FortiKey API Documentation

Introduction

FortiKey is an easy-to-implement and secure Two-Factor Authentication (2FA) platform designed to help businesses protect their users' accounts from unauthorized access. This documentation will guide you through the integration process and explain how to use the FortiKey API to add 2FA to your application.

Table of Contents

1. Getting Started
    - API Key
    - Base URL
    - Authentication
    - Rate Limiting
    - Error Handling
2. TOTP Implementation
    - Creating a New TOTP Secret
    - Validating a TOTP Token
    - Retrieving a TOTP Secret
    - Updating a TOTP Secret
    - Deleting a TOTP Secret
3. Backup Codes
    - Generating Backup Codes
    - Validating Backup Codes
4. Analytics
    - Authentication Statistics
    - User Activity
5. Implementation Examples
    - NodeJS
    - Python
    - PHP
6. Security Best Practices
7. Troubleshooting
8. API Reference

Getting Started

API Key

To use the FortiKey API, you'll need an API key. You can obtain this through the FortiKey dashboard after registering for an account.

1. Log in to your FortiKey dashboard
2. Navigate to "Manage API Keys"
3. Generate a new API key
4. Store this API key securely - it will be needed for all your API requests

Base URL

All API requests should be made to the following base URL:

```
https://fortikeybackend.onrender.com
```

Authentication

All API requests (except for TOTP validation) require authentication using your API key. Include your API key in the request headers as follows:

```
X-API-Key: your_api_key_here
```

Rate Limiting

To protect our infrastructure and ensure fair usage, the FortiKey API implements rate limiting:

- **Standard API Endpoints**: 100 requests per minute
- **TOTP Validation Endpoint**: 10 attempts per minute per IP address
- **Authentication Endpoints**: 5 attempts per minute per IP address

If you exceed these limits, you'll receive a `429 Too Many Requests` response. The response will include a `Retry-After` header indicating how many seconds to wait before retrying.

Error Handling

The FortiKey API uses conventional HTTP status codes to indicate the success or failure of a request:

- `200 OK`: Request succeeded
- `201 Created`: Resource successfully created
- `400 Bad Request`: Invalid request syntax/parameters
- `401 Unauthorized`: Invalid or missing API key
- `404 Not Found`: Resource not found
- `429 Too Many Requests`: Rate limit exceeded
- `500 Internal Server Error`: Server encountered an error

All error responses will include a JSON body with additional information:

```
{
  "message": "Detailed error message",
  "code": "ERROR_CODE"
}
```

TOTP Implementation

Creating a New TOTP Secret

To enable 2FA for a user, you first need to generate a TOTP secret for them.

Endpoint: `POST /totp-secrets`

Request Headers:

```
X-API-Key: your_api_key_here
Content-Type: application/json
```

Request Body:

```
{
  "company": "Your Company Name",
  "externalUserId": "user123",
  "backupCodes": ["code1", "code2", "code3", "code4", "code5", "code6", "code7", "code8"]
}
```

- `company`: Your company name, which will appear in the authenticator app
- `externalUserId`: Your internal user identifier (e.g., username, email, or user ID)
- `backupCodes`: An array of backup codes (optional - if not provided, the system will generate them)

Response (200 OK):

```
{
  "_id": "60a2e6b5c9f89a1d28e4f9b3",
  "secret": "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
  "backupCodes": ["code1", "code2", "code3", "code4", "code5", "code6", "code7", "code8"],
  "uri": "otpauth://totp/Your%20Company%20Name:user123?
secret=ABCDEFGHIJKLMNOPQRSTUVWXYZ&issuer=Your%20Company%20Name&algorithm=SHA1&digits=6&period=30",
  "companyId": "5f9e2d3b0e4a2c1b8a7d6f3e",
  "metadata": {
    "company": "Your Company Name",
    "createdBy": "API"
  }
}
```

The response includes:

