ADR-015: Moneta Network Authentication and Identification

ADR-015: Moneta Network Authentication and Identification

Status: Proposed

Date: 2025-06-01

1. Context

The Moneta Network is a platform involving Membership Organizations (MOs), Publishers, and a central Moneta Core. It aims to facilitate user authentication for accessing Publisher services and manage transactions, similar to a payment network. MO users leverage an mPass (analogous to a Visa card), with private keys stored on their MO-managed mobile application and public keys in Moneta Core. The platform will operate globally.

Current Challenges & Requirements:

- **Authentication:** Design a secure OIDC-based passwordless authentication system managed by Moneta Core, using AWS Cognito, Lambda, DynamoDB, and API Gateway. Common flows involve users clicking a "Login with Moneta mPass" button on Publisher sites/apps, redirecting to Moneta Core's hosted UI, authenticating via their mPass mobile app (QR scan, OTP), and then being redirected back to the Publisher with OIDC tokens. The system must support session management, token revocation, refresh tokens, and SSO for publisher groups.
- **Identification:** The current UUIDv4 identifiers for MOs, Publishers, and mPasses (MO UUID + User UUID) are not human-friendly. A new coding/alias scheme is needed, supporting global scale, high availability, performance, and security.
- **Lifecycle Management:** The identifier system must support the lifecycle of MOs, Publishers (pending, destroy) and mPasses (pending, destroy, expiration, user lock, moving mPass between devices, mobile break/recovery).
- **Key Management:** An mPass should link to one active key pair and store several expired key pairs. Multiple encryption and signing algorithms need to be supported/suggested.
- **mPass Tiers:** Each mPass should be assigned a tier (e.g., Standard, Platinum). Tier information will influence policies like consumption limits and accepted debt amounts. Business flows for tier management (e.g., upgrades) are needed.
- **Billing Account Management:** mPasses, issued by MOs, need to be associated with billing accounts. Types include:
  - **Individual Account:** Default; user is responsible for payments.
  - **Company Account:** Multiple mPasses under one account; company owner handles payments.
  - **Family Account:** Similar to Company, but one specific mPass acts as account owner, sets policies for other mPasses under the account, and is responsible for all charges.
- **Technical Stack:** Primarily AWS serverless (Cognito, Lambda, DynamoDB, API Gateway), but open to other AWS services (S3, RDS, SNS). EKS is mentioned but serverless is preferred for this module.
- **Non-Functional Requirements:** Low authentication latency, data synchronization capabilities between Moneta Core modules, extensibility for future auth methods.
- **Development & Operations:** Recommendations for Lambda development stack and CI/CD.
- **Risk Assessment:** Security risks, pros, and cons for the overall solution and specific authentication methods.

Goals:

1. To design a robust, secure, and user-friendly OIDC passwordless authentication mechanism.
2. To propose a human-readable and manageable identification scheme for MOs, Publishers, and mPasses, incorporating tier and billing account structures.
3. To outline the technical architecture and operational considerations for these systems, ensuring scalability, security, and maintainability.

2. Main Use Cases

This section outlines the primary business functions addressed by this ADR, grouped by functional area.

2.1. Authentication Flows

This group covers how users authenticate to access publisher services and how their sessions are managed.

Use Cases:

- User Authenticates to Publisher via Moneta Core
- User Logs Out from Publisher
- User Experiences SSO across Grouped Publishers
- Moneta Core System Manages User Session

Narrative:

The primary authentication flow involves a User attempting to access a Publisher System. The Publisher System redirects the User to the Moneta Core System for authentication. The User selects a passwordless method (e.g., QR scan, OTP) and uses their mPass Mobile App to complete the challenge presented by Moneta Core. Upon successful authentication, Moneta Core issues OIDC tokens back to the Publisher System, which then grants access to the User.

Beyond the initial login, the system supports:

- **Logout:** The User can initiate a logout from the Publisher System, which communicates with Moneta Core to invalidate the session and relevant tokens.
- **Refresh Token:** Publisher Systems can use refresh tokens to obtain new access tokens from Moneta Core without requiring the User to re-authenticate, maintaining a seamless experience.
- **SSO (Single Sign-On):** If a User is already authenticated with Moneta Core and accesses another Publisher System belonging to an affiliated group, Moneta Core facilitates SSO, allowing the User to access the new Publisher service without re-entering credentials.
- **Session Management:** Moneta Core is responsible for the overall lifecycle of the User's session, including creation, maintenance, and termination based on activity, token expiry, or explicit logout.

2.2. Partner (MO & Publisher) Management

This group covers the administrative functions for managing Membership Organizations (MOs) and Publishers on the Moneta Network.

Use Cases:

- Moneta Core Admin Onboards New MO
- Moneta Core Admin Views MO Details
- Moneta Core Admin Updates MO Configuration
- Moneta Core Admin Manages MO Lifecycle (e.g., Suspend, Activate, Destroy)
- Moneta Core Admin Onboards New Publisher
- Moneta Core Admin Views Publisher Details
- Moneta Core Admin Updates Publisher Configuration
- Moneta Core Admin Manages Publisher Lifecycle (e.g., Suspend, Activate, Destroy)

Narrative:

Partner management is primarily handled by a Moneta Core Admin interacting with the Moneta Core System.

For Membership Organizations (MOs), the Admin can perform CRUD (Create, Read, Update, Delete - though 'Delete' is more accurately 'Destroy' in lifecycle terms) operations. This includes onboarding new MOs, configuring their parameters (e.g., assigning MICs, setting up communication endpoints), viewing their status and details, and managing their lifecycle (pending approval, active, suspended, destroyed). Changes in the Moneta Core System might trigger notifications or updates to the MO's own external systems.

Similarly, for **Publishers**, the Moneta Core Admin manages their onboarding, configuration (e.g., assigning MPCs, defining service endpoints, associating with SSO groups), viewing details, and lifecycle management. These actions ensure that only legitimate and correctly configured partners operate on the network.

2.3. mPass Management

This group covers the lifecycle and attribute management of individual mPasses, including their tiers and billing account associations.

Use Cases:

- MO Admin Issues New mPass to User
- User/MO Admin Views mPass Details
- User/MO Admin Updates mPass (e.g., user-lock, MO-initiated suspension)
- User/MO Admin Manages mPass Lifecycle (Destroy, Reactivate if applicable)
- Moneta Core System Manages mPass Expiration
- User Manages mPass Keys (implicitly, via device transfer/recovery flows)
- MO Admin Assigns/Upgrades/Downgrades mPass Tier
- MO Admin Associates mPass with Billing Account
- User (as Billing Account Owner) Manages Linked mPasses (e.g., for Family Accounts, setting policies)
- User (as Billing Account Owner) Views Consolidated Billing Information (handled by MO, Moneta Core provides data)

Narrative:

mPass management involves multiple actors and system interactions:

- **Issuance & Lifecycle:** An **MO Admin** issues an mPass to a **User** via the **Moneta Core System**. The User activates and uses the mPass through their **mPass Mobile App**. Both the User (e.g., locking their mPass) and the MO Admin (e.g., suspending an mPass) can manage aspects of its lifecycle. Moneta Core automatically handles processes like mPass expiration.

- **Key Management:** While not a direct CRUD operation by the user on keys, processes like transferring an mPass to a new device or recovering an mPass after a device break implicitly involve the generation of new key pairs on the mPass Mobile App, with the new public key being registered with Moneta Core.
- **Tier Management:** The MO Admin is responsible for assigning an initial tier to an mPass and managing upgrades or downgrades based on business rules or user requests. This information is stored by Moneta Core and affects the policies applied to the mPass.
- **Billing Account Management:** The MO Admin associates an mPass with a specific billing account (Individual, Company, or Family) within the Moneta Core System. For Family accounts, the designated **Billing Account Owner** (a User) can manage policies (e.g., spending limits) for other mPasses linked to their family account, typically via their mPass Mobile App, which then communicates these policy changes to Moneta Core.

These use cases define the core interactions for identity, access, and account management within the Moneta Network from the perspective of this ADR.

