# Building Face ID and Touch ID Authentication with Amazon Cognito

Passwords are dead. Phishing attacks exploit weak credentials in 80% of breaches, and for regulated industries like healthcare and finance, that's not just a security issue—it's a compliance nightmare.

What if users could authenticate with just their face or fingerprint? With Amazon Cognito and WebAuthn, you can build passwordless authentication that works seamlessly with Apple's Face ID and Touch ID. No more forgotten passwords, no more credential theft, just secure biometric authentication that meets HIPAA and PCI-DSS requirements.

This guide shows you how to integrate Face ID and Touch ID with Cognito User Pools using custom authentication flows and WebAuthn. We'll build a complete working example that you can deploy and test on your Mac.

## Why Face ID and Touch ID Matter

Biometric authentication eliminates the weakest link in security: passwords. Face ID and Touch ID are "platform authenticators" that use WebAuthn standards, keeping biometric data on-device while providing phishing-resistant authentication.

For regulated workloads, this means:

**HIPAA Compliance**- : Strong authentication for PHI access

**PCI-DSS Requirements**- : Multi-factor authentication without shared secrets

**Zero Trust**- : Device-bound credentials that can't be stolen

**Better UX**- : Sub-3-second authentication with no typing

## Architecture

![Architecture Diagram](./generated-diagrams/faceid-cognito-architecture-clean.png)

The solution uses Cognito's custom authentication flow with three Lambda triggers:

**DefineAuthChallenge**1. : Determines the authentication flow

**CreateAuthChallenge**2. : Generates WebAuthn challenges

**VerifyAuthChallengeResponse**3. : Validates biometric responses

WebAuthn handles the biometric prompts, while Cognito manages user sessions and JWT tokens.

## Implementation

### 1. Cognito User Pool Setup

Create a User Pool with custom authentication enabled:

aws cognito-idp create-user-pool \

--pool-name "FaceIDDemo" \

--username-attributes email \

--explicit-auth-flows ALLOW\_CUSTOM\_AUTH

### 2. Lambda Triggers

Create three Lambda functions for the custom auth flow:

**DefineAuthChallenge**: Controls the authentication flow

exports.handler = async (event) => {

if (event.request.session.length === 0) {

event.response.challengeName = 'CUSTOM\_CHALLENGE';

event.response.issueTokens = false;

} else if (event.request.session[0].challengeResult === true) {

event.response.issueTokens = true;

} else {

event.response.failAuthentication = true;

}

return event;

};

**CreateAuthChallenge**: Generates WebAuthn challenges

const crypto = require('crypto');

exports.handler = async (event) => {

if (event.request.challengeName === 'CUSTOM\_CHALLENGE') {

const challenge = crypto.randomBytes(32).toString('base64url');

event.response.publicChallengeParameters = {

challenge,

rpId: 'localhost',

userVerification: 'required'

};

event.response.privateChallengeParameters = { challenge };

}

return event;

};

**VerifyAuthChallengeResponse**: Validates biometric responses

exports.handler = async (event) => {

const expectedChallenge = event.request.privateChallengeParameters?.challenge;

const credential = JSON.parse(event.request.challengeAnswer);

// Verify WebAuthn response

const clientData = JSON.parse(

Buffer.from(credential.response.clientDataJSON, 'base64url')

);

event.response.answerCorrect =

clientData.challenge === expectedChallenge &&

clientData.type === 'webauthn.get';

return event;

};

### 3. Client-Side WebAuthn Integration

The client uses WebAuthn to trigger Face ID/Touch ID prompts:

// Registration: Create biometric credential

const credential = await navigator.credentials.create({

publicKey: {

challenge: challengeBuffer,

rp: { id: 'localhost', name: 'Face ID Demo' },

user: { id: userIdBuffer, name: username, displayName: username },

authenticatorSelection: {

authenticatorAttachment: 'platform',

userVerification: 'required' // Triggers Face ID/Touch ID

}

}

});

// Authentication: Use stored credential

const assertion = await navigator.credentials.get({

publicKey: {

challenge: challengeBuffer,

allowCredentials: [{ id: credentialId, type: 'public-key' }],

userVerification: 'required' // Triggers Face ID/Touch ID

}

});

userVerification: 'required'authenticatorAttachment: 'platform'The key is setting and to ensure biometric authentication.

### 4. Testing

Requirements:

- macOS with Touch ID or Face ID

- Safari browser (best compatibility)

- HTTPS (required for WebAuthn)

Test flow:

1. Register with email/username

2. Complete biometric enrollment

3. Sign in with Face ID/Touch ID

### 5. Production Considerations

- Replace localStorage with DynamoDB

rpId- Update to your domain

- Enable Cognito advanced security

- Add proper public key verification

- Set up CloudTrail for audit logs

## Key Benefits

**Security**- : Eliminates password vulnerabilities and phishing attacks

**Compliance**- : Meets HIPAA, PCI-DSS requirements with strong authentication

**User Experience**- : Sub-3-second authentication with familiar biometric prompts

**Privacy**- : Biometric data never leaves the device

## Best Practices

- Use DynamoDB for credential storage in production

- Implement fallback authentication methods

- Enable comprehensive audit logging

- Test across different Apple devices

- Monitor authentication success rates

## Conclusion

Face ID and Touch ID authentication with Cognito provides enterprise-grade security without compromising user experience. The WebAuthn standard ensures compatibility while keeping biometric data secure on-device.

This approach is particularly valuable for regulated industries where strong authentication is required but user friction must be minimized. The combination of AWS's robust infrastructure with Apple's biometric technologies creates a powerful, compliant authentication solution.

Check out the [complete implementation](https://github.com/your-repo) for full source code and deployment instructions.