- `_id`: MongoDB document ID for the TOTP secret (store this if you need to reference this specific secret later)
- `secret`: The TOTP secret in Base32 format
- `backupCodes`: Array of backup codes
- `uri`: The otpauth URI that can be encoded as a QR code for scanning with authenticator apps
- `companyId`: Your FortiKey company ID
- `metadata`: Additional information about the TOTP secret

Generating a QR Code:

The `uri` field from the response can be used to generate a QR code. Many libraries are available for this purpose:

```
// Example using qrcode-generator in Node.js
const qrcode = require('qrcode');
const totpUri = response.uri;

// Generate QR code as data URL
qrcode.toDataURL(totpUri, (err, dataUrl) => {
  if (err) throw err;
  console.log(dataUrl); // Use this in an <img> tag
});
```

Display this QR code to your user, who can then scan it with an authenticator app like Google Authenticator, Authy, or Microsoft Authenticator.

Validating a TOTP Token

When a user attempts to log in, they'll provide a 6-digit TOTP token from their authenticator app. You need to validate this token.

Endpoint: `POST /totp-secrets/validate`

Request Headers:

```
X-API-Key: your_api_key_here
Content-Type: application/json
```

Request Body:

```
{
  "externalUserId": "user123",
  "token": "123456"
}
```

- `externalUserId`: The user identifier you used when creating the TOTP secret
- `token`: The 6-digit code from the user's authenticator app

Response (200 OK):

```
{
  "message": "TOTP token is valid"
}
```

Response (400 Bad Request):

```
{
  "message": "Invalid TOTP token"
}
```

Implementation Flow:

1. User enters their username and password
2. If credentials are valid, prompt for the TOTP token
3. User enters the 6-digit token from their authenticator app
4. Call the validation endpoint to verify the token
5. If valid, complete the authentication process
6. If invalid, ask the user to try again or use a backup code

Retrieving a TOTP Secret

You can retrieve a TOTP secret by external user ID or by MongoDB document ID.

By External User ID:

Endpoint: `GET /totp-secrets/user/{externalUserId}`

Request Headers:

```
X-API-Key: your_api_key_here
```

Response (200 OK):

```
{
  "_id": "60a2e6b5c9f89a1d28e4f9b3",
  "externalUserId": "user123",
  "companyId": "5f9e2d3b0e4a2c1b8a7d6f3e",
  "createdAt": "2023-05-17T15:30:45.123Z",
  "metadata": {
    "company": "Your Company Name",
    "createdBy": "API"
  }
}
```

By MongoDB Document ID:

Endpoint: `GET /totp-secrets/{id}`

Request Headers:

```
X-API-Key: your_api_key_here
```

Response (200 OK):

```
{
  "_id": "60a2e6b5c9f89a1d28e4f9b3",
  "secret": "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
  "backupCodes": ["code1", "code2", "code3", "code4", "code5", "code6", "code7", "code8"],
  "externalUserId": "user123"
}
```

Updating a TOTP Secret

You can update a TOTP secret, for example, to add or change backup codes.

Endpoint: `PATCH /totp-secrets/{id}`

Request Headers:

```
X-API-Key: your_api_key_here
Content-Type: application/json
```

Request Body:

```
{
  "backupCodes": ["newcode1", "newcode2", "newcode3", "newcode4", "newcode5", "newcode6",
"newcode7", "newcode8"]
}
```

Response (200 OK):

```
{
  "_id": "60a2e6b5c9f89a1d28e4f9b3",
  "externalUserId": "user123",
  "backupCodes": ["newcode1", "newcode2", "newcode3", "newcode4", "newcode5", "newcode6",
"newcode7", "newcode8"],
  "companyId": "5f9e2d3b0e4a2c1b8a7d6f3e",
  "createdAt": "2023-05-17T15:30:45.123Z",
  "metadata": {
    "company": "Your Company Name",
    "createdBy": "API"
  }
}
```

Deleting a TOTP Secret

If a user disables 2FA or deletes their account, you should delete their TOTP secret.

