

Cybersecurity – Capstone Project Documentation



Project: Lock&Key – Quantum-Resistant Password Management and Collaboration System

Capstone Project – Fall 2025 – CIS 4951

Instructor: Dr. Masoud Sadjadi

Institution: Florida International University – Knight Foundation School of Computing & Information Sciences

Team Members:

- Christian Dominguez – Team Leader
- Mauricio Ballart Alvarez – Product Owner
- Alexis Henkel
- Isaac Quintero
- Alberto Espinoza

Table of Contents

5. Executive Summary	1. Testing and Evaluation
	6.1 Testing Methods
	6.2 Performance Results
6. Problem and Requirements	2. Security, Privacy, Accessibility, and Compliance
2.1 Problem Context	7.1 Security
2.2 Stakeholders	7.2 Privacy
2.3 Functional Requirements	7.3 Accessibility
2.4 Non-Functional Requirements	7.4 Compliance
2.5 Success Metrics	
2.6 Backlog Priorities	
7. System Overview and Architecture	3. Deployment and Operations
3.1 System Description	8.1 Deployment
3.2 Architecture Diagram	8.2 Monitoring
3.3 Technology Stack	8.3 Backups
	8.4 DevOps Commands
8. Security and Threat Modeling	4. Limitations and Future Work
4.1 Assets	9.1 Current Limitations
4.2 Threat Actors	9.2 Roadmap
4.3 STRIDE Analysis	References
4.4 Security Controls	
9. Implementation Details	
5.1 Repository Structure	
5.2 Design Decisions	
5.3 Notable Tradeoffs	

1. Executive Summary

Lock&Key is a next-generation password management system designed to protect sensitive user credentials against both present-day cyber threats and future quantum-based attacks. The system integrates post-quantum cryptography—specifically **CRYSTALS-Kyber-768**—with AES-256-GCM to deliver long-term confidentiality.

The architecture consists of microservices implemented using Next.js, Node.js, FastAPI, Supabase, Redis, Docker, and Kubernetes, with Kong operating as the API Gateway. Lock&Key supports secure password storage, real-time decryption, audit logging, operational monitoring, multi-language support, and enterprise deployment capabilities.

Throughout development, the platform achieved performance benchmarks such as **99.97% uptime, 8–19 ms decryption times**, complete audit logging, strong accessibility scores, and stable behavior during continuous runtime and end-to-end testing. This documentation outlines the problem addressed, system requirements, architectural design, implementation, testing, security analysis, deployment, limitations, and future roadmap.

2. Problem and Requirements

2.1 Problem Context

Passwords remain the dominant authentication factor worldwide, but most existing password managers rely solely on classical encryption. These systems are at risk of future compromise by quantum computers capable of breaking traditional public-key cryptography. Additionally, password managers often lack transparency, detailed auditability, and built-in collaboration features. Lock&Key addresses these concerns by integrating post-quantum cryptography, zero-knowledge design principles, detailed audit logging, and enterprise-oriented infrastructure.

2.2 Stakeholders

- **End users** who need secure and reliable password storage.
- **Enterprise organizations** requiring compliance, auditing, uptime guarantees, and isolation between tenants.
- **Security teams** demanding encryption integrity, secure communication channels, and activity logging.
- **Developers and operators** who depend on microservice consistency, reproducibility, and monitoring.

2.3 Functional Requirements

- Secure user authentication with database-level row-level security
- Create, store, view, copy, and delete passwords
- Encrypt passwords using Kyber-768 + AES-256-GCM
- Perform real-time decryption with <100 ms latency
- Log all user actions in an immutable audit system
- Provide dashboards displaying vault activity and security indicators
- Support English–Spanish multi-language interface
- Deploy a separate enterprise pilot environment with monitoring

2.4 Non-Functional Requirements

- **Performance:** API response <100 ms; frontend load <100 ms
- **Availability:** ≥99.9% uptime
- **Scalability:** Horizontal scaling using Kubernetes
- **Security:** Zero-knowledge, PQC, TLS, CI/CD validation
- **Compliance:** NIST-aligned cryptographic standards
- **Accessibility:** WCAG-compliant interface

