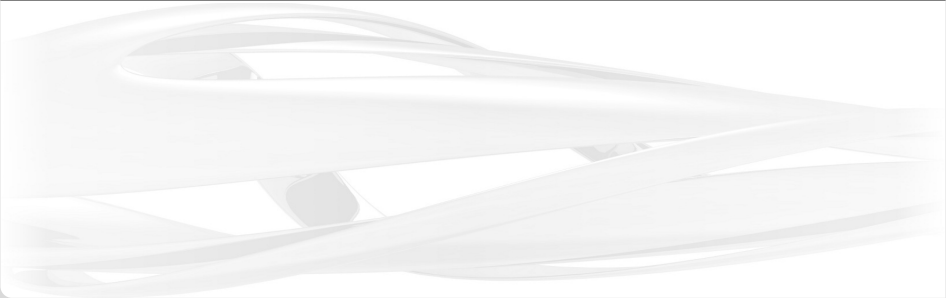


Rainbow Tables

Abschlussproject „GPU Computing” – Niklas Baumstark

Lehrstuhl für Computergrafik



- Viele User verwenden selbes Passwort für verschiedene Dienste
- Szenario: Kompromittierter Server im Internet
 - Angreifer kann Datenbank auslesen
 - Angreifer kann u.U. live eingegebene Passwörter mitschneiden
- Ziel: Angreifer kann inaktive Passwörter nicht lesen
- Lösung: Vor Speicherung Hashfunktion h auf Passwort anwenden

Gegeben

Passwortraum P , Hashraum H , Hashfunktion $h : P \rightarrow H$, Hashwert $y \in H$

Gesucht

Urbild $p \in P$ mit $h(p) = y$

Kein Speicher $\rightarrow \mathcal{O}(|P|)$ Zeit pro Anfrage

Iteriere über P bis wir Urbild finden

$\mathcal{O}(|P|)$ Speicher $\rightarrow \mathcal{O}(1)$ Zeit pro Anfrage

Baue einmalig Datenbank aller Hashwerte auf

- Rainbow Tables bieten Tradeoff zwischen Speicher und Abfragezeit [Oechslin, 2003]
- Benutze Familie von Reduktionsfunktionen $r_i : H \rightarrow P$
- Berechne Ketten von Werten die zwischen H und P alternieren, ausgehend von Startwerten $x_0 \in P$

$$x_0 \xrightarrow{h} h_0 \xrightarrow{r_0} x_1 \xrightarrow{h} h_1 \xrightarrow{r_1} x_2 \xrightarrow{h} \dots \xrightarrow{r_{t-1}} x_t$$

- Wähle verschiedene x_0 , speichere für jede Kette nur x_0 und x_t
- Bei Hashabfrage partielle Ketten aufbauen und Endpunkte in der Datenbank nachschlagen

- Wenn x_i für festes i in verschiedene Ketten übereinstimmt kollidieren sie ab diesem Punkt bis zum Ende \rightarrow *Merge*
- Ziel: Hohe Abdeckung von P , also möglichst wenige Merges
- *Perfekte* Tabelle hat gar keine Merges, alle Endpunkte verschieden



- Aufbauen und Abfragen von perfekten Rainbow Tables mithilfe der GPU, über Alphabet Σ
- Ziele
 - Speichereffizienter Aufbau der Tabelle im Device Memory
 - Entfernen von Duplikaten während des Aufbaus
 - Kompetitiver Durchsatz

- Stelle Strings als Zahlen zur Basis $|\Sigma|$ dar \rightarrow Basisumwandlung
- Ein Work Item pro Kette
- Mikrooptimierungen essentiell, besonders Vermeiden von Branch Divergence
- Duplikatentfernung
 - Führe alle X Iterationen durch
 - Schritt 1: Sortiere Ketten nach Endpunkt (Bitonic Sort)
 - Schritt 2: Markiere Ketten deren Endpunkt gleich dem des Vorgängers ist
 - Schritt 3: Filtere alle markierten Ketten aus mit Prefix Scan [Blelloch and Maggs, 2010]
 - Nebeneffekt: Tabelle am Ende schon sortiert

- Berechne partielle Ketten für alle Queries, nach Länge sortiert
- Sortiere Endpunkte der partiellen Ketten
- Benutze Binärsuche um Startpunkte zu finden
- Rekonstruiere Ketten um Hashwert zu finden

Für $t = 1000$, $|\Sigma| = 36$, $n = 6$

Programm	Hashreduktionen pro Sek.	Abfragen pro Sek.
RainbowCrack	5.0M	16.6
niklas-cpu	4.6M	12.0
niklas-ocl	662M	714.0
CryptoHaze-ocl	735M	(51.0)
CryptoHaze-cuda	1070M	(100.0)

-  Blelloch, G. E. and Maggs, B. M. (2010).
Algorithms and theory of computation handbook.
chapter Parallel Algorithms, pages 25–25. Chapman & Hall/CRC.
-  Oechslin, P. (2003).
Making a Faster Cryptanalytic Time-Memory Trade-Off.
In *Advances in Cryptology - CRYPTO 2003*, pages 617–630.