Endpoint: `DELETE /totp-secrets/{id}`

Request Headers:

```
X-API-Key: your_api_key_here
```

Response (204 No Content): No response body.

Backup Codes

Generating Backup Codes

Backup codes are a fallback mechanism if a user loses access to their authenticator app. When you create a TOTP secret, you can either:

1. Generate your own backup codes and include them in the request
2. Let FortiKey generate them for you

If you want to generate your own:

```
function generateBackupCodes(count = 8, length = 10) {
  const codes = [];
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

  for (let i = 0; i < count; i++) {
    let code = '';
    for (let j = 0; j < length; j++) {
      const randomIndex = Math.floor(Math.random() * characters.length);
      code += characters[randomIndex];
    }
    codes.push(code);
  }

  return codes;
}

const backupCodes = generateBackupCodes();
```

Validating Backup Codes

Users can use backup codes instead of TOTP tokens when they don't have access to their authenticator app. FortiKey provides an endpoint to validate backup codes and automatically marks them as used after successful validation.

Endpoint: `POST /totp-secrets/validate-backup-code`

Request Headers:

```
X-API-Key: your_api_key_here
Content-Type: application/json
```

Request Body:

```
{
  "externalUserId": "user123",
  "backupCode": "ABC123XYZ"
}
```

- `externalUserId`: The user identifier you used when creating the TOTP secret
- `backupCode`: The backup code provided by the user

Response (200 OK):

```
{
  "message": "Backup code is valid",
  "remainingCodes": 7
}
```

Response (400 Bad Request):

```
{
  "message": "Invalid backup code"
}
```

Regenerating Backup Codes

When a user runs out of backup codes or wants new ones for security reasons, you can regenerate a new set of backup codes.

Endpoint: `POST /totp-secrets/user/:externalUserId/regenerate-backup`

Request Headers:

```
X-API-Key: your_api_key_here
```

URL Parameters:

```
externalUserId: The user's external ID
```

Response (200 OK):

```
{
  "message": "Backup codes regenerated successfully",
  "backupCodes": [
    "CODE1ABC",
    "CODE2DEF",
    "CODE3GHI",
    "CODE4JKL",
    "CODE5MNO",
    "CODE6PQR",
    "CODE7STU",
    "CODE8VWX"
  ]
}
```

Response (404 Not Found):

```
{
  "message": "TOTP secret not found"
}
```

Implementation Strategy:

1. Store the backup codes securely in your database
2. When a user provides a backup code, validate it using the endpoint
3. FortiKey automatically marks the code as used after successful validation
4. When a user runs out of backup codes, regenerate new ones using the regenerate endpoint
5. Show the new backup codes to the user and instruct them to save them securely

---

Analytics

FortiKey provides comprehensive analytics and monitoring capabilities through the FortiKey dashboard, allowing you to monitor 2FA usage in your application.

> **Important**: Analytics data should only be accessed through the FortiKey dashboard. These features are not available via API endpoints and are exclusively provided through the FortiKey web application.

Authentication Statistics

Through the FortiKey dashboard, you can access the following analytics:

- Total authentication attempts
- Success and failure rates
- Backup code usage statistics
- Device and browser breakdown
- Time-based trends and patterns

To access these statistics:

1. Log in to your FortiKey dashboard
2. Navigate to “Usage Analytics”
3. Use the dashboard filters to customize the time period and data displays

User Activity

The FortiKey dashboard also provides user-specific analytics to help identify potential issues or suspicious activity.

