Improving Authentication for Real-time Communication (WebRTC)

Linesh Malkam (21CSB0A35)
Ritik Raj Yadav (21CSB0A48)

1. Introduction

WebRTC stands for Web Real-Time Communication. It's an open-source project that enables peer-to-peer communication directly in the web browser without the need for any plugins or additional software. WebRTC provides a set of APIs (Application Programming Interfaces) that allow web developers to create real-time communication applications, such as video conferencing, voice calling, and file sharing.

Problem : When using the DTLS-SRTP security method recommended by WebRTC, a situation occurs where users are unable to authenticate their remote peers. This circumstance causes many security problems, especially the man-in-the middle (MiTM) attacks.

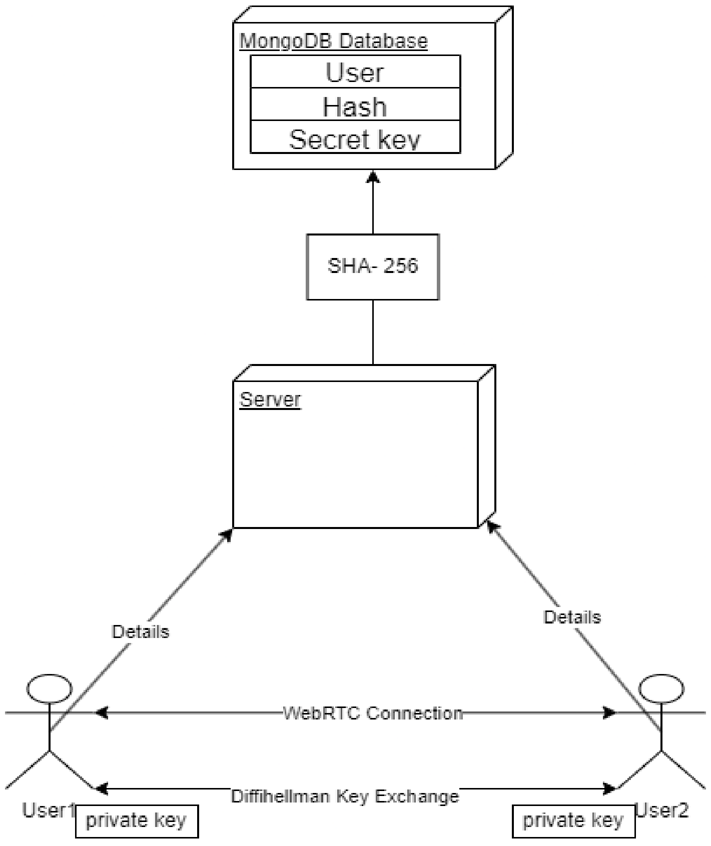
Existing Solution : WebRTC itself doesn't provide built-in authentication or identity management features, which means developers need to implement their own authentication mechanisms to secure their WebRTC applications properly. Relying solely on third-party identity providers like Google or Facebook for authentication can indeed be limiting and pose security risks.

Solution : Implementing an in-house authentication system with secure hashing and key exchange algorithms provides a strong foundation for ensuring the security and integrity of your WebRTC application. Moreover, focusing on providing a seamless user experience enhances user satisfaction and adoption of your real-time communication services.

2. Objectives

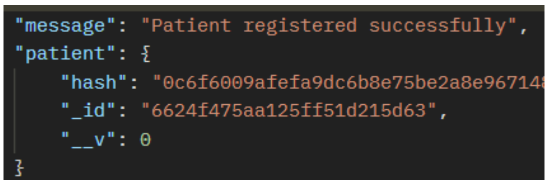
- Implementing a secure authentication system.
- Creating an automation flow from entering details to landing on real-time communication services

