Security Review Request

Nov 2019

Introduction

Userbase is a database-like product, purpose-built for web app user data. Unlike regular databases, all queries run on the client-side (the web browser), and all user data is end-to-end encrypted with keys that never get sent to the server. Users own their own databases, which are stored encrypted on the server-side, and completely partitioned from databases of other users.

Risk Statement

Userbase is intended to prevent a web app owner from being able to see certain user data generated by the users of the app. This feature is positioned as a value-add to the web app owner, rather than to the end user. Userbase is meant to address the use case where the web app owner intentionally does not want to have access to user data, either because the data is a liability (PII, etc.), or because it's private (internal company messaging, private financial data, etc.)

Userbase cannot guarantee to end users that the web app is end-to-end encrypting all user data, or that the web app is not sending some user data elsewhere (either accidentally, such as when the web app uses analytics software, or maliciously). Therefore, we are not claiming that users should automatically expect a higher level of privacy from a web app just because the app is using Userbase. We are only claiming that if a web app owner wants to avoid dealing with user data in clear, then Userbase gives the web app owner the tools to do so. It is still the full responsibility of the web app owner to earn trust of its users about any privacy claims that the web app is making.

Protected Data Scope

Userbase is only claiming to protect the content of user data stored by a web app using the Userbase Database API. Anything else is outside the scope of protection, including data access frequency, who accessed the data, from where the data got accessed, the number of items stored, the size of the items stored, and the number of users. Anyone with access to the Userbase server will be able to reveal the things that are outside the protected data scope.

Trust Model

Our only claim is that anyone with access to the encrypted data in Userbase or with access to the servers that serve the Userbase API, is not able to determine the clear user data.

We are not claiming that a malicious web app owner is not able to fool users into believing that their data is end-to-end encrypted, when in reality it is not.

We are not claiming that a web app using Userbase automatically guarantees any form of user data privacy.

We are not claiming that a malicious web app owner is not able to steal users' keys or users' data by modifying the source code of the web app.

Therefore, this system trusts that:

- The web app owner is well intentioned and uses Userbase correctly.
- The server serving the web app's source code is not compromised.
- The user protects its credentials and encryption keys.

And this system does not trust:

- The Userbase server. It should not be possible to reveal user data or user keys in case the server that serves the Userbase API gets fully compromised, assuming that the server that serves the web app's source code did not get compromised as well.
- Anyone with access to the data stored in Userbase.