3. Decision

We will implement a **custom OIDC passwordless authentication flow using AWS Cognito with AWS Lambda triggers** for challenge management. For identifiers, we will introduce a **structured, human-friendly coding scheme** managed centrally by Moneta Core, alongside the existing UUIDs for internal referencing. This system will also incorporate **mPass tiers and a flexible billing account structure.**

Key Components of the Decision:

OIDC Passwordless Authentication (Moneta Core Managed):

- **Identity Provider:** AWS Cognito User Pools will serve as the OIDC IdP.
- **Custom Authentication Flow:** Leverage Cognito's Define Auth Challenge, Create Auth Challenge, and Verify Auth Challenge Response Lambda triggers to implement passwordless methods (QR Code Scan, OTP to mPass App).
- **Hosted UI:** Cognito's Hosted UI will be customized to present these passwordless options.
- **API Gateway & Lambda:** API Gateway will expose necessary endpoints for the mPass mobile app to interact with during authentication (e.g., submit signed QR data, receive OTP instructions if not purely push-based). These will be backed by Lambda functions.
- **DynamoDB:** Used to store temporary challenge data (QR session IDs, OTP hashes), mPass public key references, and other authentication metadata.
- **Standard OIDC Features:** Session management, token revocation, and refresh token capabilities will be handled by Cognito.
- **SSO:** Cognito's native support for SSO across different app clients within the same user pool will be used for publisher groups.

Human-Friendly Identifiers, Tiers, Billing, & Lifecycle Management:

- **MO Code (Moneta Issuer Code - MIC):** A short, centrally assigned alphanumeric code (e.g., MOA01, 4-6 chars).
- **Publisher Code (Moneta Partner Code - MPC):** A short, centrally assigned alphanumeric code (e.g., PUBS7, 4-6 chars).
- **mPass Number:** A structured number, analogous to payment card numbers:
  - **MII (Moneta Issuer Identifier):** First 6-8 digits, globally unique, identifying the issuing MO (linked to MIC).
  - **MAI (Moneta Account Identifier):** Next 8-12 digits, assigned by the MO, unique within that MO.
  - **VC (Verification Character):** 1 character (e.g., Luhn check digit).
  - Example: 123456-7890123456-A.
- **mPass Tiers:** An attribute associated with each mPass (e.g., "Standard", "Platinum") influencing its policies.
- **Billing Accounts:** A separate structure to manage how mPass charges are aggregated and paid, supporting Individual, Company, and Family models.
- **Storage & Management:** These codes, tiers, and billing account associations will be stored in DynamoDB, linked to their respective UUIDs. Moneta Core will manage their lifecycle (Pending, Active, Suspended, Destroyed, Expired) via APIs and automated processes.
- **mPass Key Management:**
  - Each mPass will be associated with one active public key and a history of inactive (expired, superseded) public keys.
  - Public keys (with algorithm identifiers) stored in Moneta Core's secure storage. Private keys remain solely on the user's mPass mobile app.
- **Supported Algorithms:**
  - **Signing:** ECDSA (e.g., P-256, P-384), EdDSA (e.g., Ed25519).
  - **Encryption (for data at rest within Moneta Core, if needed beyond standard SSE):** AES-256 GCM.

4. Consequences

Pros:

- ✅ **Enhanced User Experience:** Passwordless methods are generally faster and more user-friendly. Human-readable codes improve operational clarity.
- ✅ **Flexible Financial Management:** Tier and billing account structures allow for diverse product offerings and payment arrangements.
- ✅ **Leverages AWS Managed Services:** Reduces operational burden for Cognito, Lambda, DynamoDB, API Gateway, ensuring scalability and reliability.
- ✅ **Strong Security Foundation:** OIDC is an industry standard. Custom Lambda triggers allow granular control over authentication logic. Private keys remain on user devices, reducing central risk.
- ✅ **Extensibility:** Cognito's custom auth flow allows adding new passwordless methods in the future. Tier and billing models can also be expanded.
- ✅ **Centralized Control & Consistency:** Moneta Core manages authentication, the identifier namespace, tiers, and billing account structures.
- ✅ **SSO Capability:** Natively supported by Cognito for publisher groups.
- ✅ **Global Scalability:** AWS services provide global infrastructure. DynamoDB Global Tables can be used for low-latency reads of identifier/key data if needed.

Cons:

- ❗ **Increased Implementation Complexity:** Custom authentication flows, tier logic, and billing account management add layers of complexity to design, development, and testing.
- ❗ **AWS Ecosystem Lock-in:** Significant reliance on AWS services.
- ❗ **Lambda Cold Starts:** Potential for increased latency for infrequently used auth/management functions if not mitigated (e.g., provisioned concurrency).

- ❗ **Mobile App Dependency:** The entire passwordless flow hinges on the security and functionality of the MO's mPass mobile application, which may also need to display tier/billing info.
- ❗ **Data Management Overhead:** Centralized management of codes, tiers, billing accounts, and their interdependencies requires robust processes and potentially more complex data models.

Risks & Mitigations:

- **Security Risks:**
  - **Compromised Lambda Functions:**
    - *Mitigation:* Strict IAM roles (least privilege), regular code reviews, dependency scanning, security testing (SAST/DAST), AWS WAF on API Gateway.
  - **Insecure Mobile App Key Storage:**
    - *Mitigation:* Mandate/strongly recommend MOs to ensure private keys are stored in hardware-backed secure enclaves. Provide clear security guidelines.
  - **QR Code Hijacking/Replay Attacks:**
    - *Mitigation:* Short-lived, single-use QR codes/session IDs. Signing of the QR challenge response by the mPass app. Secure HTTPS. User education.
  - **OTP Interception/Phishing:**
    - *Mitigation:* Short-lived OTPs. Binding OTPs to sessions. Rate limiting. Secure delivery. User education.
  - **Identifier Enumeration/Guessing:**
    - *Mitigation:* Avoid strictly sequential assignment. Rate limiting on lookup APIs. Strong validation.
  - **Unauthorized Tier/Billing Modification:**
    - *Mitigation:* Strong authorization controls on APIs managing tiers and billing accounts (e.g., only MO admins or designated account owners can make changes). Audit trails for all modifications.
  - **Denial of Service (DoS) against Auth Endpoints:**
    - *Mitigation:* AWS Shield, AWS WAF, scalable serverless infrastructure.
- **Operational Risks:**
  - **Moneta Core Auth Module as Central Point of Failure:**
    - *Mitigation:* Multi-AZ deployments. Consider multi-region for critical components.
  - **Identifier/Code Collision:**
    - *Mitigation:* Robust central management with atomic operations or strong consistency for assigning unique codes.
  - **Data Synchronization Failures:**
    - *Mitigation:* Reliable eventing (EventBridge) with DLQs. Monitoring. Idempotent consumers.
  - **Complexity in Policy Enforcement:** Ensuring correct application of policies based on tiers and family billing account rules requires careful logic and testing.
    - *Mitigation:* Thorough testing of policy logic. Clear definition of policy rules. Versioning of policies.

