Konzept: Websocket-Bouncer

Dr. Arne Babenhauserheide

<2021-11-19 Fr>

Ein Freund fragte mich zur Idee einer Datenschutzfreundlichen Umfragesoftware: *„Auf welcher Basis würdest du das Schreiben wollen?“*

Ich würde einen kleinen Server schreiben und wahrscheinlich ein Javascript-Tool, das sich über den Server synchronisiert. Daten sollten verschlüsselt sein wie auf cryptpad und den Server nur als Verteiler nutzen.

Ich habe hier gerade etwas websocket-code rumliegen, vielleicht wäre das eine gute Grundlage: Javascript Frontend, das den Websocket als Datenverteiler nutzt.

Mal als Schnellschuss:

Link auf dem Server: server/websocket-key#crypto-key

Alle Nachrichten auf websockets des gleichen websocket-keys werden sofort verteilt, der eigene Zustand wird in local storage gespeichert.

Die an den Server geschickten Nachrichten sind opaque Daten (einfach verschlüsselte bytes), die der Server nicht einsehen kann. Sie haben zwei Felder: hash der verschlüsselten Daten der vorherigen Nachricht (oder null, falls noch keine gesehen wurde) und die verschlüsselten Daten der aktuellen Nachricht:

Hash der vorherigen Daten Daten

Dadurch kann der Server einen Graphen aller Nachrichten aufbauen und dann neuen Clients alle noch nicht gesehenen Nachrichten schicken.

Operationen in Javascript müssen von der Reihenfolge unabhängig sein.[1]

Daten für ein Umfrageprogramm: Es gibt eine Hashmap: Users. Jeder User hat eine Liste mit den Fragen. Inhalte der Nachrichten sind:

```
{user-id: <id>, answer-id: <id>, increment: <+-N>}
```

Ablauf:

- Im Browser den Link server/websocket-key#crypto-key öffnen

[1] Also CRDTs: https://hal.inria.fr/inria-00555588/document

- Server liefert Webseite mit Javascript-Code aus (damit alles Clientseitig läuft und der Server keine Daten hat). Der crypto-key landet so nie beim Server (weil hinter dem #).
- Der client erzeugt eine user-id oder holt sie aus dem local storage.[2]
- Der client verbindet sich mit dem Websocket und der Server liefert alle bekannten Nachrichten aus.[3, 4]
- Der client aggregiert die Nachrichten, dann kann der Nutzer Einstellungen ändern und sie werden als neue Nachrichten an den Server geschickt.
- Der Server verteilt die Nachrichten.

Ich bin sicher, irgendwer macht schon sowas, aber das hier wäre frei und selbstgemacht, um CRDTs sinnvoll zu nutzen.

Das Thread-Model geht von gutartigen Teilnehmern aus: Wer die URL hat, hat vollen Zugriff.

Das hier skaliert nicht für große Datenmengen, weil bei jedem neuen Zugriff alle Nachrichten geschickt und aggregiert werden müssen. Das ist OK so. Es gäbe Optimierungen, die macchen es aber für eine kleine Umfrage unnötig kompliziert.

[2] Der Client kann auch den Nutzer nach der user-id fragen und nach der Generierung einen Code liefern, über den sich user auf anderen Geräten wieder anmelden können.

[3] Optimierung: Der Client schickt eine sync-Nachricht mit der letzten bekannten nachrichten-ID und erhält nur die Nachrichten, die er noch nicht kennt.

[4] Option: Admin-URL + Schick nur dem Admin alle Nachrichten, bis der Admin die Umfrage beendet, um taktische Abstimmungen zu vermeiden.