3. **Implementation and Results analysis**



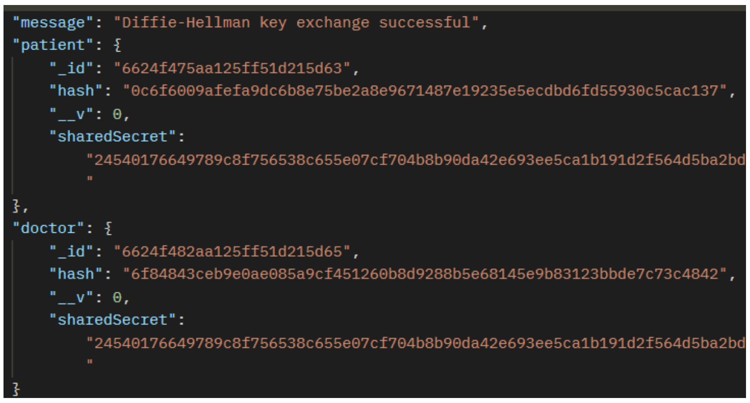
Work - Flow

1. Users' details (patients and doctors) are registered and hashed and stored in the database. (For details we will just take a unique username from user)



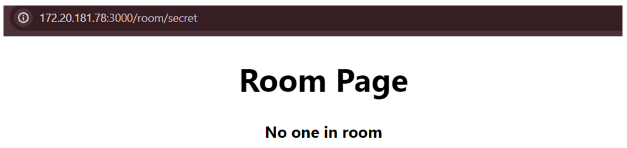
```
"message": "Patient registered successfully",
"patient": {
    "hash": "0c6f6009afefa9dc6b8e75be2a8e96714
    "_id": "6624f475aa125ff51d215d63",
    "__v": 0
}
```

2. We will perform a Diffihellman key exchange between doctor and a patient



```
"message": "Diffie-Hellman key exchange successful",
"patient": {
    "_id": "6624f475aa125ff51d215d63",
    "hash": "0c6f6009afefa9dc6b8e75be2a8e9671487e19235e5ecdbd6fd55930c5cac137",
    "__v": 0,
    "sharedSecret":
        "24540176649789c8f756538c655e07cf704b8b90da42e693ee5ca1b191d2f564d5ba2bd
        "
},
"doctor": {
    "_id": "6624f482aa125ff51d215d65",
    "hash": "6f84843ceb9e0ae085a9cf451260b8d9288b5e68145e9b83123bbde7c73c4842",
    "__v": 0,
    "sharedSecret":
        "24540176649789c8f756538c655e07cf704b8b90da42e693ee5ca1b191d2f564d5ba2bd
        "
}
```

3. Users log in with their username to access a unique room assigned based on their username. Secure keys are exchanged between peers to ensure only users with the correct keys can join the room. Private keys are acting as room numbers for communication sessions.