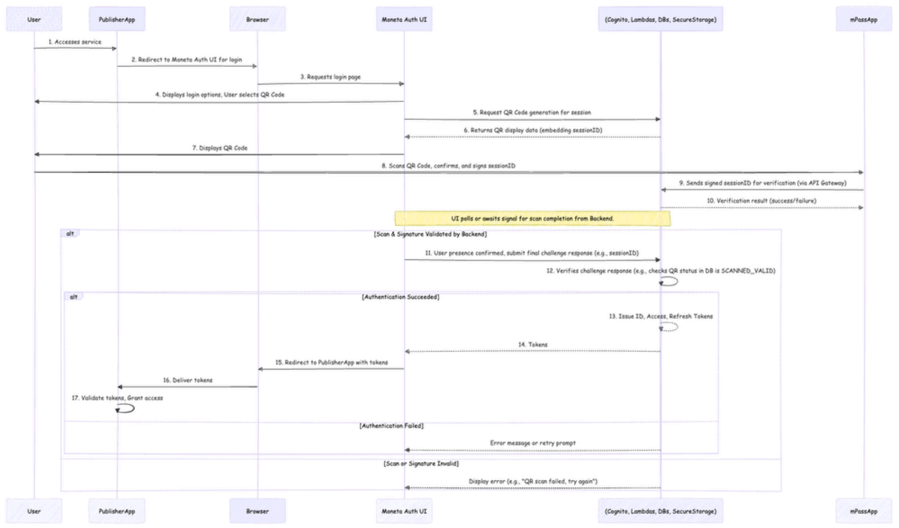
5. Technical Details

5.1. OIDC Passwordless Authentication Flow

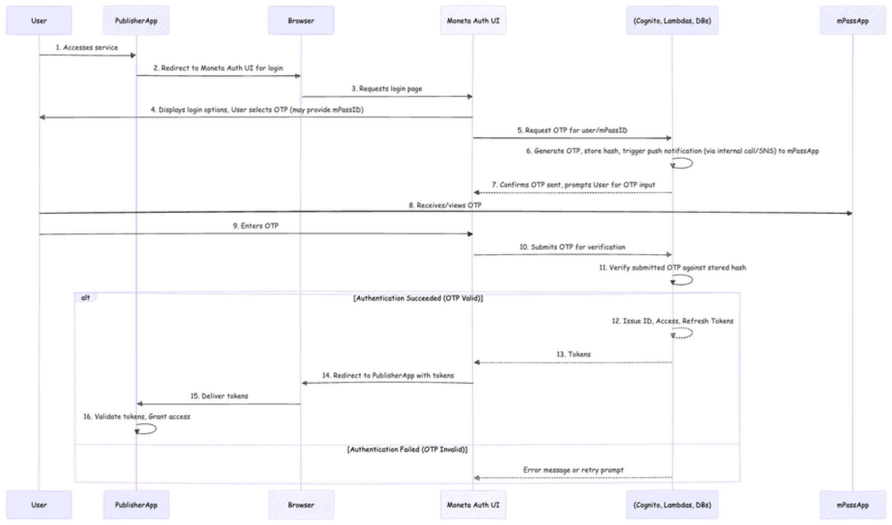
```
sequenceDiagram
    participant User
    participant PublisherApp as Publisher App/Website
    participant Browser
    participant MonetaHostedUI as Moneta Core Auth Hosted UI (Cognito)
    participant Cognito
    participant APIGW as Moneta Core API Gateway
    participant LambdaAuth as Auth Lambdas (Define, Create, Verify Challenge)
    participant DynamoDBAuth as Auth DynamoDB (Challenges, mPass Public Keys)
    participant mPassApp as MO mPass Mobile App
    participant MonetaCoreSecure as Moneta Core Secure Storage (Primary mPass Public Key Store)

    User->>PublisherApp: Accesses service
    PublisherApp->>Browser: Redirect to MonetaHostedUI (triggers /oauth2/authorize with client_id, redirect_uri, scope, response_type)
    Browser->>MonetaHostedUI: Request login page
    MonetaHostedUI->>User: Display passwordless options (e.g., "Scan QR", "Send OTP to App")

    alt QR Code Flow
        User->>MonetaHostedUI: Selects QR Code option
        MonetaHostedUI-->>Cognito: Initiate Custom Authentication (USER_SRP_AUTH or similar to start custom flow)
        Cognito->>LambdaAuth: Invoke Define Auth Challenge Lambda
        LambdaAuth-->>Cognito: Respond with `CUSTOM_CHALLENGE` and `initiate_qr` as next step
        Cognito->>LambdaAuth: Invoke Create Auth Challenge Lambda (for `initiate_qr`)
        LambdaAuth->>DynamoDBAuth: Generate unique sessionID, store {sessionID, expiresAt, status: PENDING_SCAN}
        LambdaAuth-->>Cognito: Return `publicChallengeParameters` (e.g., {sessionID, qrDisplayData}), `privateChallengeParameters`
        Cognito-->>MonetaHostedUI: Challenge details (including sessionID for QR code)
        MonetaHostedUI->>User: Display QR code (embedding sessionID or data resolvable to it)

        User->>mPassApp: Scans QR Code (extracts sessionID)
        mPassApp->>mPassApp: User confirms action. App signs sessionID (or a derived challenge) using mPass private key.
        mPassApp->>APIGW: POST /auth/qr/verify ({sessionID, mPassID, signature})
        APIGW->>LambdaAuth: Invoke QR Verification Lambda
        LambdaAuth->>MonetaCoreSecure: Fetch active public key for mPassID
        MonetaCoreSecure-->>LambdaAuth: Public Key
        LambdaAuth->>LambdaAuth: Verify signature against sessionID and public key
        alt Signature Valid
```

```
            LambdaAuth->>DynamoDBAuth: Update QR challenge: {sessionID, status: SCANNED_VALID, mPassID: mPassID}
            LambdaAuth-->>APIGW: Success (200 OK)
            APIGW-->>mPassApp: Confirmation
            Note over MonetaHostedUI, DynamoDBAuth: Hosted UI polls an API endpoint (or uses WebSocket) that checks DynamoDB for sessionID status.
            MonetaHostedUI->>Cognito: RespondToAuthChallenge (Answer: sessionID, mPassID, clientMetadata: {status: SCANNED_VALID})
        else Signature Invalid or other error
            LambdaAuth-->>APIGW: Failure (e.g., 400, 401)
            APIGW-->>mPassApp: Error message
            MonetaHostedUI->>User: Display error (e.g., "QR scan failed, try again")
        end
    else OTP to mPass App Flow
        User->>MonetaHostedUI: Selects OTP option, may enter mPass Number (or it's pre-filled/derived)
        MonetaHostedUI-->>Cognito: Initiate Custom Authentication (with mPassID if known)
        Cognito->>LambdaAuth: Invoke Define Auth Challenge Lambda
        LambdaAuth-->>Cognito: Respond with `CUSTOM_CHALLENGE` and `initiate_otp`
        Cognito->>LambdaAuth: Invoke Create Auth Challenge Lambda (for `initiate_otp`)
        LambdaAuth->>LambdaAuth: Generate OTP, hash it.
        LambdaAuth->>DynamoDBAuth: Store {mPassID, otpHash, expiresAt, attempts:0}
        LambdaAuth->>APIGW: (Internal call or via SNS) Trigger push notification to mPassApp (via MO's infrastructure or direct if possible) with OTP or prompt.
        LambdaAuth-->>Cognito: Return `publicChallengeParameters` (e.g., {message: "OTP sent to your app"}), `privateChallengeParameters`
        Cognito-->>MonetaHostedUI: Display "OTP sent to your app" and input field for OTP.
        User->>mPassApp: Receives/views OTP.
        User->>MonetaHostedUI: Enters OTP.
        MonetaHostedUI->>Cognito: RespondToAuthChallenge (Answer: OTP, mPassID)
    end

    Cognito->>LambdaAuth: Invoke Verify Auth Challenge Response Lambda
    alt QR Flow Verification
        LambdaAuth->>DynamoDBAuth: Check if sessionID status is SCANNED_VALID for the given mPassID.
        alt Valid
            LambdaAuth-->>Cognito: `answerCorrect = true`
        else Invalid
            LambdaAuth-->>Cognito: `answerCorrect = false`
        end
    else OTP Flow Verification
        LambdaAuth->>DynamoDBAuth: Fetch otpHash for mPassID. Compare submitted OTP (after hashing) with stored hash. Increment attempts. Check expiry.
        alt Valid
            LambdaAuth-->>Cognito: `answerCorrect = true`. Clean up OTP record.
        else Invalid
            LambdaAuth-->>Cognito: `answerCorrect = false`.
        end
    end

    alt Authentication Succeeded (answerCorrect = true)
        Cognito->>Cognito: User is authenticated. Issue ID Token, Access Token, Refresh Token. (Tokens may include tier/billing context if needed, or this info is fetched separately by Publisher using Access Token).
        Cognito-->>MonetaHostedUI: Tokens.
        MonetaHostedUI->>Browser: Redirect to PublisherApp's `redirect_uri` with tokens (e.g., ID token in fragment, authorization code for code flow).
        Browser->>PublisherApp: Deliver tokens/code.
        PublisherApp->>PublisherApp: Validate tokens (check signature, issuer, audience, expiry). Grant access. Publisher may call Moneta Core resource API to get mPass details (tier, limits) using access token.
    else Authentication Failed (answerCorrect = false)
        Cognito->>LambdaAuth: Invoke Define Auth Challenge Lambda again (to retry or fail).
        LambdaAuth-->>Cognito: Potentially `failAuthentication = true` or allow retry.
        Cognito-->>MonetaHostedUI: Error message or retry prompt.
    end