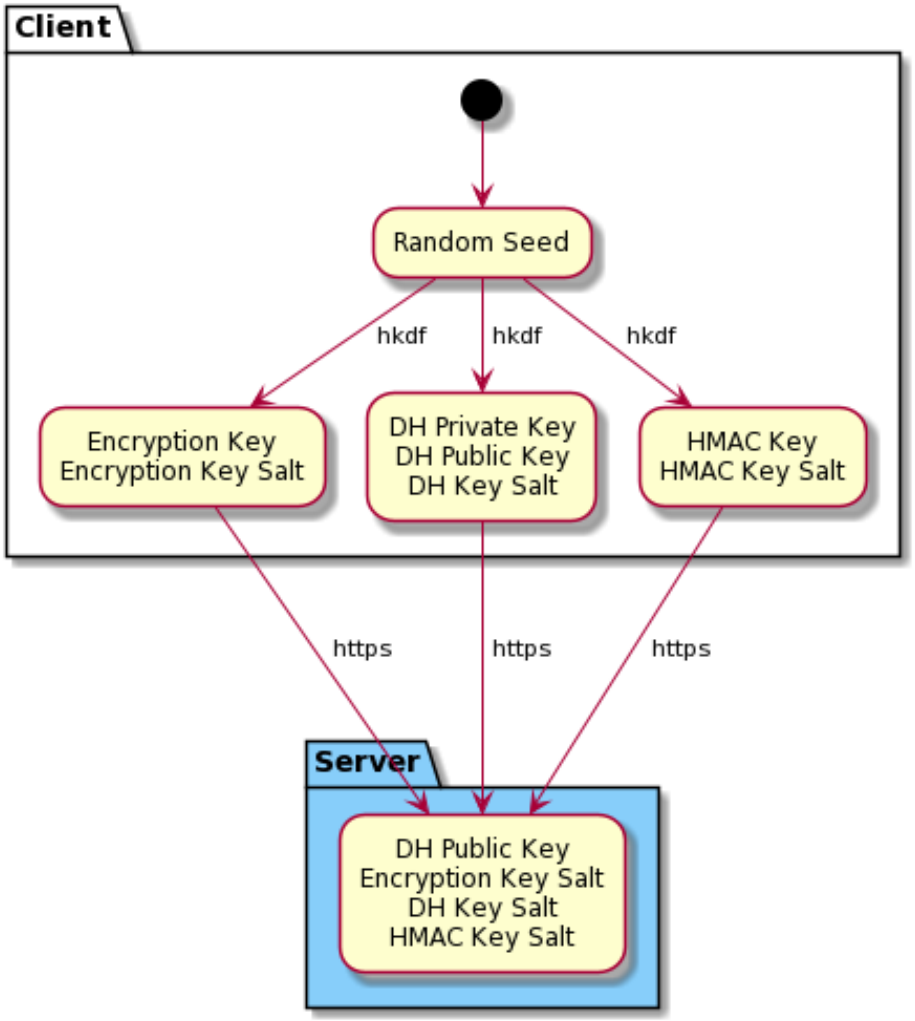
Links

- **Demo:** https://userbase-sec-review.netlify.com (code)
- **Project branch:** security-review

1. Seed Generation (HKDF)

When a user creates a new account, the client generates a random seed and stores it in the browser's local storage. This seed is the input to HKDF to derive three distinct keys for encryption (encryptionKey), Diffie-Hellman (dhPrivateKey), and authenticated hashing (hmacKey). This seed is also displayed to the user in plaintext so that the user can copy it and save it somewhere safe, such as in a password manager. The server receives and stores the salts used to generate each distinct key, as well as the public key associated with dhPrivateKey (dhPublicKey).

· Client-side entry point.
· Server-side entry point.



Seed Generation on Account Creation

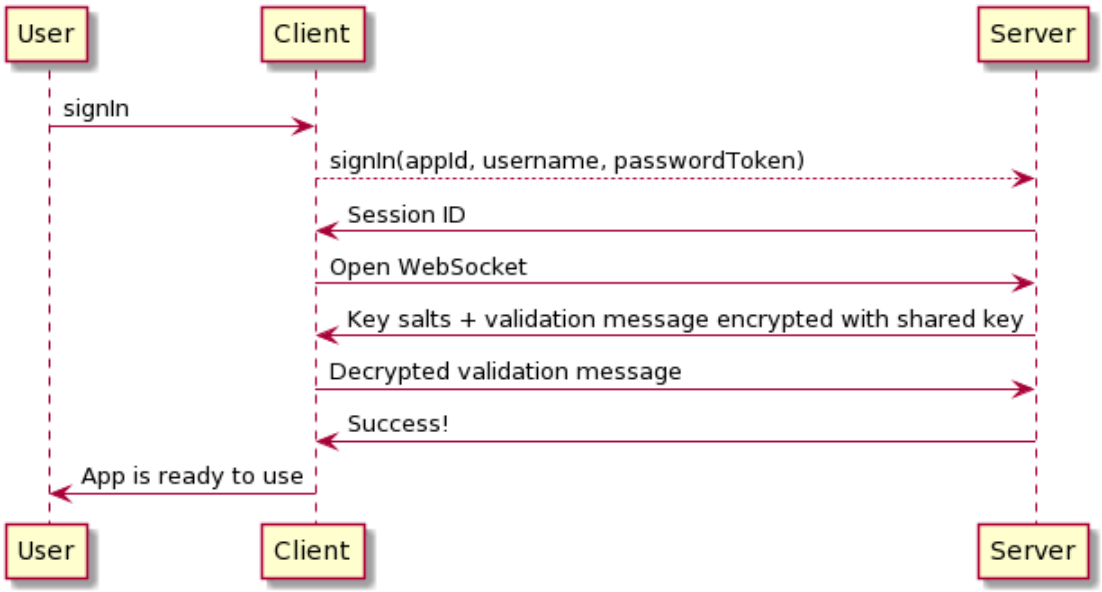
Review scope: Review client-side and server-side code to ensure someone with access to the server cannot determine either the seed or any one of the 3 distinct keys.

2. User Session Management (Diffie-Hellman & AES-GCM-256)

When a user creates a new account or logs into an existing account, the server generates a random session ID and passes it to the client. The client will then attempt to establish a WebSocket connection to the server. If the server determines that the session ID is valid, the WebSocket connection will establish. The WebSocket connection is persistent and long-lived, and all messages passed through it are in an authenticated context.

· Client-side entry points for new account, login, & automatic login using last session.
· Server-side entry point.

