Quantum-Resistant REST API for Post-Quantum Cryptography Key Exchange

---

Overview

Welcome to the **Quantum-Resistant Key Exchange REST API,** an innovative API service that is immune to the risk of quantum attack. Developers can use this **API to conduct post-quantum cryptographic key exchanges in the form of Kyber, a lattice-based scheme,** which NIST has chosen as a quantum-safe communication standard.

This API is ideal for:

- Secure messaging apps
- Government or enterprise systems
- Blockchain and financial institutions
- IoT security platforms
- MAANG-scale infrastructure engineers seeking post-quantum readiness

Note: This API uses **Kyber1024** and is compliant with **NIST PQC standards**. It is built for research and production environments seeking cryptographic resilience against quantum computers.

Features

- **Kyber-based Key Encapsulation Mechanism (KEM)**
- Quantum-safe ephemeral key exchange
- RESTful API endpoints
- JSON-based payloads

- Human-readable error handling
- OpenAPI 3.1 Spec
- Designed for scalability and composability
- Ready for integration into microservice security architecture

---

Table of Contents

Authentication

This API requires **OAuth 2.0 Bearer Token** authentication. Each request must include a valid token in the Authorization header:

Authorization: Bearer <your-access-token>

To obtain an access token, register here or use the /auth/token endpoint if you're an enterprise user.

API Base URL

https://api.quantumkeyx.com/v1

All endpoints described below are relative to this base URL.

OpenAPI Spec

```
openapi: 3.1.0
info:
  title: Quantum-Resistant Key Exchange API
  description: REST API for Kyber-based post-quantum secure communication
  version: 1.0.0
servers:
  - url: https://api.quantumkeyx.com/v1
paths:
  /keys/server:
    get:
      summary: Get server's public key
      responses:
        '200':
          description: Server public key in base64
          content:
            application/json:
```

```
          schema:
            type: object
            properties:
              public_key:
                type: string
                format: byte
/keys/client:
  post:
    summary: Send client’s public key and receive shared secret
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              client_public_key:
                type: string
                format: byte
    responses:
      '200':
        description: Encrypted shared secret and ciphertext
        content:
          application/json:
            schema:
```

```
        type: object
        properties:
          ciphertext:
            type: string
            format: byte
          shared_secret:
            type: string
            format: byte
components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

Step-by-Step Usage Guide

Step 1: Authenticate

Request your OAuth2 token:

```
POST /auth/token
Content-Type: application/json

{
  "client_id": "your-app-id",
  "client_secret": "your-secret"
}
```

Step 2: Fetch Server's Kyber Public Key

GET /keys/server

Authorization: Bearer <your-access-token>

Response:

{

"public_key": "aGVsbG9fc2VydmVyX2tleV9iYXNlNjQ="

}

Step 3: Send Client Public Key and Receive Ciphertext + Shared Secret

POST /keys/client

Authorization: Bearer <your-access-token>

Content-Type: application/json

{

"client_public_key": "Y2xpZW50X2tleV9iYXNlNjQ="

}

Response:

{

"ciphertext": "Y2lwaGVydGV4dF9ieXRlcw==",

"shared_secret": "c2hhcmVkX3NlY3JldF9iYXNlNjQ="

}

You now have a shared secret that is **post-quantum secure**, using the **Kyber1024** key exchange algorithm.

Endpoints

1. GET /keys/server

- **Purpose**: Fetch the server's public key (Kyber).
- **Use**: Required before starting key exchange.
- **Headers**: Authorization: Bearer <token>
- **Response**:


```
{
  "public_key": "base64-server-key"
}
```


---

2. POST /keys/client

- **Purpose**: Send your client public key and receive the ciphertext + shared secret.
- **Body**:


```
{
  "client_public_key": "base64-client-key"
}
```


- **Response**:


```
{
  "ciphertext": "base64-ciphertext",
  "shared_secret": "base64-secret"
}
```


---

3. POST /auth/token

- **Use**: Get your access token via OAuth2.
- **Headers**: Content-Type: application/json

- **Body**:

```
{
  "client_id": "abc",
  "client_secret": "xyz"
}
```

- **Response**:

```
{
  "access_token": "your-jwt-token",
  "expires_in": 3600
}
```

Error Handling & Codes

Code	Meaning	Message
401	Unauthorized	Invalid token
400	Bad Request	Invalid input format
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Something went wrong server-side

Rate Limiting

Tier 1 (Free): 100 requests/day
Tier 2 (Pro): 10,000 requests/day
Tier 3 (Enterprise): Custom SLA, rate-limit exempt

Rate limiting is based on **IP + Token**.

Versioning

All APIs follow semantic versioning.
Current: v1
Coming Soon: v1.1 with **Hybrid PQ-EC cryptography**

Use the version prefix in your URLs:

https://api.quantumkeyx.com/v1/keys/server

Changelog

Version	Changes	Date
v1.0.0	Initial release, Kyber key exchange	2025-07-26
v1.1.0	Forthcoming: Add SIKE (Supersingular Isogeny)	Aug 2025

Security Considerations

- No key material is persisted on the server.
- Resistant to **quantum adversaries** using Grover or Shor's algorithms.
- Uses **base64-encoded raw bytes** for safety across JSON payloads.
- Keys generated in accordance with **FIPS 140-3 Level 2 hardware modules**.