```

QR scan flow


User
PublisherApp
Browser
Moneta Auth UI
(Cognito, Lambdas, DBs, SecureStorage)
mPassApp
1. Accesses service
2. Redirect to Moneta Auth UI for login
3. Requests login page
4. Displays login options, User selects QR Code
5. Request QR Code generation for session
6. Returns QR display data (embedding sessionID)
7. Displays QR Code
8. Scans QR Code, confirms, and signs sessionID
9. Sends signed sessionID for verification (via API Gateway)
10. Verification result (success/failure)
UI polls or awaits signal for scan completion from Backend.
alt
[Scan & Signature Validated by Backend]
11. User presence confirmed, submit final challenge response (e.g., sessionID)
12. Verifies challenge response (e.g., checks QR status in DB is SCANNED_VALID)
alt
[Authentication Succeeded]
13. Issue ID, Access, Refresh Tokens
14. Tokens
15. Redirect to PublisherApp with tokens
16. Deliver tokens
17. Validate tokens, Grant access
[Authentication Failed]
Error message or retry prompt
[Scan or Signature Invalid]
Display error (e.g., "QR scan failed, try again")

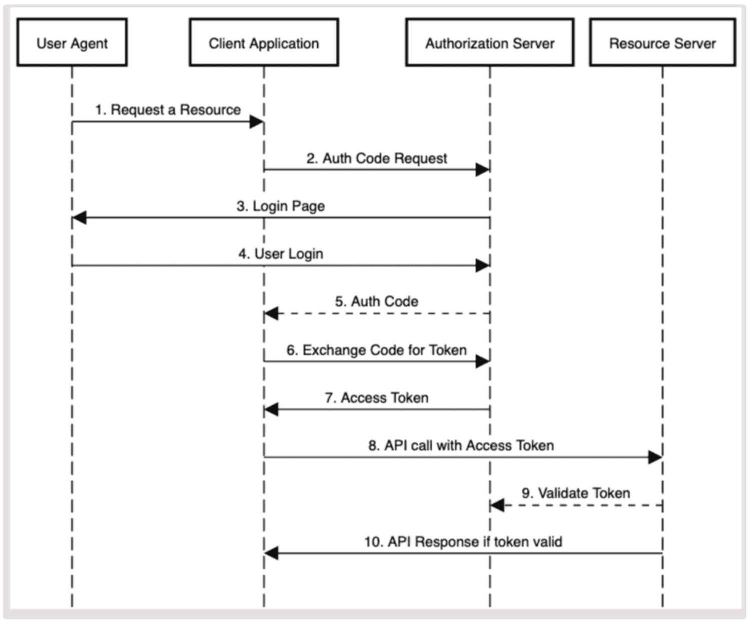
OTP flow


User
PublisherApp
Browser
Moneta Auth UI
(Cognito, Lambdas, DBs)
mPassApp
1. Accesses service
2. Redirect to Moneta Auth UI for login
3. Requests login page
4. Displays login options, User selects OTP (may provide mPassID)
5. Request OTP for user/mPassID
6. Generate OTP, store hash, trigger push notification (via internal call/SNS) to mPassApp
7. Confirms OTP sent, prompts User for OTP input
8. Receives/views OTP
9. Enters OTP
10. Submits OTP for verification
11. Verify submitted OTP against stored hash
alt
[Authentication Succeeded (OTP Valid)]
12. Issue ID, Access, Refresh Tokens
13. Tokens
14. Redirect to PublisherApp with tokens
15. Deliver tokens
16. Validate tokens, Grant access
[Authentication Failed (OTP Invalid)]
Error message or retry prompt

The OIDC passwordless authentication flow enables users to securely access Publisher services using their mPass without traditional passwords. The process is orchestrated by Moneta Core, leveraging AWS Cognito and custom Lambda functions.

1. **Initiation**: A User attempts to access a Publisher's application or website. The Publisher application (acting as an OIDC client) redirects the User's browser to the Moneta Core Authentication Hosted UI (managed by Cognito). This redirection includes standard OIDC parameters like `client_id` (identifying the Publisher app), `redirect_uri` (where to send the User back after authentication), `scope` (requested permissions), and `response_type` (e.g., `code` for Authorization Code Flow).
2. **Challenge Presentation & Selection**: The Moneta Core Hosted UI presents the User with available passwordless authentication options (e.g., "Scan QR Code," "Send OTP to mPass App"). The User selects their preferred method.
3. **Custom Authentication Challenge (Cognito & Lambdas)**:
   - Cognito initiates a custom authentication flow, triggering a sequence of AWS Lambda functions (`Define Auth Challenge`, `Create Auth Challenge`).
   - If **QR Code** is selected: The `Create Auth Challenge` Lambda generates a unique, short-lived session ID and associated QR code data. This data is stored temporarily in DynamoDB (e.g., `QRChallengeSessions` table with status `PENDING_SCAN`) and then presented to the User on the Hosted UI.
   - If **OTP to mPass App** is selected: The `Create Auth Challenge` Lambda generates a secure OTP, stores its hash in DynamoDB (e.g., `OTPChallenges` table), and triggers a mechanism (e.g., via API Gateway and SNS) to send the OTP to the User's registered mPass mobile application. The Hosted UI informs the User that an OTP has been sent and provides an input field.
4. **mPass Mobile App Interaction**:
   - **QR Code**: The User scans the displayed QR code using their mPass mobile app. The app extracts the session ID, signs it (or related challenge data) using the mPass private key stored securely on the device, and sends the session ID, mPass identifier, and the signature to a dedicated API Gateway endpoint in Moneta Core.
   - **OTP**: The User receives the OTP on their mPass mobile app and enters it into the input field on the Moneta Core Hosted UI.
5. **Challenge Verification (Cognito & Lambdas)**:
   - **QR Code**: The API Gateway endpoint receiving the signed QR data invokes a Lambda function. This Lambda retrieves the mPass public key from Moneta Core's secure storage, verifies the signature against the session ID. If valid, it updates the corresponding record in the `QRChallengeSessions` DynamoDB table to `SCANNED_VALID`. The Hosted UI, which may be polling or using a WebSocket, detects this status change and submits the challenge response (containing the session ID and mPass ID) to Cognito. Cognito then invokes the `Verify Auth Challenge Response` Lambda, which confirms the `SCANNED_VALID` status in DynamoDB.
   - **OTP**: The User submits the OTP via the Hosted UI to Cognito. Cognito invokes the `Verify Auth Challenge Response` Lambda. This Lambda retrieves the stored OTP hash for the mPass ID from DynamoDB and compares it with the hash of the submitted OTP, also checking for expiry and attempt limits.
6. **Token Issuance (Authentication Success)**: If the `Verify Auth Challenge Response` Lambda confirms the challenge was successfully met (`answerCorrect = true`), Cognito considers the user authenticated. It then issues standard OIDC tokens:
   - **ID Token**: A JWT containing claims about the authentication event and the user's identity (mPass identifier).
   - **Access Token**: A JWT authorizing the Publisher application to access specific resources on behalf of the user (details of these resources are defined by scopes).
   - **Refresh Token**: A token used to obtain new ID and Access tokens when the current ones expire, without requiring the user to re-authenticate. These tokens are returned to the Moneta Core Hosted UI.
7. **Redirection to Publisher**: The Moneta Core Hosted UI redirects the User's browser back to the Publisher application's pre-registered `redirect_uri`, including the OIDC tokens (typically, the ID token and access token in the URL fragment for implicit flow, or an authorization code for authorization code flow).
8. **Access Granted by Publisher**: The Publisher application receives the tokens (or exchanges the code for tokens). It validates the ID Token and Access Token (checking signature, issuer, audience, expiry). Upon successful validation, the Publisher grants the User access to its services or content. The Publisher may also use the Access Token to call Moneta Core resource APIs to fetch additional mPass details (like tier or specific limits) if needed.
9. **Authentication Failure**: If at any point the challenge verification fails (e.g., invalid QR signature, incorrect OTP, expired challenge), the `Verify Auth Challenge Response` Lambda returns `answerCorrect = false`. Cognito may then re-invoke the `Define Auth Challenge` Lambda to either allow the user to retry the challenge (e.g., re-enter OTP, rescan QR) or to ultimately fail the authentication attempt after a certain number of retries. The Hosted UI will display an appropriate error message to the User.

Publisher System Token Usage (Resource Server Interaction):



Once the Publisher Application (acting as an OIDC client) receives the ID Token and Access Token for the mPass user, the Access Token is used to authorize requests to the Publisher's backend services (Resource Servers).

The mPass Access Token, obtained by the Publisher Application Frontend/Client after successful user authentication with Moneta Core, serves as a bearer token.