Key user activity metrics available in the dashboard:

- Authentication attempts per user
- Success and failure rates by user
- Device usage patterns
- Suspicious activity flags and alerts

For suspicious activity monitoring:

1. Log in to your FortiKey dashboard
2. Navigate to “User Analytics”
3. Search for specific users or filter by activity type

Implementation Examples

NodeJS

Here’s a complete implementation example using Node.js with Express:

```
const express = require('express');
const axios = require('axios');
const qrcode = require('qrcode');

const app = express();
app.use(express.json());

// Configuration
const FORTIKEY_API_KEY = 'your_api_key_here';
const FORTIKEY_BASE_URL = 'https://fortikeybackend.onrender.com/api/v1';

// FortiKey API Client
const fortiKeyClient = axios.create({
  baseURL: FORTIKEY_BASE_URL,
  headers: {
    'X-API-Key': FORTIKEY_API_KEY,
    'Content-Type': 'application/json'
  }
});

// Step 1: Enable 2FA for a user
app.post('/enable-2fa', async (req, res) => {
  try {
    const { userId } = req.body;

    // Generate backup codes
    const backupCodes = Array.from({ length: 8 }, () =>
      Math.random().toString(36).substring(2, 12).toUpperCase()
    );

    // Create TOTP secret
    const response = await fortiKeyClient.post('/totp-secrets', {
      company: 'Your Company Name',
      externalUserId: userId,
      backupCodes
    });
```

```
    // Generate QR code
    const qrCodeDataUrl = await qrcode.toDataURL(response.data.uri);

    // Store backup codes in your database
    // await storeBackupCodes(userId, backupCodes);

    res.json({
      qrCode: qrCodeDataUrl,
      backupCodes,
      secret: response.data.secret // Consider whether you need to expose this
    });
  } catch (error) {
    console.error('FortiKey API Error:', error.response?.data || error.message);
    res.status(500).json({
      message: 'Failed to enable 2FA',
      error: error.response?.data?.message || error.message
    });
  }
});

// Step 2: Verify TOTP or backup code during login
app.post('/verify-2fa', async (req, res) => {
  try {
    const { userId, code } = req.body;

    // Try the code as both TOTP and backup code
    const verificationResult = await verifyAuthentication(userId, code);

    if (verificationResult.authenticated) {
      // Authentication successful
      res.json({
        authenticated: true,
        method: verificationResult.method,
        message: verificationResult.message,
        ...(verificationResult.remainingCodes && {
          remainingCodes: verificationResult.remainingCodes
        })
      });
    } else {
      // Authentication failed
      res.status(400).json({
        authenticated: false,
        message: verificationResult.message
      });
    }
```

```
  } catch (error) {
    console.error('Authentication Error:', error);
    res.status(500).json({
      message: 'Error during authentication',
      error: error.message
    });
  }
});

// Step 3: Disable 2FA for a user
app.post('/disable-2fa', async (req, res) => {
  try {
    const { userId, totpSecretId } = req.body;

    // Delete the TOTP secret
    await fortiKeyClient.delete(`/totp-secrets/${totpSecretId}`);

    // Remove backup codes from your database
    // await removeBackupCodes(userId);

    res.json({
      success: true,
      message: '2FA has been disabled for the user'
    });
  } catch (error) {
    console.error('FortiKey API Error:', error.response?.data || error.message);
    res.status(500).json({
      message: 'Failed to disable 2FA',
      error: error.response?.data?.message || error.message
    });
  }
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Python

Implementation example using Python with Flask:

```
from flask import Flask, request, jsonify
import requests
import qrcode
import io
import base64
import random
import string

app = Flask(__name__)

# Configuration
FORTIKEY_API_KEY = 'your_api_key_here'
FORTIKEY_BASE_URL = 'https://fortikeybackend.onrender.com/api/v1'

# FortiKey API Headers
headers = {
    'X-API-Key': FORTIKEY_API_KEY,
    'Content-Type': 'application/json'
}

# Helper function to generate QR code as data URL
def generate_qr_data_url(uri):
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    qr.add_data(uri)
    qr.make(fit=True)
    img = qr.make_image(fill_color="black", back_color="white")

    img_io = io.BytesIO()
    img.save(img_io, 'PNG')
    img_io.seek(0)
    return '' + base64.b64encode(img_io.read()).decode('utf-8')

