Aufgabenblatt 5

Aufgabe – Authentifizierung beim Publish-Subscribe-System

In der Schnittstelle des Publish-Subscribe-Systems aus der vorherigen Aufgabe fehlte die Möglichkeit, Benutzer an- und abzumelden. Eine Authentifizierung soll nun ergänzt werden.

Die Implementierung soll auf Basis eines **Challenge-Response-Protokolls** erfolgen:

- Bei dieser Art der **Authentifizierung** wird die Übertragung eines *Credentials* in Form eines Passworts im Klartext vom Client zum Server vermieden.
- Außerdem wird das Passwort nicht direkt auf dem Server gespeichert. Stattdessen verwaltet der Server **Hash-Werte** *H(user;pwd)*[1] aus den Kombinationen von Nutzername *user* und Passwort *pwd* zu dem Nutzernamen in einer Datei, einem sog. *Digest*.

Auf eine Authentifizierungsanfrage antwortet der Server zunächst mit der Übermittlung eines Zahlenwerts *nonce* (Abkürzung für ‚*Number used once*'). Der anfragende Client antwortet mit einer Nachricht *nonce;user;H(nonce;user;H(user;pwd))*. Der Server kann auf Basis des gespeicherten Hashwertes *H(user;pwd)* den übermittelten Hashwert nun verifizieren ohne – und das ist entscheidende - das Passwort des Nutzers im Klartext zu kennen. Die Annahme dabei ist, dass die Hash-Funktion eine **Einwegfunktion** ist, d.h. der Hash-Wert eindeutig ist und keinerlei Rückschluss auf die ursprüngliche Zeichenkette zulässt. Bei jeder der folgenden Aktionen kann nun auch die Integrität der übermittelten Daten (z.B. RPC-Parameter) server-seitig verifiziert werden. Dazu werden im Client Nachrichten der Form *nonce;data;H(nonce;data;H(user;pwd))* generiert und an den Server übertragen.

Setzen Sie dieses Protokoll nun für das bestehende Publish-Subscribe-System um. Gehen Sie dabei wie folgt vor:

- Erweitern Sie zunächst die **Schnittstellen-Datei** des Servers. Es werden 3 neue Funktionen

```
// nonce vom Server anfragen
rpc get_session (UserName) returns (SessionId) {}
// Validierung der Session
rpc validate (PubSubParam) returns (ReturnCode) {}
// Invalidierung
rpc invalidate (SessionId) returns (ReturnCode) {}
```

  benötigt. Der zurückgegebene nonce-Wert der Funktion **`get_session`** wird als Sitzungsschlüssel vom Typ **`SessionId`** interpretiert. Da bei einem **RPC nur ein Parameter** übertragen werden kann, wird ein generischer Datentyp **`PubSubParam`** eingeführt, der eine Nachricht (ggf. leer), ein Topic oder eine Adresse sein kann und mit dem die bislang vorhandenen Funktionen nun aufgerufen werden. Bsp. (Setzen des Topics):

```
rpc set_topic (PubSubParam) returns (ReturnCode) {}
```

[1] Die zwei Zeichenketten werden durch ein Semikolon zu einer neuen Zeichenkette konkateniert.