1. **Client-Side Storage:** The Publisher Application Frontend securely stores this access token (e.g., in memory, HttpOnly secure cookie for web apps).

2. **API Requests:** When the user performs actions requiring authorization (e.g., purchasing a plan, accessing protected content, unsubscribing), the Publisher Application Frontend includes the mPass Access Token in the Authorization header of requests to its backend services (e.g., Paywall Service, Subscription/Billing Service, Core Service Logic).
3. **Resource Server Validation:** Each Publisher backend service, acting as a Resource Server, validates the incoming Access Token. This typically involves:

- Checking the token's signature against Moneta Core's public keys.
- Verifying the token's expiration time (exp claim).
- Ensuring the token's audience (aud claim) matches the Publisher service.
- Checking the issuer (iss claim) is Moneta Core.

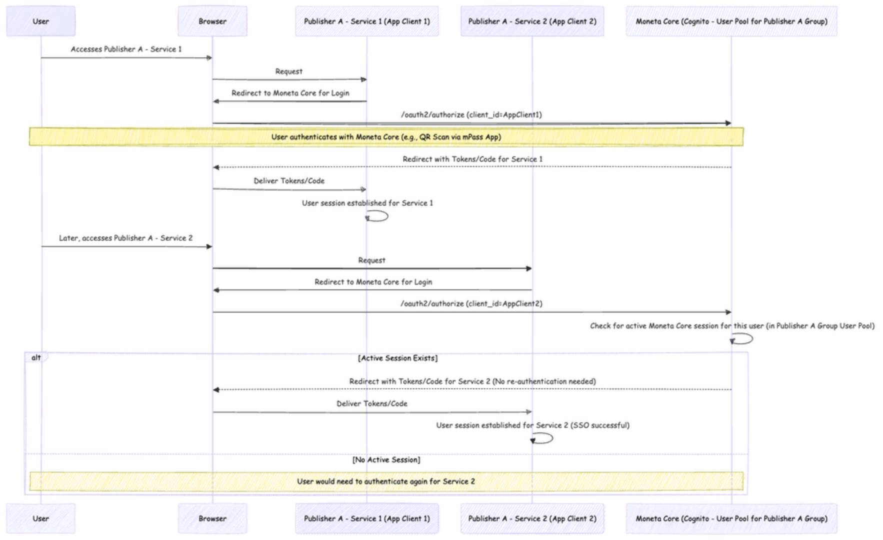
1. **Accessing mPass Details (Optional):** If a Publisher service needs more details about the mPass (e.g., current tier, specific consumption limits not included in the token, mPass status), it can make a server-to-server call to a dedicated Moneta Core Resource API. This call would be authenticated either using the mPass Access Token itself (if Moneta Core allows token introspection or delegation for this purpose) or a pre-established system-level credential between the Publisher and Moneta Core.
2. **Business Logic Execution:** Based on successful token validation and any necessary mPass details, the Publisher service executes the requested business logic (e.g., processes the purchase, grants access to content, updates subscription status).
3. **Response to Client:** The Publisher service then responds to the Publisher Application Frontend.

This model ensures that access to the Publisher's resources is protected and granted only to authenticated and authorized mPass users, with policies potentially influenced by mPass-specific attributes managed by Moneta Core.

SSO within a Publisher Group:

If multiple Publisher applications (e.g., Publisher A's streaming service, Publisher A's gaming portal) are configured as distinct App Clients under the *same Cognito User Pool* (representing the "Publisher A Group"), Moneta Core facilitates Single Sign-On (SSO) across these applications.

```
sequenceDiagram
    participant User
    participant Browser
    participant PublisherA_App1 as Publisher A - Service 1 (App Client 1)
    participant PublisherA_App2 as Publisher A - Service 2 (App Client 2)
    participant MonetaCore_Cognito as Moneta Core (Cognito - User Pool for Publisher A Group)

    User->>Browser: Accesses Publisher A - Service 1
    Browser->>PublisherA_App1: Request
    PublisherA_App1->>Browser: Redirect to Moneta Core for Login
    Browser->>MonetaCore_Cognito: /oauth2/authorize (client_id=AppClient1)
    Note over User,MonetaCore_Cognito: User authenticates with Moneta Core (e.g., QR Scan via mPass App)
    MonetaCore_Cognito-->>Browser: Redirect with Tokens/Code for Service 1
    Browser->>PublisherA_App1: Deliver Tokens/Code
    PublisherA_App1->>PublisherA_App1: User session established for Service 1

    User->>Browser: Later, accesses Publisher A - Service 2
    Browser->>PublisherA_App2: Request
    PublisherA_App2->>Browser: Redirect to Moneta Core for Login
    Browser->>MonetaCore_Cognito: /oauth2/authorize (client_id=AppClient2)
    MonetaCore_Cognito->>MonetaCore_Cognito: Check for active Moneta Core session for this user (in Publisher A Group User Pool)
    alt Active Session Exists
        MonetaCore_Cognito-->>Browser: Redirect with Tokens/Code for Service 2 (No re-authentication needed)
        Browser->>PublisherA_App2: Deliver Tokens/Code
        PublisherA_App2->>PublisherA_App2: User session established for Service 2 (SSO successful)
    else No Active Session
        Note over User,MonetaCore_Cognito: User would need to authenticate again for Service 2
    end

```


User
Browser
Publisher A - Service 1 (App Client 1)
Publisher A - Service 2 (App Client 2)
Moneta Core (Cognito - User Pool for Publisher A Group)
Accesses Publisher A - Service 1
Request
Redirect to Moneta Core for Login
/oauth2/authorize (client_id=AppClient1)
User authenticates with Moneta Core (e.g., QR Scan via mPass App)
Redirect with Tokens/Code for Service 1
Deliver Tokens/Code
User session established for Service 1
Later, accesses Publisher A - Service 2
Request
Redirect to Moneta Core for Login
/oauth2/authorize (client_id=AppClient2)
Check for active Moneta Core session for this user (in Publisher A Group User Pool)
alt
[Active Session Exists]
Redirect with Tokens/Code for Service 2 (No re-authentication needed)
Deliver Tokens/Code
User session established for Service 2 (SSO successful)
[No Active Session]
User would need to authenticate again for Service 2

Narrative for SSO within a Publisher Group:

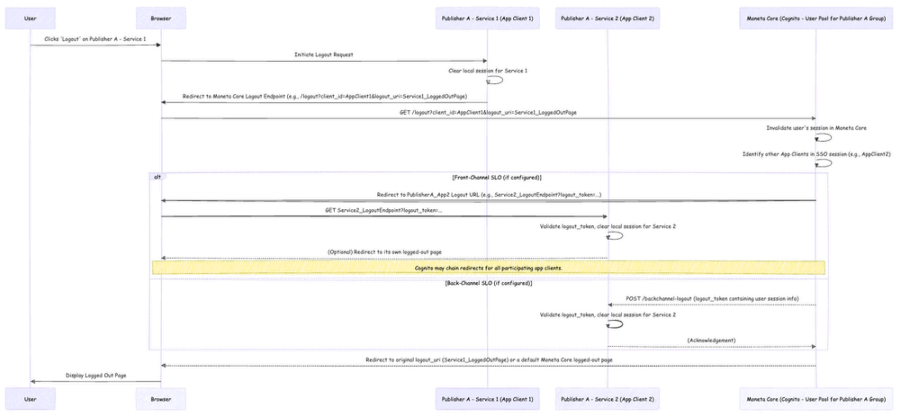
1. **Initial Login to Service 1:** The user authenticates via Moneta Core to access "Publisher A - Service 1" (which is App Client 1 in the shared Cognito User Pool for Publisher A Group). Cognito establishes an active session for the user within this User Pool.
2. **Accessing Service 2:** When the same user subsequently navigates to "Publisher A - Service 2" (App Client 2 in the same User Pool), Service 2 initiates its OIDC login flow, redirecting the user to Moneta Core.
3. **SSO Check by Moneta Core:** Moneta Core (Cognito) receives the authentication request for App Client 2. It checks if an active session already exists for this user within the "Publisher A Group" User Pool.
4. **Seamless Access to Service 2:** Since an active session is found (from the login to Service 1), Moneta Core directly issues the necessary OIDC tokens (or an authorization code) for "Publisher A - Service 2" back to the user's browser. The user is *not* required to repeat the passwordless authentication steps.
5. **Session Established for Service 2:** "Publisher A - Service 2" receives the tokens and establishes its local session for the user, providing a seamless SSO experience across the different services offered by Publisher A.

