API Authentication

The Oracle App Backend uses HMAC-SHA256 signature-based authentication for both REST APIs and WebSocket connections.

Configuration

Backend Setup

Add the following environment variables to your `.env` file:

```
# Format: apiKey1:secret1,apiKey2:secret2
AUTH_API_KEYS=client1:mySecretKey123,client2:anotherSecret456

# Maximum allowed time difference (milliseconds)
AUTH_TIMESTAMP_SKEW_MS=30000
```

Important: - Leave `AUTH_API_KEYS` empty to disable authentication entirely - Each API key should be paired with a secret using the format `key:secret` - Multiple key-secret pairs are separated by commas - Keep secrets secure and never commit them to version control

Generating API Keys and Secrets

```
# Generate a random API key
openssl rand -hex 16

# Generate a random secret
openssl rand -base64 32
```

Frontend Implementation

REST API Authentication

All REST endpoints under `/api` require authentication when enabled.

Required Headers

```
x-api-key: YOUR_API_KEY
x-signature: COMPUTED_HMAC_SIGNATURE
x-timestamp: CURRENT_TIMESTAMP_MS
```

Signature Computation The signature is computed as follows:

```
payload = METHOD + PATH + TIMESTAMP + SHA256(BODY)
signature = HMAC-SHA256(payload, SECRET)
```

Example:

```
Method: GET
Path: /api/assets/btc-usd
Timestamp: 1737291600000
Body: (empty for GET)

payload = "GET/api/assets/btc-usd1737291600000e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b93
signature = HMAC-SHA256(payload, "mySecretKey123")
```

JavaScript/TypeScript Example

```typescript
import crypto from 'crypto';

async function makeAuthenticatedRequest(
  apiKey: string,
  secret: string,
  method: string,
  path: string,
  body?: any
): Promise<Response> {
  const timestamp = Date.now().toString();
  const bodyString = body ? JSON.stringify(body) : '';

  // Compute body hash
  const bodyHash = crypto
    .createHash('sha256')
    .update(bodyString)
    .digest('hex');

  // Build signature payload
  const payload = `${method.toUpperCase()}${path}${timestamp}${bodyHash}`;

  // Compute HMAC signature
  const signature = crypto
    .createHmac('sha256', secret)
    .update(payload)
    .digest('hex');

  // Make request
  const response = await fetch(`https://api.example.com${path}`, {
    method,
    headers: {
      'Content-Type': 'application/json',
      'x-api-key': apiKey,
      'x-signature': signature,
      'x-timestamp': timestamp,
    },
```

```
    body: bodyString || undefined,
  });

  return response;
}

// Usage
const response = await makeAuthenticatedRequest(
  'client1',
  'mySecretKey123',
  'GET',
  '/api/assets/btc-usd'
);
```

Browser Example (using Web Crypto API)

```
async function sha256(message) {
  const msgBuffer = new TextEncoder().encode(message);
  const hashBuffer = await crypto.subtle.digest('SHA-256', msgBuffer);
  const hashArray = Array.from(new Uint8Array(hashBuffer));
  return hashArray.map(b => b.toString(16).padStart(2, '0')).join('');
}

async function hmacSha256(secret, message) {
  const enc = new TextEncoder();
  const key = await crypto.subtle.importKey(
    'raw',
    enc.encode(secret),
    { name: 'HMAC', hash: 'SHA-256' },
    false,
    ['sign']
  );
  const signature = await crypto.subtle.sign('HMAC', key, enc.encode(message));
  const signatureArray = Array.from(new Uint8Array(signature));
  return signatureArray.map(b => b.toString(16).padStart(2, '0')).join('');
}

async function makeAuthenticatedRequest(apiKey, secret, method, path, body = null) {
  const timestamp = Date.now().toString();
  const bodyString = body ? JSON.stringify(body) : '';

  // Compute body hash
  const bodyHash = await sha256(bodyString);

  // Build signature payload
  const payload = `${method.toUpperCase()}${path}${timestamp}${bodyHash}`;
```

```
  // Compute HMAC signature
  const signature = await hmacSha256(secret, payload);

  // Make request
  const response = await fetch(`https://api.example.com${path}`, {
    method,
    headers: {
      'Content-Type': 'application/json',
      'x-api-key': apiKey,
      'x-signature': signature,
      'x-timestamp': timestamp,
    },
    body: bodyString || undefined,
  });

  return response;
}
```

WebSocket Authentication

WebSocket connections to `/api/ws/prices` and `/api/ws/price` require authentication via query parameters.

Required Query Parameters

```
apiKey (or key): YOUR_API_KEY
signature (or sig): COMPUTED_HMAC_SIGNATURE
timestamp (or ts): CURRENT_TIMESTAMP_MS
```

Signature Computation for WebSocket

```
payload = METHOD + PATH + TIMESTAMP + SHA256("")
signature = HMAC-SHA256(payload, SECRET)
```

Note: WebSocket upgrades always use `GET` method and empty body.