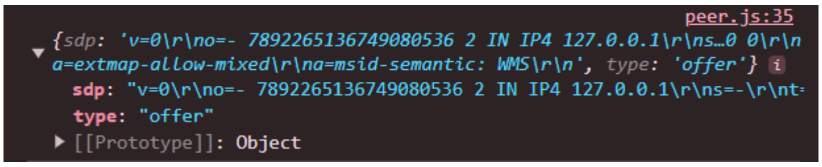


172.20.181.78:3000/room/secret

Room Page

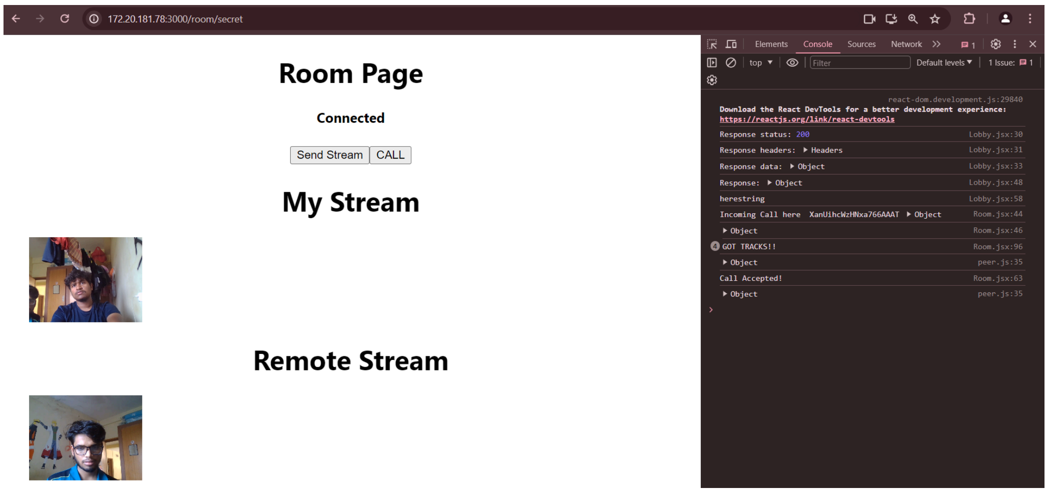
No one in room

4. Now the peers have joined their allotted room for communication. Offer is being between peers with the help of SDP packet.



```
                                                             peer.js:35
▼ {sdp: 'v=0\r\no=- 7892265136749080536 2 IN IP4 127.0.0.1\r\ns…0 0\r\n
  a=extmap-allow-mixed\r\na=msid-semantic: WMS\r\n', type: 'offer'} i
    sdp: "v=0\r\no=- 7892265136749080536 2 IN IP4 127.0.0.1\r\ns=-\r\nt=
    type: "offer"
  ▶ [[Prototype]]: Object
```

5. Once the offers are exchanged and accepted now users can communicate in real time.



172.20.181.78:3000/room/secret

Room Page

Connected

My Stream

Remote Stream

```
                                   react-dom.development.js:29840
Download the React DevTools for a better development experience:
https://reactjs.org/link/react-devtools
Response status: 200                                 Lobby.jsx:30
Response headers: ▶ Headers                          Lobby.jsx:31
Response data: ▶ Object                              Lobby.jsx:33
Response: ▶ Object                                   Lobby.jsx:48
herestring                                           Lobby.jsx:58
Incoming Call here  XanUihcWzHNxa766AAAT ▶ Object    Room.jsx:44
 ▶ Object                                            Room.jsx:46
4 GOT TRACKS!!                                       Room.jsx:96
 ▶ Object                                              peer.js:35
Call Accepted!                                       Room.jsx:63
 ▶ Object                                              peer.js:35
>
```

4. Conclusion

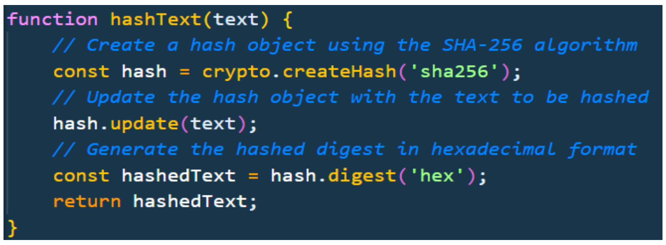
In standard WebRTC, while secure channels are established between peers, mutual authentication isn't supported, leaving them vulnerable to man-in-the-middle attacks. Proposed solutions like using an external identity provider for authentication, such as Facebook Connect, introduce trust dependencies on these third-party services, which may not always be ideal.

The proposed in-house authentication system uses SHA for hashing and a secure key exchange mechanism. Additionally, interactive parameters allow real-time changes to user communication authority. Unlike existing literature that lacks similar solutions, this approach offers flexibility for future studies.

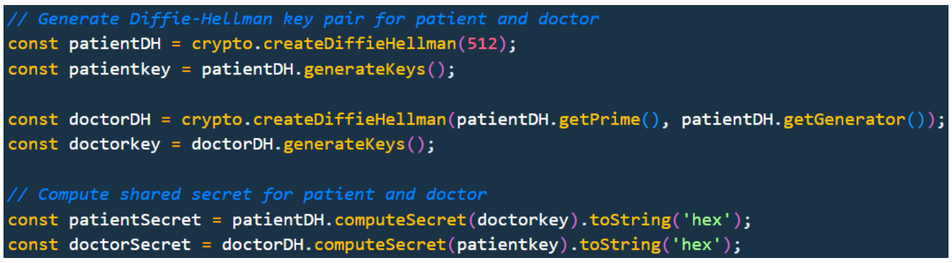
5. Learning outcomes

- Implemented WebRTC Protocol where DTLS-SRTP is used.
- Learned about how SDP (Session Description Packet) are being exchanged between peers.
- Improved understanding about Hashing and Key exchange Algorithms.
- Implemented an Express server for API calls, Socket server for client side and server side communication and MongoDB for storing the data of users Securely.
- Mastered seamless management and flow of data using React Framework from the Client to server and vice versa.