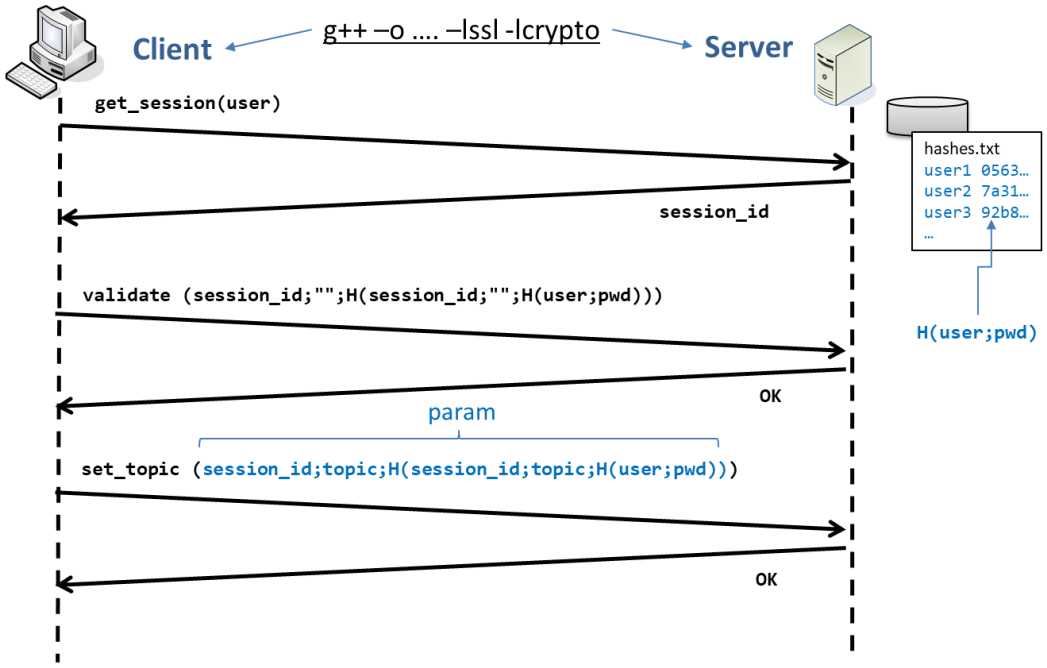
In **PubSubParam** wird das **oneof** Konstrukt (siehe den Language Guide https://developers.google.com/protocol-buffers/docs/proto3#oneof, hier wird insbesondere auf das Speichermanagement in C++ eingegangen) genutzt, das in etwa einer **union** in C/C++entspricht:

```
// Nutzername
message UserName {
    string name = 1;
}
// Sitzungschluessel
message SessionId {
    int32 id = 1;
}
// Generische Nachricht
message PubSubParam {
    oneof param {
        /* Achtung: Camel-Case Schreibweise wird bei C++ ignoriert */
        Message optMessage = 1;
        Topic optTopic = 2;
        SubscriberAddress optAddress = 3;
        EmptyMessage void = 4;
    }
    SessionId sid = 5;
    string hash_string = 6;
}
```

- Im Zuge der Abarbeitung des Protokolls können zusätzliche Fehler entstehen, die Berücksichtigung finden müssen.

```
// Rueckgaben vom Service
message ReturnCode {
    enum Values {
        OK = 0;
        CANNOT_REGISTER = 1;
        CLIENT_ALREADY_REGISTERED = 2;
        CANNOT_UNREGISTER = 3;
        CANNOT_SET_TOPIC = 4;
        NO_HASH_FOR_SESSION = 5;
        WRONG_HASH_FOR_SESSION = 6;
        USER_ALREADY_LOGGED_IN = 7;
        SESSION_INVALID = 8;
        UNKNOWN_ERROR = 9;
        /* Hier koennen bei Bedarf weitere Return-Codes ergaenzt werden */
    }
    Values value = 1;
}
```

- Der Ablauf der Authentifizierung mit Hilfe dieser Funktionen wird in dem Sequenzdiagramm unten dargestellt: bei einem Login werden vom Client aus nacheinander die Funktionen **get_session()** und **validate()** aufgerufen, bei einem Logout die Funktion **invalidate()**.



Hinweise:

- Für die Generierung des Sitzungsschlüssels kann die System-Funktion **`clock()`** verwendet werden, welche die aktuelle CPU-Zeit als eindeutigen Wert liefert.

- Als Hashfunktion *H* soll **SHA-256** verwendet werden. Im Dateibereich zum Praktikum finden Sie einen C Source-Code, welcher die Nutzung der Bibliothek und der SHA-Funktion verdeutlicht. Dort wird auch der Umgang mit dem Hash-Digest gezeigt. Dieses wird in einer Datei **`hashes.txt`** erwartet. Die Einträge dort haben das Format:

  `student db398679a8775d55b61228cbeb8dff61a86dbe5d3b68557e5a4c36a927c17e5f`
  …

  Die Datei sollte beim Server-Start eingelesen und in einer Datenstruktur (z.B. **`std::map<std::string, std::string>`**) gespeichert werden.

- Des Weiteren wird im Server eine Datenstruktur (ebenfalls **`map`** s.o.) zur Zuordnung der Sessions zu Nutzern benötigt, so dass nach geglückter Authentifizierung nur die Session-ID bei einem RPC übermittelt werden muss.

- Es wird empfohlen, die als Hashwert übermittelten Credentials zu Anfang im aufgerufenen RPC (also **`publish()`**, **`set_topic()`**, …) durch eine zentrale Funktion **`check_session()`** im Server zu überprüfen und ggf. eine Fehlermeldung zurück zu geben.

- Der Receiver muss nicht verändert werden.

Für das Testat ist ein Protokoll bekannter Formatierung (siehe Blatt 1) inkl. der durchgeführten Tests vorzulegen.

Testierung: 17./18.5.2021