# Generate random backup codes
def generate_backup_codes(count=8, length=10):
```

```
    codes = []
    for _ in range(count):
        code = ''.join(random.choices(string.ascii_uppercase + string.digits, k=length))
        codes.append(code)
    return codes

# Enable 2FA for a user
@app.route('/enable-2fa', methods=['POST'])
def enable_2fa():
    data = request.json
    user_id = data.get('userId')

    if not user_id:
        return jsonify({"message": "userId is required"}), 400

    try:
        # Generate backup codes
        backup_codes = generate_backup_codes()

        # Create TOTP secret
        response = requests.post(
            f"{FORTIKEY_BASE_URL}/totp-secrets",
            json={
                "company": "Your Company Name",
                "externalUserId": user_id
            },
            headers=headers
        )

        response.raise_for_status()
        totp_data = response.json()

        # Generate QR code
        qr_code_data_url = generate_qr_data_url(totp_data['uri'])

        # Here you would store backup codes in your database
        # store_backup_codes(user_id, backup_codes)

        return jsonify({
            "qrCode": qr_code_data_url,
            "backupCodes": backup_codes,
            "secret": totp_data['secret'],
            "secretId": totp_data['_id']
        })
```

```
    except requests.RequestException as e:
        app.logger.error(f"FortiKey API Error: {e}")
        return jsonify({
            "message": "Failed to enable 2FA",
            "error": str(e)
        }), 500

# Add this function to the Python implementation
def verify_authentication(user_id, code):
    """
    Try to verify the code as both TOTP token and backup code
    """
    try:
        # First try as TOTP token
        try:
            response = requests.post(
                f"{FORTIKEY_BASE_URL}/totp-secrets/validate",
                json={
                    "externalUserId": user_id,
                    "token": code
                },
                headers=headers
            )

            # If response is successful, it's a valid TOTP token
            if response.status_code == 200:
                return {
                    "authenticated": True,
                    "method": "totp",
                    "message": "TOTP verification successful"
                }
        except requests.RequestException:
            # Ignore error, we'll try as backup code next
            pass

        # Next try as backup code
        try:
            response = requests.post(
                f"{FORTIKEY_BASE_URL}/totp-secrets/validate-backup-code",
                json={
                    "externalUserId": user_id,
                    "backupCode": code
                },
                headers=headers
            )
```

```
            # If response is successful, it's a valid backup code
            if response.status_code == 200:
                response_data = response.json()
                return {
                    "authenticated": True,
                    "method": "backup",
                    "message": "Backup code verification successful",
                    "remainingCodes": response_data.get("remainingCodes", 0)
                }
        except requests.RequestException:
            # Both TOTP and backup code failed
            pass

        # If we get here, neither TOTP nor backup code was valid
        return {
            "authenticated": False,
            "message": "Invalid authentication code"
        }

    except Exception as e:
        app.logger.error(f"Authentication error: {str(e)}")
        raise Exception("Error during authentication verification")

# Update the verify-2fa endpoint
@app.route('/verify-2fa', methods=['POST'])
def verify_2fa():
    data = request.json
    user_id = data.get('userId')
    code = data.get('code')  # This could be either a TOTP token or backup code

    if not user_id or not code:
        return jsonify({"message": "userId and code are required"}), 400

    try:
        # Try the code as both TOTP and backup code
        verification_result = verify_authentication(user_id, code)

        if verification_result["authenticated"]:
            response_data = {
                "authenticated": True,
                "method": verification_result["method"],
                "message": verification_result["message"]
            }
```

```
            # Add remaining codes if it's a backup code
            if "remainingCodes" in verification_result:
                response_data["remainingCodes"] = verification_result["remainingCodes"]

            return jsonify(response_data)
        else:
            return jsonify({
                "authenticated": False,
                "message": verification_result["message"]
            }), 400

    except Exception as e:
        app.logger.error(f"Authentication error: {str(e)}")
        return jsonify({
            "message": "Error during authentication",
            "error": str(e)
        }), 500