This SSO capability enhances user convenience by reducing repeated logins when navigating between related services of the same publisher group.

Logout from Publisher Group Services (Single Logout - SLO):

When a user logs out from one service within a Publisher Group that shares an SSO session via Moneta Core (Cognito), the goal is to achieve Single Logout (SLO), terminating the user's session across all participating services in that group and with Moneta Core itself.

```
sequenceDiagram
    participant User
    participant Browser
    participant PublisherA_App1 as Publisher A - Service 1 (App Client 1)
    participant PublisherA_App2 as Publisher A - Service 2 (App Client 2)
    participant MonetaCore_Cognito as Moneta Core (Cognito - User Pool for Publisher A Group)

    User->>Browser: Clicks 'Logout' on Publisher A - Service 1
    Browser->>PublisherA_App1: Initiate Logout Request
    PublisherA_App1->>PublisherA_App1: Clear local session for Service 1
    PublisherA_App1->>Browser: Redirect to Moneta Core Logout Endpoint (e.g., /logout?client_id=AppClient1&logout_uri=Service1_LoggedOutPage)
    Browser->>MonetaCore_Cognito: GET /logout?client_id=AppClient1&logout_uri=Service1_LoggedOutPage
    MonetaCore_Cognito->>MonetaCore_Cognito: Invalidate user's session in Moneta Core
    MonetaCore_Cognito->>MonetaCore_Cognito: Identify other App Clients in SSO session (e.g., AppClient2)

    alt Front-Channel SLO (if configured)
        MonetaCore_Cognito->>Browser: Redirect to PublisherA_App2 Logout URL (e.g., Service2_LogoutEndpoint?logout_token=...)
        Browser->>PublisherA_App2: GET Service2_LogoutEndpoint?logout_token=...
        PublisherA_App2->>PublisherA_App2: Validate logout_token, clear local session for Service 2
        PublisherA_App2-->>Browser: (Optional) Redirect to its own logged-out page
        Note over MonetaCore_Cognito, Browser: Cognito may chain redirects for all participating app clients.
    else Back-Channel SLO (if configured)
        MonetaCore_Cognito-->>PublisherA_App2: POST /backchannel-logout (logout_token containing user session info)
        PublisherA_App2->>PublisherA_App2: Validate logout_token, clear local session for Service 2
        PublisherA_App2-->>MonetaCore_Cognito: (Acknowledgement)
    end

    MonetaCore_Cognito-->>Browser: Redirect to original logout_uri (Service1_LoggedOutPage) or a default Moneta Core logged-out page
    Browser->>User: Display Logged Out Page

```


User
Browser
Publisher A - Service 1 (App Client 1)
Publisher A - Service 2 (App Client 2)
Moneta Core (Cognito - User Pool for Publisher A Group)
Clicks 'Logout' on Publisher A - Service 1
Initiate Logout Request
Clear local session for Service 1
Redirect to Moneta Core Logout Endpoint (e.g., /logout?client_id=AppClient1&logout_uri=Service1_LoggedOutPage)
GET /logout?client_id=AppClient1&logout_uri=Service1_LoggedOutPage
Invalidate user's session in Moneta Core
Identify other App Clients in SSO session (e.g., AppClient2)
alt
[Front-Channel SLO (if configured)]
Redirect to PublisherA_App2 Logout URL (e.g., Service2_LogoutEndpoint?logout_token=...)
GET Service2_LogoutEndpoint?logout_token=...
Validate logout_token, clear local session for Service 2
(Optional) Redirect to its own logged-out page
Cognito may chain redirects for all participating app clients.
[Back-Channel SLO (if configured)]
POST /backchannel-logout (logout_token containing user session info)
Validate logout_token, clear local session for Service 2
(Acknowledgement)
Redirect to original logout_uri (Service1_LoggedOutPage) or a default Moneta Core logged-out page
Display Logged Out Page
User
Browser
Publisher A - Service 1 (App Client 1)
Publisher A - Service 2 (App Client 2)
Moneta Core (Cognito - User Pool for Publisher A Group)

Narrative for Logout from Publisher Group Services (SLO):

1. **User Initiates Logout:** The user clicks a "Logout" button on one of the Publisher's services (e.g., "Publisher A - Service 1").
2. **Local Session Cleanup (Service 1):** "Publisher A - Service 1" clears its own local session information for the user.
3. **Redirect to Moneta Core Logout:** Service 1 redirects the user's browser to the Moneta Core (Cognito) /logout endpoint. This request typically includes the client_id of Service 1 and a logout_uri (a URL on Service 1 where the user should be redirected after logout is complete).
4. **Moneta Core Session Termination:** Moneta Core invalidates its central SSO session for the user.
5. **Propagate Logout to Other Services (SLO):**

- **Front-Channel SLO:** If configured, Moneta Core redirects the user's browser sequentially to pre-configured logout endpoints of other participating Publisher services (e.g., "Publisher A - Service 2"). Each service, upon receiving this request, validates it and clears its local session. This relies on browser redirects and can be less reliable if any step fails.
- **Back-Channel SLO:** If configured, Moneta Core makes direct server-to-server requests to logout endpoints exposed by other participating Publisher services. These requests typically include a logout_token (a JWT) that the services can validate to securely terminate the user's local session. This is generally more reliable.
- *Cognito supports both mechanisms. The choice depends on configuration and Publisher application capabilities.*

1. **Final Redirect:** After attempting to propagate the logout, Moneta Core redirects the user's browser to the logout_uri originally provided by Service 1, or to a default logged-out page if none was specified or if SLO propagation is complete.
2. **User Logged Out:** The user is now logged out from Service 1, Moneta Core, and ideally, all other services within the Publisher Group that were part of the SSO session.

Implementing robust SLO requires careful configuration in Cognito and cooperation from all participating Publisher applications (app clients) to handle logout requests correctly.

Key Lambda Triggers in Cognito:

1. **Define Auth Challenge**
2. **Create Auth Challenge**
3. **Verify Auth Challenge Response**

Session Management, Token Revoke, Refresh Token:

SSO for Publisher Groups: (This higher-level point is now detailed above)

5.2. Human-Friendly Identifiers, Tiers, Billing & Lifecycle

Identifier Schemes:

- MIC (Moneta Issuer Code)
- MPC (Moneta Partner Code)
- mPass Number (MII-MAI-VC)

Data Storage (DynamoDB):

- **MOTable:**
  - moID (PK, UUIDv4)
  - mic (GSI PK - Moneta Issuer Code, for lookup by human-friendly code)
  - status (String: Pending, Active, Suspended, Destroyed)
  - name (String, official name of the MO)
  - createdAt (Timestamp)
  - updatedAt (Timestamp)
  - metadata (JSON blob for MO-specific details, e.g., contact info, webhook endpoints, configuration)
- **PublishersTable:**
  - publisherID (PK, UUIDv4)
  - mpc (GSI PK - Moneta Partner Code, for lookup by human-friendly code)
  - status (String: Pending, Active, Suspended, Destroyed)
  - name (String, official name of the Publisher)
  - ssoGroupID (String, optional, identifier for SSO grouping if applicable)
  - createdAt (Timestamp)
  - updatedAt (Timestamp)
  - metadata (JSON blob for Publisher-specific details, e.g., service endpoints, categories, logo URL)
- **MPassTable:**
  - mPassID (PK, UUIDv4)
  - mPassNumber (GSI PK - Human-friendly mPass number, for lookup)
  - moID (GSI SK for moID-mPassNumber index - String, UUIDv4 of the issuing MO, FK to MOTable.moID)
  - userID_internal (String, MO's internal user identifier for this mPass holder)
  - status (String: Pending, Active, Suspended_UserLock, Suspended_AdminLock, Expired, Destroyed)
  - tier (String, e.g., "Standard", "Platinum", "Gold")
  - billingAccountID (GSI SK for billingAccountID-mPassID index - String, FK to BillingAccountsTable.billingAccountID)
  - activePublicKeyID (String, UUIDv4, FK to MPassPublicKeysTable.keyID for the currently active key)
  - createdAt (Timestamp)
  - updatedAt (Timestamp)
  - expiresAt (Timestamp)
  - metadata (JSON blob for mPass-specific details)
- **MPassPublicKeysTable:**
  - mPassID (PK, UUIDv4, FK to MPassTable.mPassID)
  - keyID (SK, UUIDv4 for the key itself)
  - publicKeyData (String, Base64 encoded public key)
  - algorithm (String, e.g., ECDSA_P256, ED25519)
  - status (String: Active, Inactive, Superseded, Revoked)
  - createdAt (Timestamp)
  - activatedAt (Timestamp)
  - deactivatedAt (Timestamp, optional)
- **BillingAccountsTable:**
  - billingAccountID (PK, UUIDv4)
  - moID (GSI PK for moID-accountType index - String, UUIDv4 of the MO managing this billing account, FK to MOTable.moID)
  - accountType (String: "Individual", "Company", "Family")
  - ownerEntityID (String, UUIDv4 - For Company, this could be a company identifier (e.g., publisherID if a publisher is the account owner) or a dedicated company entity ID. For Family or Individual, this is the mPassID of the owner from MPassTable.)
  - status (String: "Active", "Suspended", "Closed")
  - paymentResponsibilityInfo (JSON blob, e.g., details of who pays for Company/Family)
  - policySettings (JSON blob, for Family accounts, defines policies set by owner mPass for other mPasses in the account, e.g., spending limits, category restrictions)
  - createdAt (Timestamp)
  - updatedAt (Timestamp)
  - *Note: mPasses are linked to this table via MPassTable.billingAccountID. A GSI on MPassTable using billingAccountID as PK and mPassID as SK can find all mPasses for a billing account.*