Before the server allows the client to do anything with a database (e.g. insert an item), the client must prove access to the user's seed by proving to the server it knows dhPrivateKey. On WebSocket connect, the server generates a random validation message, then generates a shared DH key using a secret private key known only to the server + the user's dhPublicKey, encrypts the validation message, then passes the encrypted validation message to the client. The client then generates the shared DH key using the user's dhPrivateKey + the server's dhPublicKey, decrypts the validation message, then sends it back to the server. If the validation message sent by the client matches the one generated by the server, the session is fully authenticated.



Basic Authentication Flow

Step 1 - Establish WebSocket connection:

· Client-side entry point.
· Server-side entry point and the handling of a new WebSocket connection.

Step 2 - Receive encrypted validation message + salts, then validate key:

· Client-side entry point.
· Server-side entry point.

Review scope: Review client-side and server-side code to ensure stateful session management is implemented securely, and that someone with access to the server cannot determine dhPrivateKey.

---

3. Encryption/Decryption (AES-GCM-256)

When a new database gets created for a user, a random AES key is generated client-side for that database (dbKey). This dbKey is then encrypted with the user's encryptionKey, and the server receives the encrypted dbKey. When the client fetches the database, the server also returns the encrypted dbKey. The client can then decrypt the dbKey using the user's encryptionKey. All transactions written to the database are then encrypted using the dbKey.

Step 1 - Open Database

· Client-side entry point.
· Server-side entry point.

Step 2 - Insert/Update/Delete transactions to the database:

· Client-side entry point.
· Server-side entry point.

Review scope: Review client-side and server-side code to ensure someone with access to the server cannot uncover plaintext user data in the database transactions.

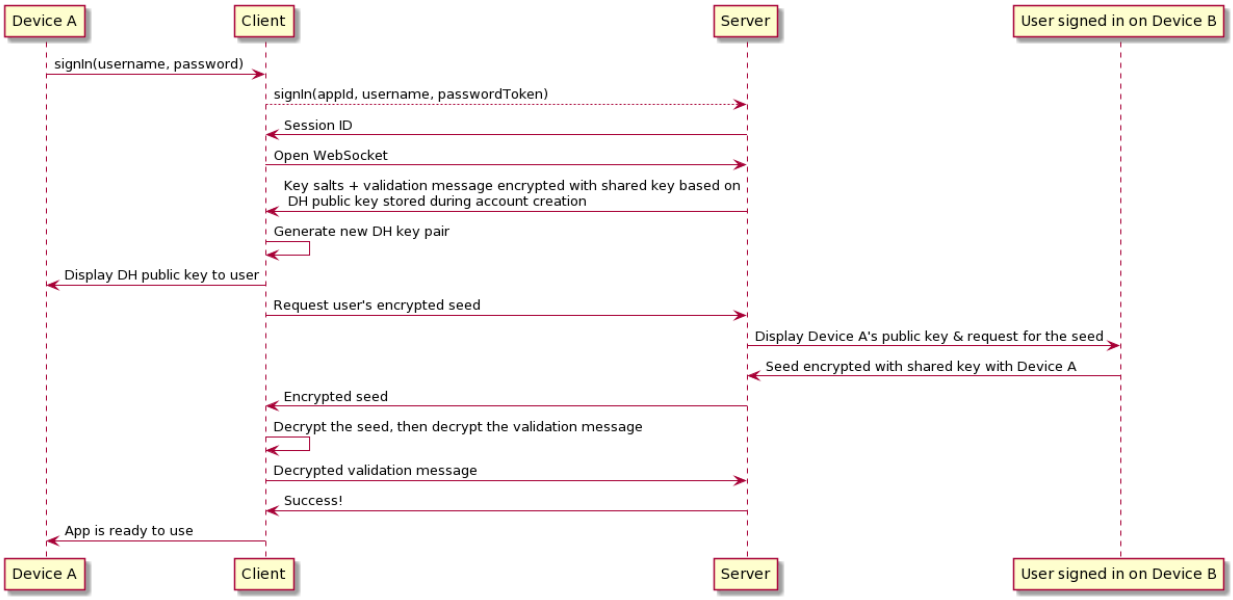
---

4. Seed Exchange Across Devices (Diffie-Hellman, SHA-256, & AES-GCM-256)

When a user logs in from Client A, and Client A does not have the seed saved locally (for example, when logging in from a new device) or backed up by the user's password, Client A will automatically request the seed from any other client that has it.

