Multi-Factor Authentication Implementation

Comprehensive guide to MFA methods and PSD2 SCA compliance

Version: 1.0

Author: Security Engineering Team

Category: Security | Authentication | PSD2 Compliance

Compliance: PSD2, FCA SYSC, PCI-DSS

Last Updated: December 12, 2025

This guide provides comprehensive implementation details for Monzo Multi-Factor Authentication (MFA) system. MFA is mandatory for all high-risk operations including payment authorizations above 30 EUR (PSD2 SCA requirement), account modifications, and API access token generation. This document covers TOTP, SMS, push notification, and biometric authentication methods.

Regulatory Requirements

PSD2 Strong Customer Authentication (SCA)

EU Payment Services Directive 2 (PSD2) mandates Strong Customer Authentication for electronic payments, requiring two independent elements from:

Knowledge: Something the user knows (password, PIN)

Possession: Something the user has (phone, hardware token)

Inherence: Something the user is (fingerprint, face recognition)

Scope: SCA applies to payments exceeding 30 EUR, account access, and actions implying fraud/security risk

Exemptions: Recurring payments, trusted beneficiaries, low-risk transactions (under 30 EUR with daily aggregate limit 100 EUR)

FCA Requirements

Financial Conduct Authority (FCA) SYSC requirements for authentication:

• Multi-factor authentication for remote banking access

• Transaction signing for high-value payments

• Risk-based authentication adjusting security based on context

• Secure authentication credential storage (hashed, encrypted)

MFA Methods Supported

Method	Security Level	User Experience	Use Case
TOTP (Authenticator)	High	Medium	API access, account recovery
SMS OTP	Medium	High	Payment confirmation, login
Push Notification	High	Very High	Payment approval, account changes
Biometric (Touch/Face)	Very High	Very High	App login, quick payments
Hardware Token	Very High	Low	Corporate accounts, high-value

TOTP Implementation

Time-Based One-Time Password Setup

TOTP generates 6-digit codes that rotate every 30 seconds using HMAC-SHA1:

```
import pyotp
import qrcode
from io import BytesIO

def setup_totp_for_user(user_id):
    """
    Generate TOTP secret and QR code for user.
    """
    # Generate random 32-character base32 secret
    secret = pyotp.random_base32()

    # Create TOTP URI for authenticator apps
    totp_uri = pyotp.totp.TOTP(secret).provisioning_uri(
        name=user_id,
        issuer_name='Monzo'
    )

    # Generate QR code
    qr = qrcode.QRCode(version=1, box_size=10, border=4)
    qr.add_data(totp_uri)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")

    # Save secret to database (encrypted!)
    save_totp_secret(user_id, encrypt(secret))

    return {
        'secret': secret,
        'qr_code': img,
        'uri': totp_uri
    }
```

TOTP Verification

```
def verify_totp(user_id, code):
    """
    Verify TOTP code with time window tolerance.
    """
    # Retrieve encrypted secret from database
    encrypted_secret = get_totp_secret(user_id)
    secret = decrypt(encrypted_secret)

    # Create TOTP object
    totp = pyotp.TOTP(secret)

    # Verify with 1-window tolerance (90 seconds)
    # Allows for clock drift
    valid = totp.verify(code, valid_window=1)

    if valid:
        # Log successful authentication
        log_mfa_success(user_id, 'totp')
        return True
    else:
        # Log failed attempt
        log_mfa_failure(user_id, 'totp')
        increment_failed_attempts(user_id)
        return False
```

SECURITY: TOTP secrets must be encrypted at rest using AES-256. Never store secrets in plaintext. Use separate encryption keys per environment.

SMS OTP Implementation

Generate and Send SMS Code

```
import random
import hashlib
from datetime import datetime, timedelta

def send_sms_otp(user_id, phone_number, purpose):
    """
    Generate and send 6-digit SMS OTP.
    """
    # Generate random 6-digit code
    code = str(random.randint(100000, 999999))

    # Hash code before storing (bcrypt or similar)
    code_hash = hashlib.sha256(
        f"{code}:{user_id}".encode()
    ).hexdigest()

    # Store hash with 5-minute expiry
    store_otp_challenge(
        user_id=user_id,
        code_hash=code_hash,
        purpose=purpose,
        expires_at=datetime.utcnow() + timedelta(minutes=5)
    )

    # Send SMS via Twilio/SNS
    message = (
        f"Your Monzo verification code is {code}. "
        f"Valid for 5 minutes. "
        f"Never share this code."
    )
    send_sms(phone_number, message)

    # Log for audit
    log_otp_sent(user_id, 'sms', purpose)

    return True
```