Lifecycle Management:

- **mPass Lifecycle:** (Existing states remain, managed within MPassTable)
  - Tier assignment happens at issuance or through an upgrade process (updates MPassTable.tier).
  - Billing account association happens at issuance (updates MPassTable.billingAccountID).
- **mPass Tier Management:**
  - **Assignment:** Tiers are typically assigned by the MO during mPass issuance based on MO's product offerings or user eligibility.

- **Information:** Tier information can be used by Moneta Core and Publishers (after fetching via an authorized API call) to apply specific policies, limits, or offer differentiated services.
- **Business Flow: Tier Upgrade Process:**
  1. **User Request:** User requests a tier upgrade via the mPass mobile application or MO's portal.
  2. **MO Review & Approval:** MO reviews the request based on their criteria (e.g., usage history, payment record, new fees).
  3. **MO Action:** MO administrator uses their interface (interacting with Moneta Core APIs) to approve and update the mPass tier.
  4. **Moneta Core Update:** `mPassTable` record for the mPass is updated with the new `tier`.
  5. **Notification:** User is notified of the tier change via mPass app/email.
  6. **Policy Application:** New policies associated with the upgraded tier take effect.

- **Billing Account Management:**
  - **Individual Account:**
    - Default type. The `billingAccountID` on the mPass essentially links to the individual user's financial relationship with the MO.
    - The `ownerEntityID` in `BillingAccountsTable` would typically be the mPass's `entityID`.
  - **Company Account:**
    - A single `BillingAccountsTable` record represents the company's billing entity.
    - Multiple mPasses (from `EntitiesTable`) will share the same `billingAccountID`, linking them to this company account.
    - The `ownerEntityID` in `BillingAccountsTable` would be an identifier for the company itself.
    - The MO manages payment processing with the company for all charges accumulated under this account.
  - **Family Account:**
    - A primary mPass holder (the "owner") establishes a Family account.
    - The `BillingAccountsTable` record will have `accountType: "Family"`, and `ownerEntityID` will be the `entityID` of the owner's mPass.
    - Other family members' mPasses are linked to this `billingAccountID`.
    - The owner mPass holder is responsible for all payments accumulated by mPasses under this account.
    - The owner can set policies (e.g., spending limits per mPass, allowed merchant categories) for other mPasses in the family account. These policies are stored in `BillingAccountsTable.policySettings` and enforced by Moneta Core and/or communicated to Publishers.
  - **Association:** MOs manage the association of mPasses to billing accounts via Moneta Core APIs, typically during mPass issuance or through specific account management features offered by the MO.

5.3. Cross-Cutting Technical Requirements

- **Tech Stack (AWS):**
  - Cognito, Lambda, DynamoDB, API Gateway (core).
  - S3 (for Cognito Hosted UI customizations, static assets).
  - SNS (for push notifications to trigger OTP delivery, or for eventing).
  - CloudFront (CDN for Hosted UI and API Gateway for performance and caching).
  - AWS WAF (security for CloudFront/API Gateway).
  - Amazon EventBridge (for data synchronization).
  - KMS (for enhanced encryption of sensitive data in DynamoDB if needed).
- **Low Authentication Latency:**
  - **Lambda:** Provisioned Concurrency for critical auth Lambdas. Optimized code (Node.js or Python). Minimize package sizes.
  - **DynamoDB:** Proper indexing, use DAX (DynamoDB Accelerator) if read-heavy for common data. Global Tables for multi-region reads if necessary.
  - **API Gateway:** Edge-optimized endpoints.
  - **Cognito:** Generally performant; ensure user pool is in regions close to users.
  - **Network:** CloudFront to serve content closer to users.
- **Data Synchronization (Auth Module to other Moneta Core Microservices):**
  a. **DynamoDB Streams:** Enable streams on `EntitiesTable` and `MPassPublicKeysTable`.
  b. **Stream Processor Lambda:** A Lambda function subscribes to these streams.
  c. **Amazon EventBridge:** The Lambda processes stream records and publishes domain events (e.g., `MPassActivated`, `MOLifecycleChanged`, `PublisherUpdated`) to a custom EventBridge event bus.
  d. **Subscribing Microservices:** Other Moneta Core microservices subscribe to relevant events on this bus via their own Lambdas or other integrations, updating their local data stores or triggering workflows.
- **Recommended Lambda Development Tech Stack:**
  - **Runtime: Node.js (LTS)** or **Python (latest supported)**. Both are excellent for serverless. Node.js often preferred for API-driven, I/O-bound tasks due to its event loop model and potentially faster cold starts for simple functions.
  - **IaC: AWS SAM (Serverless Application Model)** or **Serverless Framework**.
  - **Testing:** Jest (Node.js) / Pytest (Python) for unit tests. Tools like `moto` (Python) or localstack for mocking AWS services during integration tests.
  - **SDK:** AWS SDK (v3 for Node.js for modularity).
- **Recommended CI/CD Tool and Method:**
  - **Tools: AWS CodePipeline** (orchestration) with **AWS CodeCommit** (source), **AWS CodeBuild** (build/test), **AWS CodeDeploy** (deploy Lambdas/SAM applications). Alternatives: GitHub Actions, GitLab CI.
  - **Method:**
    1. Git-based workflow (e.g., Gitflow, Trunk-Based Development).
    2. Automated builds, linting, unit tests on every commit/PR.
    3. Separate Staging/QA environment mirroring production.
    4. Automated integration and end-to-end tests in Staging.

    v. Deployment to Production using **Canary Releases** or **Blue/Green Deployments** for Lambdas (supported by SAM/CodeDeploy).
    vi. Automated rollbacks based on CloudWatch alarms or deployment failures.

- **Extensibility for Future Authentication Methods:**
  - The Cognito Custom Authentication Flow is inherently extensible.
  - New methods (e.g., FIDO2/WebAuthn, device biometrics relayed by mPass app) can be added by:
    i. Updating the `Define Auth Challenge` Lambda to recognize and sequence the new method.
    ii. Implementing new logic in `Create Auth Challenge` and `Verify Auth Challenge Response` Lambdas for the new method.
    iii. Potentially adding new API Gateway endpoints if the mPass app interaction model differs.
- **DynamoDB Streams:** Enable streams on MOTable, PublishersTable, MPassTable, MPassPublicKeysTable, and the BillingAccountsTable.
- **Stream Processor Lambda:** A Lambda function subscribes to these streams.
- **Amazon EventBridge:** The Lambda processes stream records and publishes domain events (e.g., MPassTierChanged, MPassStatusChanged, MOCreated, PublisherUpdated, BillingAccountCreated, FamilyPolicyUpdated) to a custom EventBridge event bus.
- **Subscribing Microservices:** Other Moneta Core microservices subscribe to relevant events on this bus via their own Lambdas or other integrations, updating their local data stores or triggering workflows.