To exchange the seed, Client A generates a temporary DH public/private key pair. The server only receives Client A's public key for this request. When the same user signs in from Client B that has the seed, the user receives a prompt to send the seed to Client A displaying a SHA-256 hash of Client A's public key in the browser. If Client B accepts, Client B derives a DH shared key using the user's dhPrivateKey + Client A's public key, encrypts the seed using the DH shared key, and sends the encrypted seed to the server. Client A will then automatically receive the encrypted seed and the user's dhPublicKey, derive the DH shared key, decrypt the seed, and store it locally.

The requests are deleted from the server automatically upon confirmed receipt of the seed. If never confirmed, they expire after 24 hours.



Seed Exchange Across Devices

Step 1 - Requesting the seed while signed into Client A:

· Client-side entry point.
· Server-side entry point.

Step 2 - Receiving seed requests & sending encrypted seeds while signed into Client B:

· Client-side entry point.
· Server-side entry points.

Step 3 - Receiving the encrypted seed while signed into Client A:

· Client-side entry point.
· Server-side entry point.

Review scope: Review client-side and server-side code to ensure someone with access to the server cannot determine the seed or the user's dhPrivateKey.

---

5. Database Sharing Across Users (Diffie-Hellman, SHA-256, & AES-GCM-256)

User A can grant access to a database it owns to User B. In order to actually send the grant, User A receives a prompt in the browser displaying a SHA-256 hash of User B's dhPublicKey asking to confirm. Once confirmed, Client A derives a DH shared key using User A's dhPrivateKey + User B's dhPublicKey, encrypts the dbKey using this DH shared key, then sends the encrypted dbKey to the server. User B receives a prompt to accept the grant from User A that displays a SHA-256 hash of User A's dhPublicKey. Once confirmed, User B's client derives the DH shared key using User B's dhPrivateKey + User A's dhPublicKey, then decrypts the dbKey.

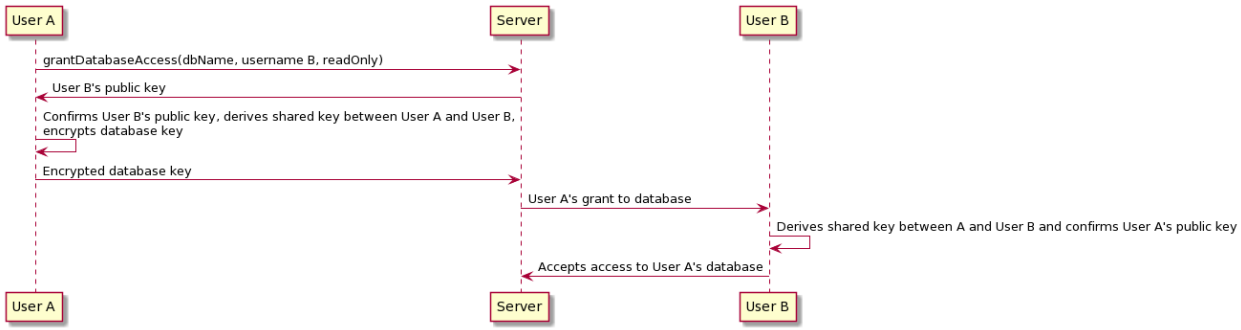
Grants are deleted from the server automatically upon acceptance. If never accepted, they expire after 24 hours.

Step 1 - Granting access from User A's client to User B:

· Client-side entry point.
· Server-side entry point.

Step 2 - Accepting a grant from User B's client:

· Client-side entry point.
· Server-side entry points.



Database sharing

Review scope: Review client-side and server-side code to ensure someone with access to the server cannot determine either the dbKey, User A's dhPrivateKey, or User B's dhPrivateKey.

---

6. Primary Key Uniqueness (HMAC)

In order for the server to determine uniqueness of primary keys in a database, the client sends a keyed-hash of the primary key to the server using HMAC(hmacKey, primary key).

· Client-side example.

Review scope: Review client-side code and this approach to ensure that someone with access to the server cannot determine either hmacKey or the plaintext primary key.