# Disable 2FA for a user
@app.route('/disable-2fa', methods=['POST'])
def disable_2fa():
    data = request.json
    user_id = data.get('userId')
    totp_secret_id = data.get('totpSecretId')

    if not user_id or not totp_secret_id:
        return jsonify({"message": "userId and totpSecretId are required"}), 400

    try:
        # Delete the TOTP secret
        response = requests.delete(
            f"{FORTIKEY_BASE_URL}/totp-secrets/{totp_secret_id}",
            headers=headers
        )

        response.raise_for_status()

        # Here you would remove backup codes from your database
        # remove_backup_codes(user_id)

        return jsonify({
            "success": True,
            "message": "2FA has been disabled for the user"
        })
```

```
    except requests.RequestException as e:
        app.logger.error(f"FortiKey API Error: {e}")
        return jsonify({
            "message": "Failed to disable 2FA",
            "error": str(e)
        }), 500

if __name__ == '__main__':
    app.run(debug=True, port=3000)
```

PHP

Implementation example using PHP:

```
<?php
// Required libraries:
// - guzzlehttp/guzzle: For API requests
// - chillerlan/php-qrcode: For QR code generation

require 'vendor/autoload.php';

use GuzzleHttp\Client;
use GuzzleHttp\Exception\RequestException;
use chillerlan\QRCode\QRCode;
use chillerlan\QRCode\QROptions;

// Configuration
$config = [
    'fortikey_api_key' => 'your_api_key_here',
    'fortikey_base_url' => 'https://fortikeybackend.onrender.com/api/v1',
    'company_name' => 'Your Company Name'
];

// Initialize Guzzle client
$client = new Client([
    'base_uri' => $config['fortikey_base_url'],
    'headers' => [
        'X-API-Key' => $config['fortikey_api_key'],
        'Content-Type' => 'application/json'
    ]
]);

// Generate random backup codes
function generateBackupCodes($count = 8, $length = 10) {
    $codes = [];
    $characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

    for ($i = 0; $i < $count; $i++) {
        $code = '';
        for ($j = 0; $j < $length; $j++) {
            $code .= $characters[rand(0, strlen($characters) - 1)];
        }
        $codes[] = $code;
```

```
    }

    return $codes;
}

// Enable 2FA for a user
function enableTwoFactor($userId) {
    global $client, $config;

    try {
        // Generate backup codes
        $backupCodes = generateBackupCodes();

        // Create TOTP secret
        $response = $client->post('/totp-secrets', [
            'json' => [
                'company' => $config['company_name'],
                'externalUserId' => $userId,
                'backupCodes' => $backupCodes
            ]
        ]);

        $totpData = json_decode($response->getBody(), true);

        // Generate QR code
        $qrOptions = new QROptions([
            'outputType' => QRCode::OUTPUT_IMAGE_PNG,
            'eccLevel' => QRCode::ECC_L,
            'scale' => 5,
            'imageBase64' => true,
        ]);

        $qrCode = new QRCode($qrOptions);
        $qrCodeDataUrl = '' . $qrCode->render($totpData['uri']);

        // Here you would store backup codes in your database
        // storeBackupCodes($userId, $backupCodes);

        return [
            'success' => true,
            'qrCode' => $qrCodeDataUrl,
            'backupCodes' => $backupCodes,
            'secret' => $totpData['secret'],
            'secretId' => $totpData['_id']
        ];
```

```
    } catch (RequestException $e) {
        $errorMessage = $e->getMessage();

        if ($e->hasResponse()) {
            $errorBody = json_decode($e->getResponse()->getBody(), true);
            $errorMessage = $errorBody['message'] ?? $errorMessage;
        }

        return [
            'success' => false,
            'message' => 'Failed to enable 2FA',
            'error' => $errorMessage
        ];
    }
}

