314    3.479224%
```

```
Building probability vectors...
```

```
Cracking remaining 89804 possibilities...
```

```
Password : ????
```

Quellen



Jon Erickson.

Hacking: The Art of Exploitation, 2nd Edition.

No Starch Press, 2008.