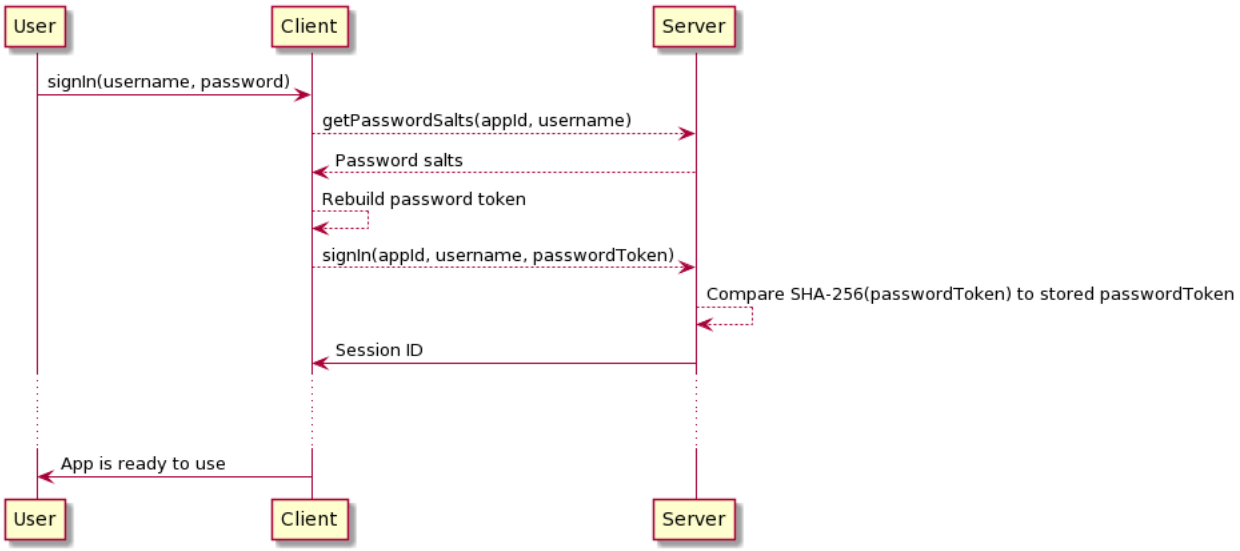
---

7. Client-side password hashing

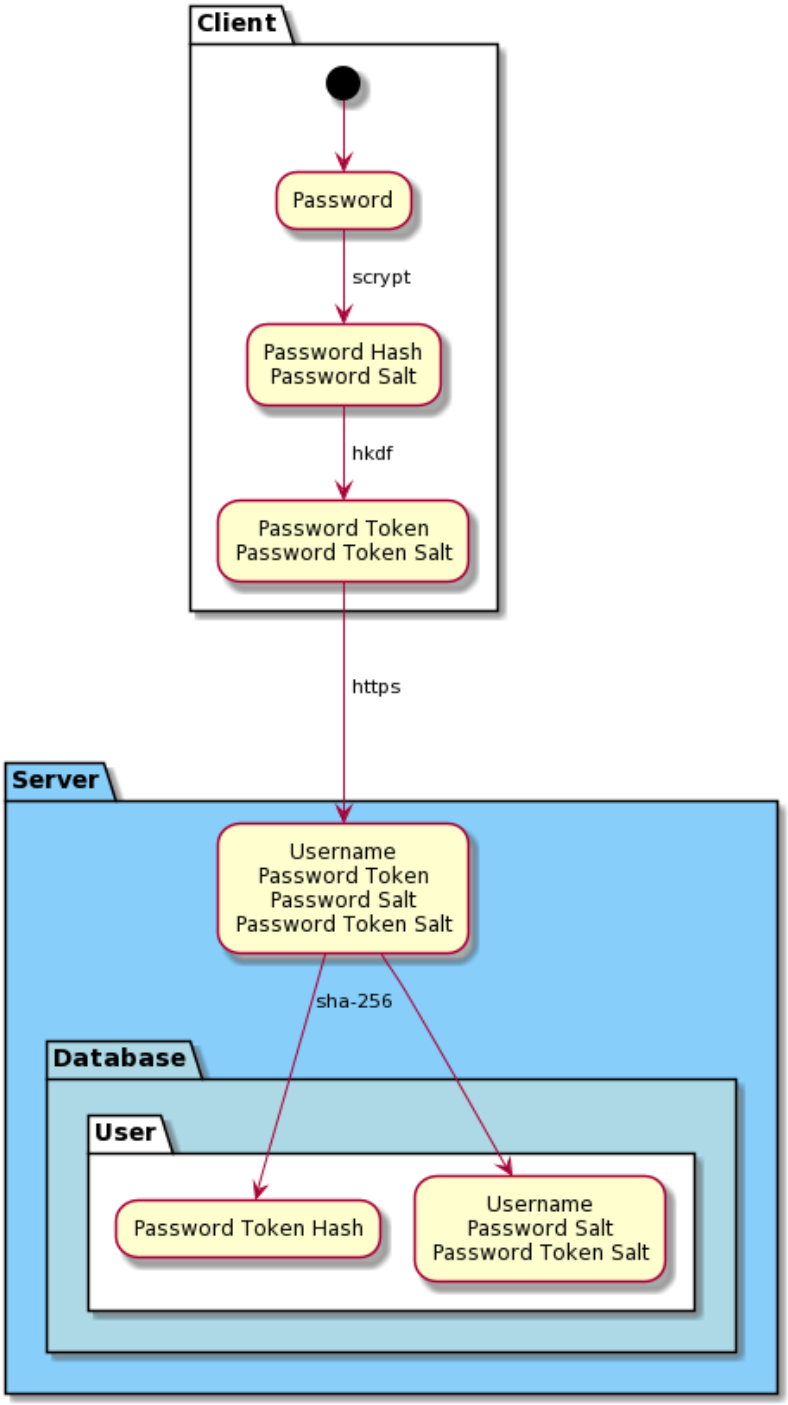
We hash passwords client-side to maximize difficulty (within reason) for someone with access to the server to crack passwords.