6. Source code: https://github.com/lmalkam/Improved-Authentication-For-WebRTC.git



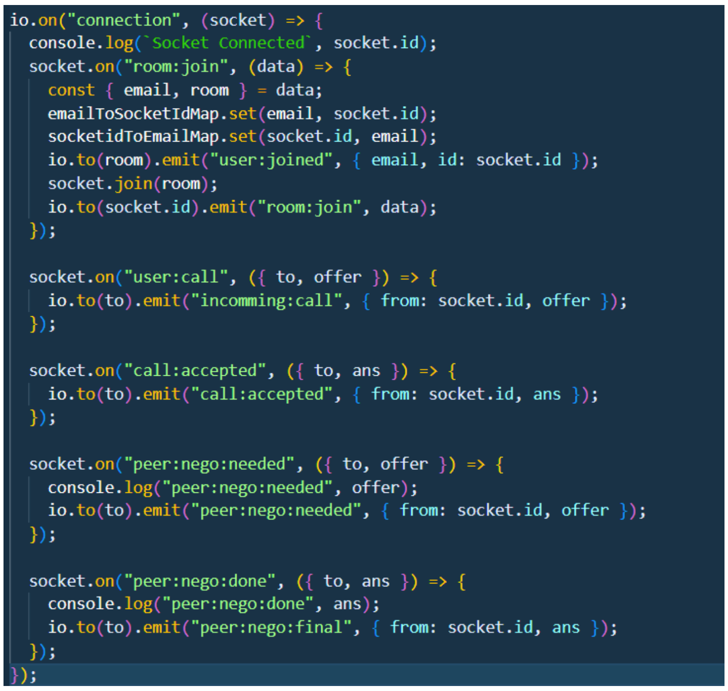
```
function hashText(text) {
    // Create a hash object using the SHA-256 algorithm
    const hash = crypto.createHash('sha256');
    // Update the hash object with the text to be hashed
    hash.update(text);
    // Generate the hashed digest in hexadecimal format
    const hashedText = hash.digest('hex');
    return hashedText;
}
```



```
// Generate Diffie-Hellman key pair for patient and doctor
const patientDH = crypto.createDiffieHellman(512);
const patientkey = patientDH.generateKeys();

const doctorDH = crypto.createDiffieHellman(patientDH.getPrime(), patientDH.getGenerator());
const doctorkey = doctorDH.generateKeys();

// Compute shared secret for patient and doctor
const patientSecret = patientDH.computeSecret(doctorkey).toString('hex');
const doctorSecret = doctorDH.computeSecret(patientkey).toString('hex');
```



```
io.on("connection", (socket) => {
  console.log(`Socket Connected`, socket.id);
  socket.on("room:join", (data) => {
    const { email, room } = data;
    emailToSocketIdMap.set(email, socket.id);
    socketidToEmailMap.set(socket.id, email);
    io.to(room).emit("user:joined", { email, id: socket.id });
    socket.join(room);
    io.to(socket.id).emit("room:join", data);
  });

  socket.on("user:call", ({ to, offer }) => {
    io.to(to).emit("incomming:call", { from: socket.id, offer });
  });

  socket.on("call:accepted", ({ to, ans }) => {
    io.to(to).emit("call:accepted", { from: socket.id, ans });
  });

  socket.on("peer:nego:needed", ({ to, offer }) => {
    console.log("peer:nego:needed", offer);
    io.to(to).emit("peer:nego:needed", { from: socket.id, offer });
  });

  socket.on("peer:nego:done", ({ to, ans }) => {
    console.log("peer:nego:done", ans);
    io.to(to).emit("peer:nego:final", { from: socket.id, ans });
  });
});
```

References

1. WebRTC Documentation : https://webrtc.org/getting-started/overview
2. Improving WebRTC using Blockchain (Research Paper) : https://ieeexplore.ieee.org/abstract/document/9297333
3. Usable authentication systems for real time web-based audio/video communications : https://ieeexplore.ieee.org/document/7899400
4. WebRTC: Your Privacy is at Risk : https://dl.acm.org/doi/pdf/10.1145/3019612.3019844
5. NodeJS Crypto Library : https://nodejs.org/api/crypto.html
6. DTLS-SRTP documentation : https://datatracker.ietf.org/doc/html/rfc5764