// Add regenerate backup codes function to the PHP implementation
function regenerateBackupCodes($userId) {
    global $client;

    try {
        // Call the regenerate endpoint
        $response = $client->post("/totp-secrets/user/{$userId}/regenerate-backup");

        // Parse the response
        $responseData = json_decode($response->getBody(), true);

        return [
            'success' => true,
            'message' => 'Backup codes regenerated successfully',
            'backupCodes' => $responseData['backupCodes']
        ];

    } catch (RequestException $e) {
        $errorMessage = $e->getMessage();

        if ($e->hasResponse()) {
            $errorBody = json_decode($e->getResponse()->getBody(), true);
            $errorMessage = $errorBody['message'] ?? $errorMessage;
        }

        return [
            'success' => false,
            'message' => 'Failed to regenerate backup codes',
```

```
            'error' => $errorMessage
        ];
    }
}

// Modify the verifyTwoFactor function to check both TOTP and backup codes
function verifyTwoFactor($userId, $code) {
    global $client;

    try {
        // First try as TOTP token
        try {
            $response = $client->post('/totp-secrets/validate', [
                'json' => [
                    'externalUserId' => $userId,
                    'token' => $code
                ]
            ]);

            // If we get here, the code was a valid TOTP token
            return [
                'success' => true,
                'authenticated' => true,
                'method' => 'totp',
                'message' => 'TOTP verification successful'
            ];
        } catch (RequestException $tokenError) {
            // If token validation failed, try as backup code
            if ($tokenError->getResponse() && $tokenError->getResponse()->getStatusCode() ===
400) {
                try {
                    $response = $client->post('/totp-secrets/validate-backup-code', [
                        'json' => [
                            'externalUserId' => $userId,
                            'backupCode' => $code
                        ]
                    ]);

                    // Parse response
                    $responseData = json_decode($response->getBody(), true);

                    // If we get here, the code was a valid backup code
                    return [
                        'success' => true,
                        'authenticated' => true,
```

```
                        'method' => 'backup',
                        'message' => 'Backup code verification successful',
                        'remainingCodes' => $responseData['remainingCodes'] ?? 0
                    ];
                } catch (RequestException $backupError) {
                    // Both TOTP and backup code validation failed
                    return [
                        'success' => false,
                        'authenticated' => false,
                        'message' => 'Invalid authentication code'
                    ];
                }
            } else {
                // Unexpected error during token validation
                throw $tokenError;
            }
        }
    } catch (RequestException $e) {
        $errorMessage = 'Error validating authentication';
        $statusCode = 500;

        if ($e->hasResponse()) {
            $statusCode = $e->getResponse()->getStatusCode();
            $errorBody = json_decode($e->getResponse()->getBody(), true);
            $errorMessage = $errorBody['message'] ?? $errorMessage;
        }

        return [
            'success' => false,
            'authenticated' => false,
            'message' => $errorMessage
        ];
    }
}

// Disable 2FA for a user
function disableTwoFactor($userId, $totpSecretId) {
    global $client;

    try {
        // Delete the TOTP secret
        $response = $client->delete("/totp-secrets/{$totpSecretId}");

        // Here you would remove backup codes from your database
        // removeBackupCodes($userId);
```

```
        return [
            'success' => true,
            'message' => '2FA has been disabled for the user'
        ];

    } catch (RequestException $e) {
        $errorMessage = $e->getMessage();

        if ($e->hasResponse()) {
            $errorBody = json_decode($e->getResponse()->getBody(), true);
            $errorMessage = $errorBody['message'] ?? $errorMessage;
        }