When creating a new account, the user provides their username/password, the client generates a random salt, then hashes the password with Scrypt. The resulting password hash then goes through HKDF to derive a password token. The client sends the server the salts used to create the password hash and token. Finally, the server hashes the password token before storage with SHA-256.

When logging in, the user provides their username/password. The client then makes an unauthenticated request to the server to retrieve the user's salts. If no user is found, an error is retuned to the client. If a user is found, the server sends the client the user's password salt and password token salt, which the client uses to rebuild the password token and passes it to the server for authentication.



Sign in with client-side password hashing



Account creation with client-side password hashing

Sign Up

· Client-side entry point.
· Server-side entry point.

Sign In

· Client-side entry point.
· Server-side entry point.

Review scope: Review client-side and server-side code and this approach to ensure someone with access to the server cannot easily crack strong passwords.

---

8. Server-side seed backup

To improve the user experience when logging in from new devices, we've added the option to have the user seed backed up on the server-side, encrypted with a password-derived function. This allows users to log in from a new device normally, without having to accept a key exchange grant from another device, or without having to input the seed manually.

When creating a new account, after hashing the password with Scrypt, the client inputs the password hash through HKDF to generate a Password-Based Encryption Key in addition to the Password Token as described in section 7 above. The randomly generated seed from Section 1 is then encrypted using the Password-Based Encryption Key. The client then sends the Password-Based Encryption Key Salt and Password Encrypted Seed to the server, along with the rest of the items noted in Section 7.

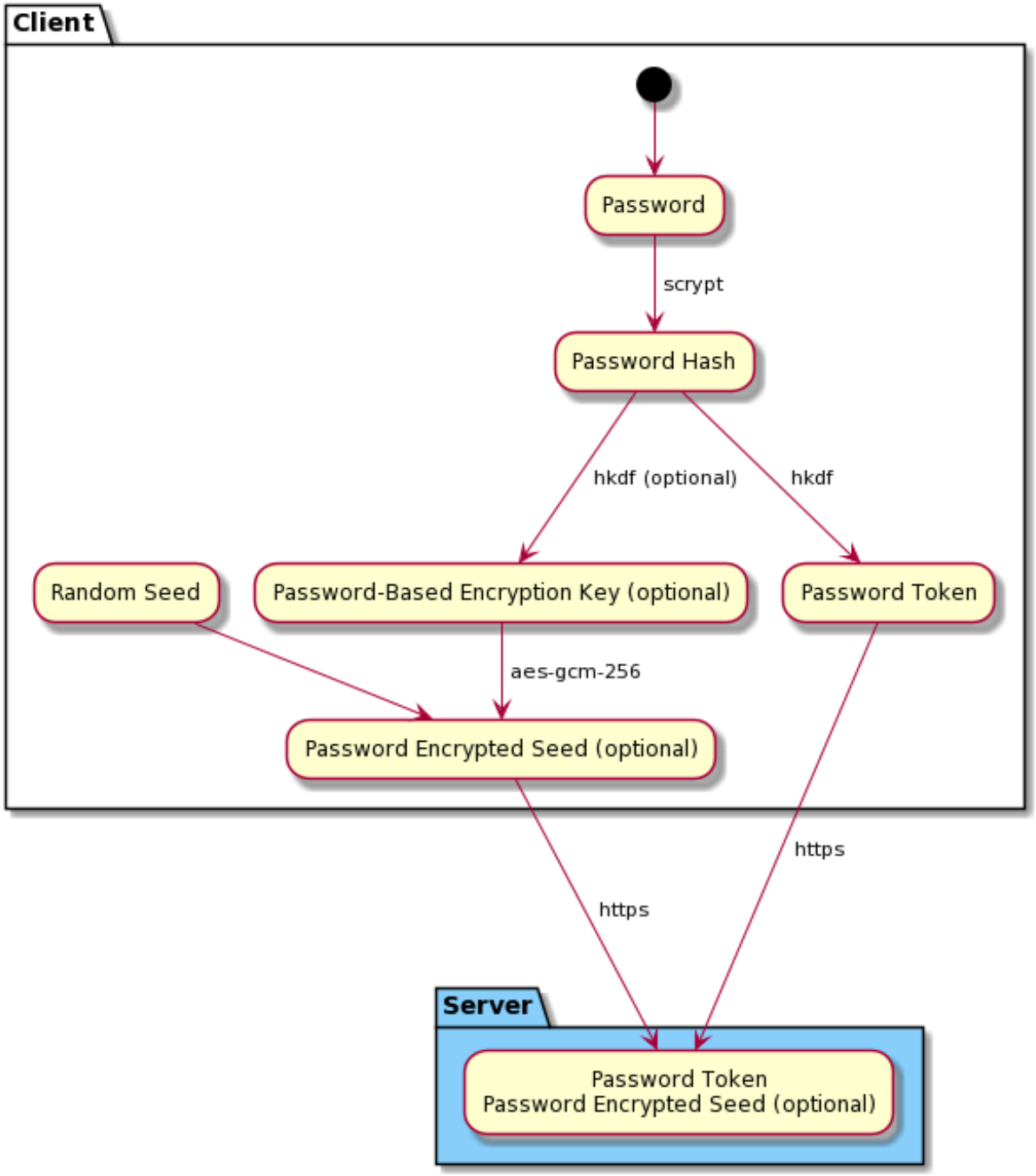
When logging in, after successfully providing the user's password, the server returns the Password-Based Seed Backup (the Password-Based Encryption Key Salt & Password Encrypted Seed) to the client, enabling the client to recover the seed.