Example:

```
Method: GET
Path: /api/ws/price
Timestamp: 1737291600000
Body: (empty)

payload = "GET/api/ws/price1737291600000e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495
signature = HMAC-SHA256(payload, "mySecretKey123")
```

JavaScript/TypeScript Example

```typescript
import crypto from 'crypto';

function createAuthenticatedWebSocket(
  apiKey: string,
  secret: string,
  path: string,
  additionalParams?: Record<string, string>
): WebSocket {
  const timestamp = Date.now().toString();

  // Compute empty body hash
  const bodyHash = crypto
    .createHash('sha256')
    .update('')
    .digest('hex');

  // Build signature payload (always GET for WebSocket)
  const payload = `GET${path}${timestamp}${bodyHash}`;

  // Compute HMAC signature
  const signature = crypto
    .createHmac('sha256', secret)
    .update(payload)
    .digest('hex');

  // Build query string
  const params = new URLSearchParams({
    apiKey,
    signature,
    timestamp,
    ...additionalParams,
  });

  // Create WebSocket connection
  const ws = new WebSocket(`wss://api.example.com${path}?${params}`);

  return ws;
}

// Usage - broadcast stream
const broadcastWs = createAuthenticatedWebSocket(
  'client1',
  'mySecretKey123',
  '/api/ws/prices'
```

```
);

// Usage - asset-specific stream
const assetWs = createAuthenticatedWebSocket(
  'client1',
  'mySecretKey123',
  '/api/ws/price',
  { assetId: 'btc-usd', frequency: '2000' }
);
```

Browser Example

```
async function createAuthenticatedWebSocket(apiKey, secret, path, additionalParams = {}) {
  const timestamp = Date.now().toString();

  // Compute empty body hash
  const bodyHash = await sha256('');

  // Build signature payload (always GET for WebSocket)
  const payload = `GET${path}${timestamp}${bodyHash}`;

  // Compute HMAC signature
  const signature = await hmacSha256(secret, payload);

  // Build query string
  const params = new URLSearchParams({
    apiKey,
    signature,
    timestamp,
    ...additionalParams,
  });

  // Create WebSocket connection
  const ws = new WebSocket(`wss://api.example.com${path}?${params}`);

  return ws;
}

// Usage
const ws = await createAuthenticatedWebSocket(
  'client1',
  'mySecretKey123',
  '/api/ws/price',
  { assetId: 'btc-usd', frequency: '1000' }
);
```

```
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  console.log('Received:', data);
};
```

Security Considerations

1. **Timestamp Replay Protection**: Each request timestamp must be unique and recent. The server rejects timestamps that:
   - Are older/newer than `AUTH_TIMESTAMP_SKEW_MS` (default 30 seconds)
   - Have already been seen for the same API key (replay attack prevention)
2. **Secure Storage**: Store API keys and secrets securely:
   - Use environment variables or secure vaults
   - Never hardcode credentials in source code
   - Never commit secrets to version control
3. **HTTPS/WSS**: Always use secure connections in production:
   - Use `https://` for REST APIs
   - Use `wss://` for WebSocket connections
4. **Key Rotation**: Periodically rotate API keys and secrets:
   - Generate new key-secret pairs
   - Update backend configuration
   - Distribute new credentials to clients
   - Remove old credentials after migration
5. **Rate Limiting**: Implement additional rate limiting at the infrastructure level (e.g., nginx, cloud load balancer)

Error Responses

REST API Errors

```
{
  "message": "Missing API key"
}
```

Status Code: `401 Unauthorized`

Common error messages: - `"Missing API key"` - `"Unknown API key"` - `"Missing signature"` - `"Missing timestamp"` - `"Invalid timestamp"` - `"Timestamp outside allowable window"` - `"Invalid signature"` - `"Replay detected"`

WebSocket Errors

When authentication fails, the server sends an HTTP 401 response during the upgrade:

```
HTTP/1.1 401 Unauthorized
```

```
Content-Type: application/json
Connection: close

{"message":"Invalid signature"}
```

The WebSocket connection will not be established.

Testing Authentication

Disable Authentication (Development)

To disable authentication during development, leave `AUTH_API_KEYS` empty in your `.env`:

```
AUTH_API_KEYS=
```

Test with cURL

```bash
# Generate timestamp
TIMESTAMP=$(date +%s)000

# Compute body hash (empty for GET)
BODY_HASH=$(echo -n "" | sha256sum | cut -d' ' -f1)

# Compute signature
PAYLOAD="GET/api/assets/btc-usd${TIMESTAMP}${BODY_HASH}"
SIGNATURE=$(echo -n "$PAYLOAD" | openssl dgst -sha256 -hmac "mySecretKey123" | cut -d' ' -f2

# Make request
curl -X GET "https://api.example.com/api/assets/btc-usd" \
  -H "x-api-key: client1" \
  -H "x-signature: $SIGNATURE" \
  -H "x-timestamp: $TIMESTAMP"
```

API Endpoints

Protected Endpoints (require authentication when enabled)

- `GET /api/overview` - System overview
- `GET /api/asset-classes` - List asset classes
- `GET /api/assets` - List assets (paginated)
- `GET /api/assets/:id` - Get asset details
- `GET /api/assets/:id/history` - Get price history
- `WS /api/ws/prices` - Broadcast price stream
- `WS /api/ws/price` - Asset-specific price stream

Public Endpoints (always accessible)

- `GET /health` - Health check

Support

For issues or questions about authentication: 1. Check that your timestamp is within the allowed skew window 2. Verify the signature computation follows the exact format: `METHOD + PATH + TIMESTAMP + BODY_HASH` 3. Ensure the path used in signature matches the request path exactly 4. Confirm your API key and secret are correctly configured on the backend