5.4. Security Assessment Summary

- **Overall Solution:**
  - **Strengths:** Leverages robust OIDC standard, AWS security features. Passwordless reduces attack vectors. Private keys on device.
  - **Weaknesses:** Complexity can introduce vulnerabilities if not carefully implemented. Heavy reliance on mobile app security. Moneta Core auth is a critical target.
- **QR Code Authentication:**
  - **Pros:** Good UX, contactless.
  - **Cons/Risks:** Shoulder surfing for QR, QR replay (if not single-use/short-lived), man-in-the-middle for API calls from app.
  - **Mitigations:** Short-lived signed challenges, HTTPS, mPass app security.
- **OTP to mPass App Authentication:**
  - **Pros:** No camera needed, familiar.
  - **Cons/Risks:** OTP interception (phishing, compromised device notifications), push notification failures.
  - **Mitigations:** Short-lived OTPs, rate limiting, secure channel for OTP display in-app, user education.
- **Human-Friendly Identifiers:**
  - **Pros:** Usability.
  - **Cons/Risks:** Enumeration if predictable. Management complexity.
  - **Mitigations:** Non-sequential components where possible, strong validation, rate limiting on lookups.
- **mPass Key Management:**
  - **Pros:** User control of private keys.
  - **Cons/Risks:** Device compromise, loss of device/keys.
  - **Mitigations:** Hardware-backed key storage on mobile, secure recovery mechanisms, user education.
- **Tier and Billing Management Security:**
  - **Pros:** Allows granular control over financial aspects and service levels.
  - **Cons/Risks:** Unauthorized modification of tiers or billing account details. Misconfiguration of family account policies leading to unintended consequences.
  - **Mitigations:** Role-Based Access Control (RBAC) for APIs managing tiers and billing. Strong authentication for MO admins and family account owners. Detailed audit logs for all changes to tiers, billing accounts, and policies. Clear UI/UX for policy settings in family accounts.

6. Open Issues / Next Steps

- Detailed design of the mPass app interaction for key generation, secure storage, **and display/management of tier/billing information.**
- Finalize the exact structure and character sets for MIC, MPC, and MII.
- Develop a detailed recovery flow for users who lose access to their mPass app/device.
- Perform a thorough threat modeling exercise on the chosen authentication flows, **as well as tier and billing management processes.**
- Prototype the Cognito custom authentication flow with one method (e.g., QR code).
- **Define the specific attributes and structure for policySettings within the BillingAccountsTable for Family accounts.**
- **Design APIs and internal logic for MOs and Family account owners to manage tiers and billing account policies.**
- **Specify how tier/billing information is made available to Publishers (e.g., as claims in tokens, or via a separate resource API call) and how policies are enforced at the transaction level.**
- **Consider data migration strategies if existing mPasses need to be mapped to new tier/billing structures.**
- **Refine GSI strategies for the new tables based on detailed access patterns.**

7. Story Definition

This section lists user stories derived from the "2. Main Use Cases" section, intended for backlog management.

7.1. Authentication Flow Stories

- **Auth-Story-001:** As an mPass User, I want to securely authenticate to a Publisher's service using a QR code scan via my mPass mobile app, so that I can access the Publisher's content or services.
- **Auth-Story-002:** As an mPass User, I want to securely authenticate to a Publisher's service by receiving and confirming an OTP on my mPass mobile app, so that I can access the Publisher's content or services.

- **Auth-Story-003:** As an mPass User, I want to log out from a Publisher's service, so that my session is terminated and my access is revoked. (This story should also cover SLO implications for Publisher groups).
- **Auth-Story-004:** As a Publisher System, I want to refresh an mPass User's access token using a refresh token, so that the user can maintain their session without re-authenticating.
- **Auth-Story-005:** As an mPass User, I want to experience Single Sign-On (SSO) when accessing different services from Publishers within the same group, so that I don't have to log in repeatedly.
- **Auth-Story-006:** As a Moneta Core System, I want to manage the lifecycle of user sessions (creation, validation, expiry, termination), so that user access is secure and controlled.
- **Auth-Story-007:** As a Publisher System, I want to receive OIDC ID and Access tokens after a successful user authentication, so that I can identify the user and authorize access to my resources.
- **Auth-Story-008:** As a Publisher System (Resource Server), I want to validate an mPass Access Token presented by a client, so that I can ensure the request is authorized.

7.2. Partner (MO & Publisher) Management Stories

- **Partner-Story-001:** As a Moneta Core Admin, I want to onboard a new Membership Organization (MO), providing all necessary details and configurations (e.g., assigning MIC), so that the MO can operate on the Moneta Network.
- **Partner-Story-002:** As a Moneta Core Admin, I want to view the details and current status of any MO on the network, so that I can monitor and manage them effectively.
- **Partner-Story-003:** As a Moneta Core Admin, I want to update the configuration of an existing MO, so that their operational parameters are current.
- **Partner-Story-004:** As a Moneta Core Admin, I want to manage the lifecycle of an MO (e.g., move to Pending, Active, Suspended, Destroyed status), so that the network reflects their operational state.
- **Partner-Story-005:** As a Moneta Core Admin, I want to onboard a new Publisher, providing all necessary details and configurations (e.g., assigning MPC, defining service endpoints), so that the Publisher can offer services on the Moneta Network.
- **Partner-Story-006:** As a Moneta Core Admin, I want to view the details and current status of any Publisher on the network, so that I can monitor and manage them effectively.
- **Partner-Story-007:** As a Moneta Core Admin, I want to update the configuration of an existing Publisher, so that their operational parameters are current.
- **Partner-Story-008:** As a Moneta Core Admin, I want to manage the lifecycle of a Publisher (e.g., move to Pending, Active, Suspended, Destroyed status), so that the network reflects their operational state.
- **Partner-Story-009:** As a Moneta Core Admin, I want to associate multiple Publisher app clients with the same Cognito User Pool, so that SSO can be enabled for that publisher group.

7.3. mPass Management Stories

- **mPass-Story-001:** As an MO Admin, I want to issue a new mPass to a user, including assigning an mPass Number, initial tier, and associating it with a billing account, so that the user can utilize Moneta Network services.
- **mPass-Story-002:** As an mPass User, I want to view the details of my mPass (e.g., mPass Number, tier, status) via my mPass mobile app.
- **mPass-Story-003:** As an MO Admin, I want to view the details of any mPass issued by my MO, so that I can provide support and manage accounts.
- **mPass-Story-004:** As an mPass User, I want to lock my mPass via my mobile app, so that it cannot be used for transactions if my device is compromised or lost.
- **mPass-Story-005:** As an MO Admin, I want to suspend an mPass (e.g., due to reported fraud or non-payment), so that it cannot be used.
- **mPass-Story-006:** As an MO Admin, I want to destroy an mPass (e.g., at user request or account closure), so that it is permanently deactivated.
- **mPass-Story-007:** As a Moneta Core System, I want to automatically manage mPass expiration by changing its status when the expiration date is reached.
- **mPass-Story-008:** As an mPass User, I want to transfer my mPass to a new mobile device, including generating new key pairs, so that I can continue using my mPass securely.
- **mPass-Story-009:** As an mPass User, I want to recover my mPass on a new device if my old device is broken or lost, including generating new key pairs, so that I regain access to my mPass.
- **mPass-Story-010:** As an MO Admin, I want to assign or change the tier of an mPass (e.g., Standard, Platinum), so that appropriate policies and benefits are applied.
- **mPass-Story-011:** As an MO Admin, I want to associate an mPass with an Individual Billing Account, so that the user is responsible for their payments.
- **mPass-Story-012:** As an MO Admin, I want to associate multiple mPasses with a Company Billing Account, so that the company can manage payments centrally.
- **mPass-Story-013:** As an MO Admin, I want to set up a Family Billing Account, designating an owner mPass and linking other family member mPasses to it.
- **mPass-Story-014:** As an mPass User (Family Account Owner), I want to set and manage policies (e.g., spending limits, category restrictions) for other mPasses within my Family Billing Account via my mPass mobile app.
- **mPass-Story-015:** As a Moneta Core System, I want to store and manage one active public key and a history of inactive public keys for each mPass, along with the signing algorithm used.
- **mPass-Story-016:** As a Publisher System, I want to be able to retrieve mPass details (like tier and applicable limits) from Moneta Core using an access token, so that I can apply relevant business rules.