## 5 PROTOCOL FLOW:

### 5.1 Data Model and Algorithms:

WebAuthn Server-Side Data Model (YubiKey)
Let the following notations be defined:

UID ⟶ User Identifier
CID ⟶ Credential ID
PK ⟶ Public Key (COSE encoded)
SK ⟶ Private Key (stored in YubiKey, never exposed)
CHL ⟶ Server-generated Challenge
SC ⟶ Signature Counter
RelyingPartyID ⟶ Domain identifier (e.g., example.com)
Origin ⟶ Web Origin
AuthData ⟶ Authenticator Data
clientDataJSON ⟶ Client-provided JSON
AttObj ⟶ Attestation Object
Sign ⟶ Cryptographic Signature

Credential Registration Model:
challengeCHL ::= random(32 bytes)
registrationRequest ::= ⟨UID, CHL, RelyingPartyID⟩
attestationResponse ::= ⟨CID, PK, SC, clientDataJSON, AttObj⟩
storedCredential ::= ⟨UID, CID, PK, SC⟩

Credential Authentication Model:
challengeCHL ::= random(32 bytes)
authenticationRequest ::= ⟨UID, CHL, CID, RelyingPartyID⟩
assertionResponse ::= ⟨CID, Sign(SK, CHL‖AuthData), SC, clientDataJSON, AuthData⟩
verification ::= verify(Sign, PK, CHL‖AuthData)

Credential Registration with YubiKey
Handled by: Relying Party Server

1: function handleCredentialRegistration(registrationResponse)
2: CID ← registrationResponse.CID
3: PK ← registrationResponse.PK
4: clientData ← parse(registrationResponse.clientDataJSON)
5: challenge ← session.get("challenge")

```
6:    if clientData.challenge ≠ challenge or clientData.origin ≠ expectedOrigin then
7:        return "Invalid registration"
8:    if verifyAttestation(registrationResponse.AttObj, PK) = False then
9:        return "Attestation failed"
10:   SC ← registrationResponse.SC
11:   DB.store(UID, CID, PK, SC)
12:   return "Registration successful"
```

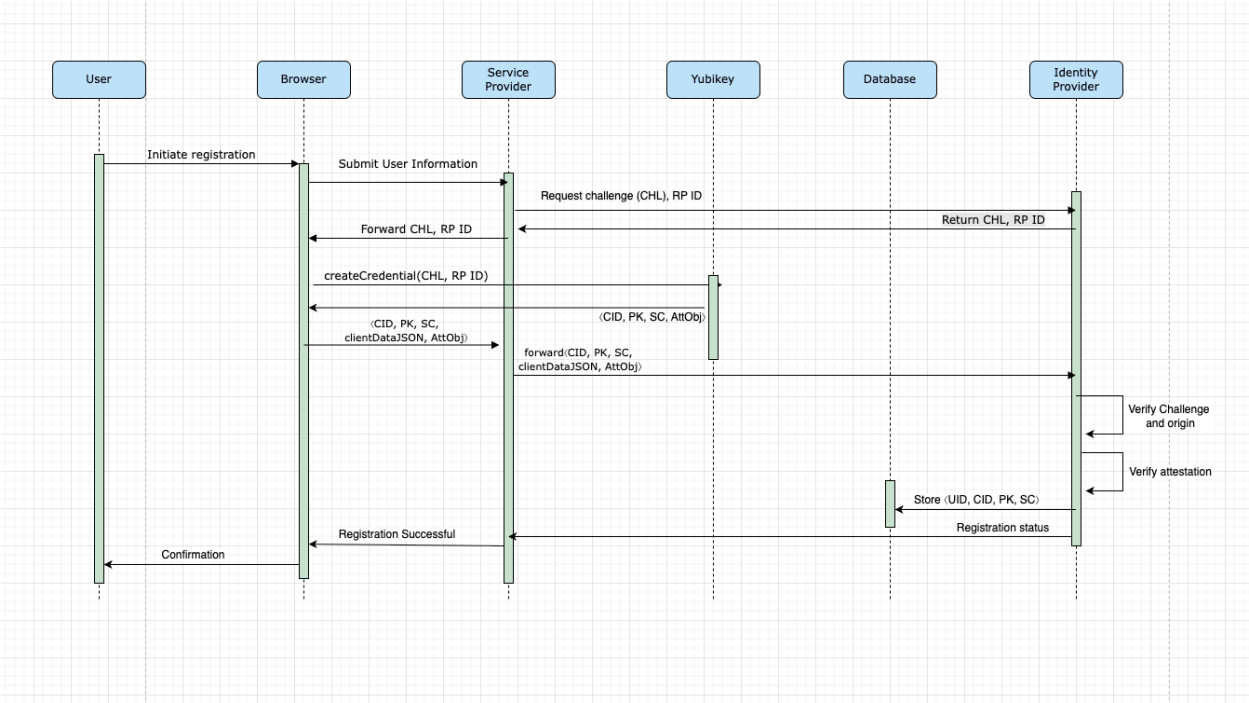
Credential Authentication with YubiKey
Handled by: Relying Party Server

```
1: function handleCredentialAuthentication(assertionResponse)
2:    CID ← assertionResponse.CID
3:    storedPK, storedSC ← DB.lookup(CID)
4:    challenge ← session.get("challenge")
5:    clientData ← parse(assertionResponse.clientDataJSON)
6:    if clientData.challenge ≠ challenge or clientData.origin ≠ expectedOrigin then
7:        return "Invalid challenge or origin"
8:    valid ← verify(assertionResponse.signature, storedPK, challenge ‖
assertionResponse.AuthData)
9:    if valid = False then
10:       return "Authentication failed"
11:   if assertionResponse.SC ≤ storedSC then
12:       return "Replay detected"
13:   DB.updateSignCount(CID, assertionResponse.SC)
14:   return "Authentication successful"
```