Sign Up

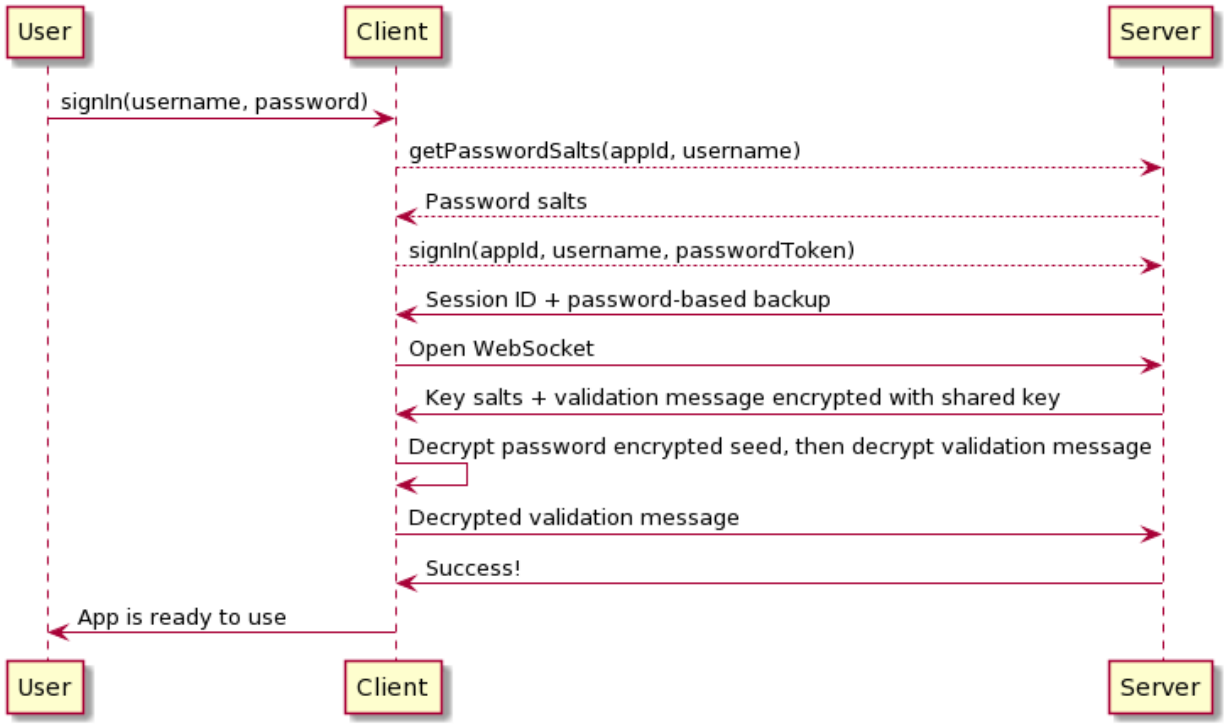
· Client-side entry point.
· Server-side entry point.