        return [
            'success' => false,
            'message' => 'Failed to disable 2FA',
            'error' => $errorMessage
        ];
    }
}
```

Security Best Practices

When implementing 2FA with FortiKey, follow these security best practices to ensure maximum protection:

1. **Securely Store API Keys**:
    - Never expose your API key in client-side code
    - Use environment variables or a secure key management solution
    - Regularly rotate your API keys (we recommend every 90 days)
2. **Backup Code Security**:
    - Store backup codes securely (use hashing, similar to password storage)
    - Allow users to view their backup codes only once
    - Implement a mechanism to mark backup codes as used
3. **Error Messages**:
    - Use generic error messages to prevent information leakage

- Log detailed errors server-side for troubleshooting

4. **Rate Limiting**:
   - Implement your own rate limiting in addition to FortiKey's built-in rate limits
   - Lock accounts after a certain number of failed attempts
5. **HTTPS/TLS**:
   - Always use HTTPS for API requests
   - Use TLS 1.2 or higher
6. **User Education**:
   - Provide clear instructions on how to set up authenticator apps
   - Explain the importance of securely storing backup codes
   - Offer guidance on what to do if a device is lost
7. **Monitoring**:
   - Regularly check authentication statistics for suspicious activity
   - Set up alerts for unusual patterns

Troubleshooting

Common Issues

TOTP Validation Failures

If users consistently have trouble validating their TOTP tokens:

1. **Time Synchronization**:
   - Ensure the user's device has accurate time (many authentication failures are due to time drift)
   - FortiKey allows a small window before/after the current period to account for minor time differences
2. **Invalid Secret**:
   - If a user reports issues after setup, consider regenerating their TOTP secret
   - Ensure the QR code was scanned correctly
3. **Rate Limiting**:

- Check if you're hitting rate limits (429 responses)
- Implement exponential backoff for retry mechanisms

API Key Issues

If you're experiencing authentication errors with your API key:

1. **Expired Key**:
   - Check if the key needs to be rotated
   - Generate a new API key from the FortiKey dashboard
2. **Header Format**:
   - Ensure the API key is correctly formatted in the request header
   - Verify there are no trailing spaces or line breaks
3. **Permissions**:
   - Confirm your API key has the necessary permissions

Error Codes

Code	Description	Solution
`INVALID_TOKEN`	The TOTP token provided is invalid	Verify the token and try again
`USER_NOT_FOUND`	The external user ID was not found	Check if the user has a TOTP secret configured
`RATE_LIMITED`	Too many requests	Wait and retry with exponential backoff
`INVALID_API_KEY`	The API key is invalid or expired	Check or rotate your API key
`INVALID_REQUEST`	The request format is incorrect	Verify your request parameters

Support

If you encounter issues not covered in this documentation:

1. Check the error message returned by the API
2. Review the relevant section in this documentation
3. Contact support through the FortiKey dashboard

4. Email support@fortikey.com with your API key (first 4 characters only) and a detailed description of the issue

---

API Reference

> **Note**: Analytics features are only available through the FortiKey dashboard interface and not via API endpoints.

TOTP Secrets

Endpoint	Method	Description
`/totp-secrets`	POST	Create a new TOTP secret
`/totp-secrets`	GET	Get all TOTP secrets
`/totp-secrets/user/{externalUserId}`	GET	Get a TOTP secret by external user ID
`/totp-secrets/{id}`	GET	Get a TOTP secret by MongoDB document ID
`/totp-secrets/{id}`	PATCH	Update a TOTP secret
`/totp-secrets/{id}`	DELETE	Delete a TOTP secret
`/totp-secrets/validate`	POST	Validate a TOTP token
`/totp-secrets/validate-backup-code`	POST	Validate a backup code

Endpoint	Method	Description
`/totp-secrets/user/:externalUserId/regenerate-backup`	POST	Regenerate backup codes for a user

Conclusion

FortiKey provides a secure, easy-to-integrate 2FA solution for your application. By following this documentation and implementing the provided examples, you can quickly add 2FA to your authentication flow, enhancing security for your users.

Remember that 2FA is just one component of a comprehensive security strategy. It should be used alongside other security measures such as strong password policies, secure session management, and regular security audits.

For updates and new features, regularly check the FortiKey dashboard and documentation.