### 5.2 Use-case & Protocol flow:

In this section, we explore a number of use-cases to illustrate how a user would interact with the system using a number of protocol flows. We consider two different use-cases: Registration and Login. We discuss each of the use-cases in the following.

**Registration**:




**M1:** The user starts the registration process on a website or app by clicking a "Register" or "Sign up" button.

**M2**: The browser submits the user information (e.g., username or email) to the Service Provider (web application backend).

**M3**: The Service Provider requests a challenge (CHL) and RP ID (Relying Party Identifier) from the Identity Provider. The challenge is a random value used to ensure the response is fresh and to prevent replay attacks. The RP ID identifies the web origin (like example.com).

**M4**: The Identity Provider generates and returns the challenge and RP ID to the Service Provider.

**M5**: The Service Provider forwards the challenge and RP ID to the browser.

**M6**: The browser calls createCredential(CHL, RP ID) using the WebAuthn API. The browser sends this info to the YubiKey, which is a FIDO2-compliant authenticator.

**M7**: The YubiKey generates CID (Credential ID - a unique ID for this credential), PK (Public Key), SC (Signature Counter), AttObj (Attestation Object-used to verify the device's identity) and returns all this to the browser.

**M8**: The browser forwards the response from the YubiKey, including CID, PK, SC, clientDataJSON, AttObj, clientDataJSON (Encoded client-side information-challenge, origin, etc.)

