SUPERVISOR: AHMED HANIF

NARWAL AUTH API DELIVERABLE 7

MUHAMMAD NASEEM
SAMI NAEEM
KYNAT MANSHA

Zero Knowledge Authentication JS library

Dependencies

1. **Node.js**:
   - **Purpose**: Required for running the JavaScript code on the server side.
   - **Explanation**: Node.js provides the runtime environment that allows JavaScript to be executed outside of a web browser, enabling server-side scripting and building scalable network applications.
2. **node-fetch**:
   - **Purpose**: A module to make HTTP requests, similar to `fetch` in the browser.
   - **Explanation**: node-fetch is a lightweight module that brings the `window.fetch` function to Node.js. It allows you to make network requests to retrieve data from APIs, send data to servers, and interact with web resources.
3. **crypto**:
   - **Purpose**: A module for cryptographic functions.
   - **Explanation**: The `crypto` module provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign, and verify functions. It's used for secure data encryption, hashing, and various other cryptographic operations.
4. **express**:
   - **Purpose**: A web framework for Node.js to build web applications and APIs.
   - **Explanation**: `express` is a fast, unopinionated, and minimalist web framework for Node.js. It simplifies the process of setting up a web server and handling HTTP requests and responses, making it easier to build robust and scalable web applications and APIs.
5. **body-parser**:
   - **Purpose**: Middleware to parse incoming request bodies in a middleware before your handlers.
   - **Explanation**: `body-parser` is a middleware for Express that parses incoming request bodies in a middleware before your handlers, making the parsed data available under the `req.body` property. It supports various formats, including JSON and URL-encoded data, enabling easy access to incoming data in HTTP requests.

.

While the code encapsulates essential functionalities for authentication, cryptography, and server communication, it's important to note that it is not yet finalized. As of now, we are in the process of integrating it into a functioning environment in order to conduct rigorous testing.

We have yet to:

1. **Integrate our code into an Environment**: Ensuring seamless integration into the existing system where authentication and server communication are critical components.

2. **Test and Validate**: Running comprehensive tests to validate the robustness and security of the code. This includes unit tests to verify individual functions as well as integration tests to ensure smooth interaction with other components.

3. **Enhance to meet the best of Security Standards**: Further enhancing security measures to safeguard sensitive user data during transmission and storage. This may involve additional cryptographic protocols or refinements based on test results and security best practices.

4. **Measure its safety**: Implementing safety measures to handle edge cases and unexpected inputs gracefully, ensuring the reliability and stability of the authentication system under various scenarios.

As we progress through these stages, the aim is to refine the codebase, address any potential vulnerabilities, and optimize performance to meet stringent security standards. By prioritizing thorough testing and continuous improvement, we are committed to delivering a robust and secure authentication solution for seamless server-client interactions.

Our JavaScript code module encapsulates essential functionalities for zero knowledge authentication, cryptography, and server communication. Here's a detailed breakdown of its components and operations:

- **Module Imports:** The code begins by importing necessary modules like node-fetch for HTTP requests, crypto for cryptographic operations, express for setting up a server, and body-parser for parsing request bodies.

- **NarwalAuthFunctions Object:** This object houses several methods crucial for authentication and server interactions.

```
const NarwalAuthFunctions = {
```

- **generatePublicKey(password, username):** This method generates a public key based on the provided password and username. It uses hashing via the hashFunction method and implements the Diffie-Hellman key exchange algorithm to compute the public key (Y).

```
async generatePublicKey(password, username) {
    const x = await this.hashFunction(password, username);
    const g = BigInt(2);
    const primenumber =

"40740719526689721725368913768187563221029367873318725012722808987087625995
26673412366794779";
    const modulus = BigInt(primenumber);
    const Y = this.Exponent(g, x, modulus);
    return Y;
  }
```

- **hashFunction(password, username):** Responsible for hashing the concatenated password and username using SHA-256, ensuring secure storage and transmission of sensitive data.

```
async hashFunction(password, username) {
    const pepper = password + username;
    const msgUint8 = new TextEncoder().encode(pepper);
    const hashBuffer = await crypto.subtle.digest("SHA-256", msgUint8);
    const hashArray = Array.from(new Uint8Array(hashBuffer));
    const hashHex = hashArray
      .map((b) => b.toString(16).padStart(2, "0"))
      .join("");
    return BigInt("0x" + hashHex);
  }
```

- **Exponent(g, x, mod):** This utility calculates modular exponentiation (g^x) % mod, crucial in cryptographic operations like the Diffie-Hellman key exchange.

```
async hashFunction(password, username) {
    const pepper = password + username;
    const msgUint8 = new TextEncoder().encode(pepper);
    const hashBuffer = await crypto.subtle.digest("SHA-256", msgUint8);
    const hashArray = Array.from(new Uint8Array(hashBuffer));
    const hashHex = hashArray
      .map((b) => b.toString(16).padStart(2, "0"))
      .join("");
    return BigInt("0x" + hashHex);
  }
```

- **solveChallenge(username, password, challenge):** Handles cryptographic challenges from the server. It computes intermediate values and derives cryptographic solutions (c and z) based on inputs and challenge parameters.

```
async hashFunction(password, username) {
    const pepper = password + username;
    const msgUint8 = new TextEncoder().encode(pepper);
    const hashBuffer = await crypto.subtle.digest("SHA-256", msgUint8);
    const hashArray = Array.from(new Uint8Array(hashBuffer));
    const hashHex = hashArray
      .map((b) => b.toString(16).padStart(2, "0"))
      .join("");
    return BigInt("0x" + hashHex);
  }

    const z = r - BigInt(`0x${c}`) * x;

    const solution = {
      username: username,
      challenge: challenge,
      c: c,
      z: z,
      isValid: true,
    };
    return solution;
  }
```

- **HTTP Communication Methods: (naming convention for the api)**
    1. **receiveResultFromServer()**: Illustrates fetching results from the server via a GET request and logs them.

```
const url = "https://example.com/api/getresult";
```

2. **receiveChallengeFromServer()**: Demonstrates receiving challenges from the server via a GET request and logging them.

```
const url = "http://localhost:3000/get-challenge"
```

3. **sendSolutionToServer(username, solution)**: Shows sending solutions to the server via a POST request and logging the server's response.

```
const url = "https://example.com/api/sendSolution"
```

4. **sendUpToServer(username, password)**: Sends user data (username and password) to the server via a POST request and logs the server's response.

```
const url = "https://example.com/api/senduserdata"
```

This code module is pivotal for secure authentication processes, leveraging cryptographic principles and HTTP communication to handle user data securely. It's designed to support robust authentication workflows, including challenge-response mechanisms typical in secure server interactions. By encapsulating these functionalities within NarwalAuthFunctions, the code fosters reusable and structured approaches to authentication and cryptographic operations, ensuring data integrity and security across server-client interactions.

Contributions: Equal(everyone contributed equally to today's tasks)

M Naseem (Documentation and contribution in code)
Sami Naeem (Documentation and contribution in code)
Kynat Mansha (Documentation and contribution in code)