Sign In

· Client-side entry point.
· Server-side entry point.



Password-based Seed Backup on Account Creation



Using Password-based Seed Backup on sign in

Review scope: Review client-side and server-side code and this approach to ensure someone with access to the server cannot easily determine a user's seed assuming the user provides a strong password.

---

9. User Forgot Password

When a user forgets their password, we send the user an email with a temporary password to use to sign in. When signing in with the temporary password, the user still needs their seed in order to pass authentication (and subsequently change their password).

To trigger the forgot password mechanism, the user can call an unauthenticated endpoint with their username. If the user has an email saved on the server, the server will generate a random temporary password, then use the user's stored password salts to generate a temporary password token (the same exact way the client generates a password token described in Section 7: Client-side password hashing), then send the user an email with the plaintext temporary password.

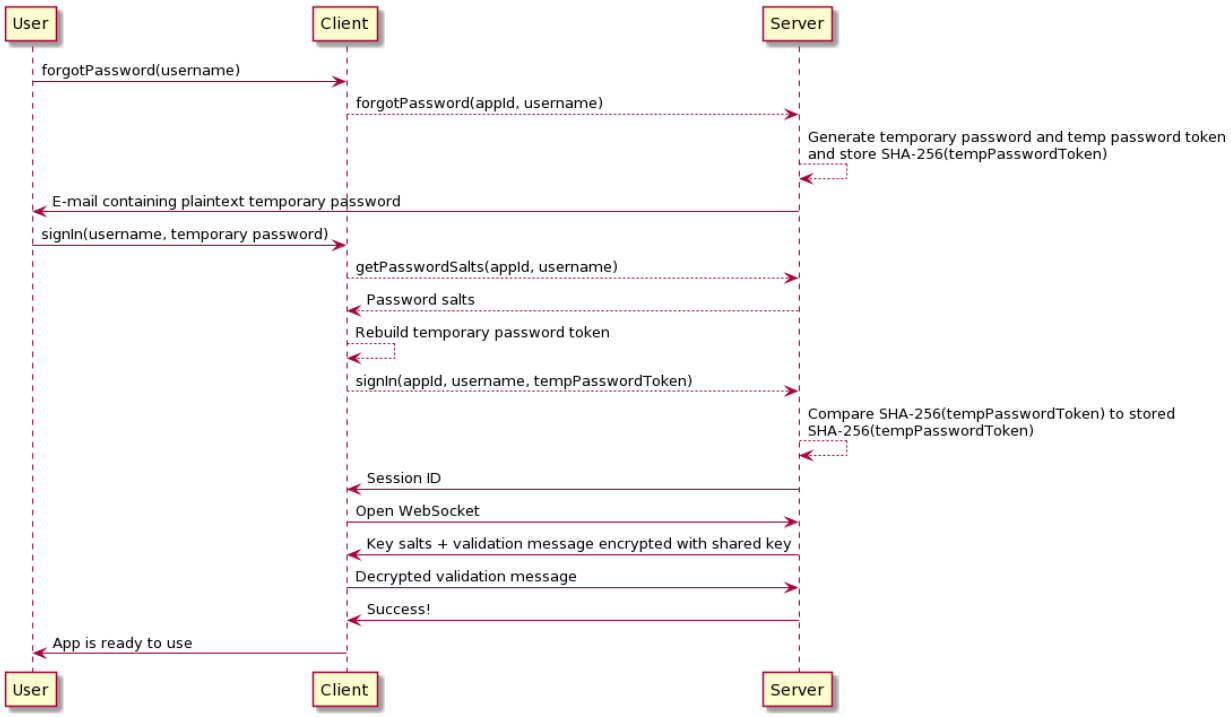
When the user signs in with the temporary password, the client rebuilds the temporary password token and passes it to the server. The server will see that the temporary password is valid, but the user still needs to prove ownership of the seed to pass authentication and sign in as described in Section 2 User Session Management.

Forgot Password

· Client-side entry point.
· Server-side entry point.

Sign In

· Client-side entry point.
· Server-side entry point.



Forgot Password

Review scope: Review client-side and server-side code and this approach to ensure the temporary password mechanism is safe and that user data is not put at greater risk if someone knows the user's temporary password but not the user's seed.