**M9**: The Service Provider forwards everything to the Identity Provider for verification.

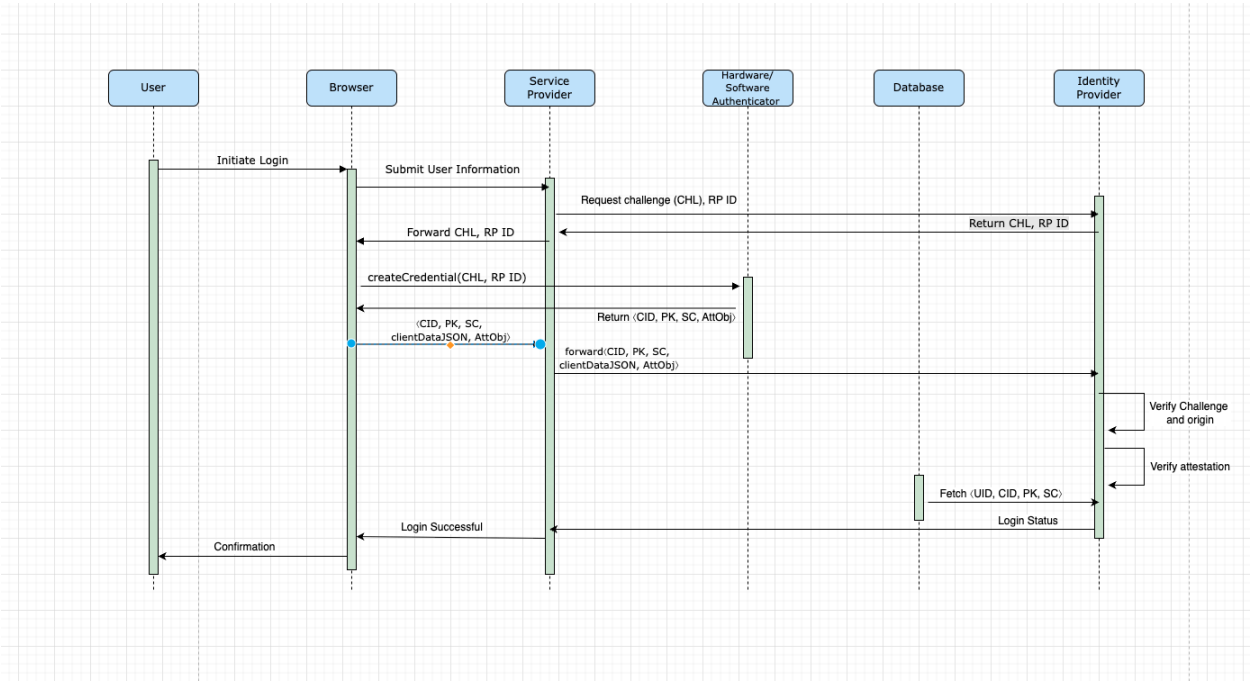
**M10**: Identity provider verifies challenge and origin which ensures the challenge matches and that it came from a trusted origin. It also verifies Attestation which confirm that the device (YubiKey) is authentic.

**M11**: Identity Provider stores UID (User ID), CID, PK, SC. These are stored in the Database for future logins.

**M12**: Identity Provider sends the registration status (success/failure) back to the Service Provider.

**M13**: The browser receives a confirmation that registration was successful and informs the user.

**LogIn**:




M1: The user triggers the login process (e.g., clicking "Login" on a website).

M2: The browser sends the user's identifier (like email/username) to the Service Provider.

M3: The Service Provider asks the Identity Provider for a login challenge and RP ID (domain name, e.g., `example.com`).

M4: Returns a Challenge and RP ID (used to verify the legitimacy of the login).

M5: Forwards the Challenge and RP ID to the Browser.

M6: The browser invokes `navigator.credentials.get()` using the challenge and RP ID. The Authenticator (e.g., biometrics, security key) performs user verification.

M7: Authenticator Returns `CID`: Credential ID, PK: Public Key, SC: Signature Counter, `clientDataJSON`: Contains challenge, origin, and type, `AttObj`: Attestation Object or

assertion result

M8: Browser Sends the assertion (authentication data) to the Service Provider.

M9: Service Provider Forwards `CID`, `PK`, `SC`, `clientDataJSON`, `AttObj`

`M10:` Identity Provider validates that the challenge matches what was sent and Verifies that the origin is trusted.

M11: Identity Provider Retrieves stored credential data using the `CID`.

M12: Identity Provider Compares the public key and verifies the digital signature and Confirms signature counter is increasing (helps detect cloning).

M13: Identity Provider Sends result of authentication to service provider (success or failure).

M14: User is logged in successfully and receives confirmation.