Verify SMS OTP

```
def verify_sms_otp(user_id, code, purpose):
    """
    Verify SMS OTP code.
    """
    # Get stored challenge
    challenge = get_otp_challenge(user_id, purpose)

    if not challenge:
        return False

    # Check expiry
    if datetime.utcnow() > challenge['expires_at']:
        delete_otp_challenge(user_id, purpose)
        return False

    # Verify code hash
    code_hash = hashlib.sha256(
        f"{code}:{user_id}".encode()
    ).hexdigest()

    if code_hash == challenge['code_hash']:
        # Success - delete challenge
        delete_otp_challenge(user_id, purpose)
        log_mfa_success(user_id, 'sms')
        return True
    else:
        # Failed attempt
        increment_failed_attempts(user_id)

        # Rate limit: Lock after 3 failed attempts
        if get_failed_attempts(user_id) >= 3:
            lock_account_temporarily(user_id, minutes=15)

        return False
```

WARNING: SMS OTP is vulnerable to SIM swapping attacks. For high-risk operations, use TOTP or push notifications instead. Always implement rate limiting (max 3 attempts per challenge).

Push Notification Authentication

Challenge-Response Flow

Push notifications provide the best user experience for payment approvals:

```
def initiate_push_challenge(user_id, action_details):
    """
    Send push notification for user approval.
    """
    # Generate challenge ID
    challenge_id = generate_uuid()

    # Store challenge with details
    store_push_challenge(
        challenge_id=challenge_id,
        user_id=user_id,
        action=action_details['type'],
        amount=action_details.get('amount'),
        recipient=action_details.get('recipient'),
        expires_at=datetime.utcnow() + timedelta(minutes=2)
    )

    # Send push notification to user devices
    notification = {
        'title': 'Approve Payment',
        'body': f"GBP {action_details['amount']} to "
                f"{action_details['recipient']}",
        'challenge_id': challenge_id,
        'action_buttons': ['Approve', 'Decline']
    }

    send_push_notification(user_id, notification)

    # Wait for response (async)
    return challenge_id
```

Handle Push Response

```
def handle_push_response(challenge_id, user_id, approved):
    """
    Process user response to push challenge.
    """
    challenge = get_push_challenge(challenge_id)

    # Verify challenge exists and belongs to user
    if not challenge or challenge['user_id'] != user_id:
        return {'error': 'Invalid challenge'}

    # Check expiry
    if datetime.utcnow() > challenge['expires_at']:
        delete_push_challenge(challenge_id)
        return {'error': 'Challenge expired'}

    # Record response
    if approved:
        log_mfa_success(user_id, 'push')
        mark_challenge_approved(challenge_id)

        # Execute the protected action
        execute_action(challenge['action'], challenge)

        return {'status': 'approved'}
    else:
        log_mfa_declined(user_id, 'push')
        mark_challenge_declined(challenge_id)
        return {'status': 'declined'}
```

Biometric Authentication

Mobile Biometric Integration

iOS Face ID / Touch ID and Android BiometricPrompt integration:

```
// iOS Swift - Face ID / Touch ID
import LocalAuthentication

func authenticateWithBiometrics(completion: @escaping (Bool) -> Void) {
    let context = LAContext()
    var error: NSError?

    if context.canEvaluatePolicy(
        .deviceOwnerAuthenticationWithBiometrics,
        error: &error
    ) {
        let reason = "Authenticate to approve payment"

        context.evaluatePolicy(
            .deviceOwnerAuthenticationWithBiometrics,
            localizedReason: reason
        ) { success, error in
            DispatchQueue.main.async {
                if success {
                    completion(true)
                } else {
                    completion(false)
                }
            }
        }
    } else {
        completion(false)
    }
}
```

Risk-Based Authentication

Dynamic MFA Requirements

Adjust MFA requirements based on risk score:

Risk Level	Score Range	MFA Requirement	Example Triggers
Low	0-30	Optional	Known device, typical amount
Medium	31-60	SMS or TOTP	New device, unusual time
High	61-85	TOTP + SMS	Large amount, new recipient
Critical	86-100	TOTP + Push + Manual	Foreign IP, account changes

```
def calculate_authentication_requirements(user_id, action):
    """
    Determine MFA requirements based on risk score.
```

```
"""
risk_score = calculate_risk_score(user_id, action)

if risk_score < 30:
    return {'methods': [], 'optional': True}
elif risk_score < 60:
    return {'methods': ['sms_or_totp'], 'optional': False}
elif risk_score < 85:
    return {'methods': ['totp', 'sms'], 'optional': False}
else:
    return {
        'methods': ['totp', 'push', 'manual_review'],
        'optional': False,
        'alert_fraud_team': True
    }
```

Best Practices

1. Always encrypt MFA secrets (TOTP, recovery codes) with AES-256

2. Implement rate limiting: 3 attempts per challenge, 15-minute lockout

3. Use short expiry times: 5 minutes for SMS, 2 minutes for push

4. Never send actual codes in logs or error messages

5. Provide backup MFA methods (if phone lost, use TOTP)

6. Log all MFA attempts for security monitoring

7. Implement step-up authentication for sensitive operations

8. Generate recovery codes during TOTP setup (10 single-use codes)

9. Use constant-time comparison for code verification (prevent timing attacks)

10. Inform users about phishing: Never ask for codes via email/phone