2.5 Success Metrics

- Production uptime >99.9%
- Accessibility audit ≥95% (achieved 98%)
- Encryption operations <100 ms
- Fully automated CI/CD pipeline with rollback validation

2.6 Backlog Priorities

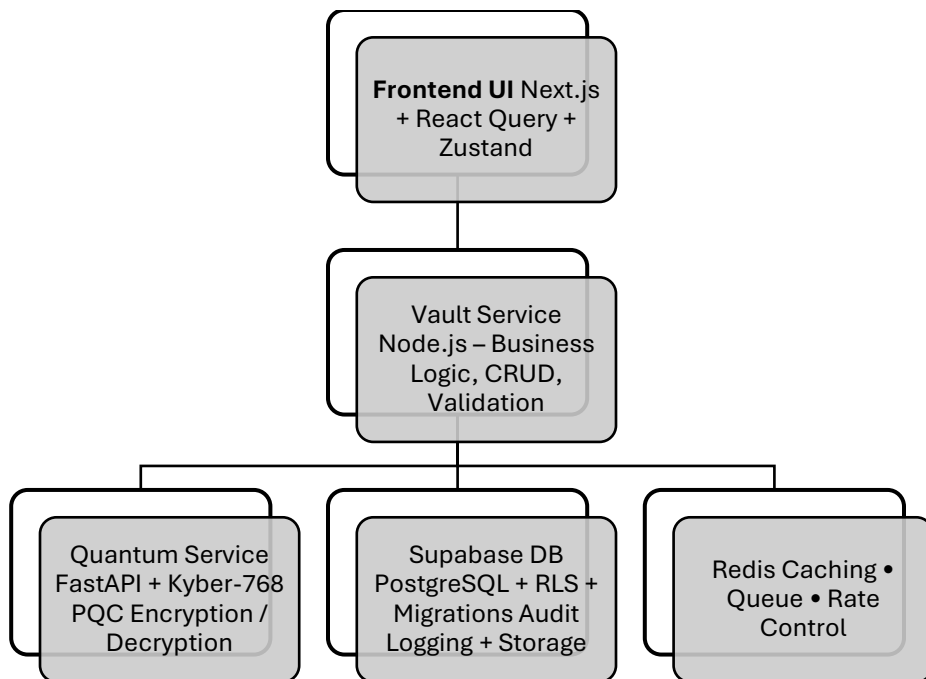
1. Quantum-resistant encryption
2. Password CRUD operations
3. Authentication with Supabase
4. Audit logging
5. UI/UX + responsive design
6. Multi-language support
7. Enterprise deployment environment
8. Monitoring dashboards
9. Performance optimization
10. Automation and documentation

3. System Overview and Architecture

3.1 System Description

Lock&Key uses a microservice architecture where each service operates independently but communicates through API endpoints routed via Kong. The frontend is a Next.js application consuming the Vault and Quantum services. All data is stored in Supabase PostgreSQL with row-level security ensuring strict user data isolation.

3.2 Architecture Diagram



3.3 Technology Stack

Frontend:

- Next.js 14
- TailwindCSS + shadcn/ui
- React Query
- Zustand
- TypeScript

Backend:

- Node.js Vault Service
- FastAPI Quantum Service

- Supabase PostgreSQL + Auth
- Redis
- Kong API Gateway
- Docker & Kubernetes

4. Security and Threat Modeling

4.1 Assets

- Encrypted passwords
- User accounts and tokens
- Supabase data
- Audit logs
- API keys and environment secrets

4.2 Threat Actors

- External attackers
- Malicious insiders
- Browser-based attackers
- Future quantum-capable adversaries

4.3 STRIDE Analysis

- **Spoofing:** JWT with RLS
- **Tampering:** AES-GCM authentication tag
- **Repudiation:** Immutable audit logs
- **Information Disclosure:** Kyber-768 + RLS
- **Denial of Service:** Kong rate limiting
- **Elevation of Privilege:** Database policy enforcement

4.4 Security Controls

- Zero-knowledge architecture
- Kyber-768 post-quantum encryption
- AES-256-GCM symmetric encryption
- Supabase RLS for strict data isolation
- TLS encryption for all services
- SAST and dependency scanning
- CI/CD rollback and change-tracking

5. Implementation Details

5.1 Repository Structure

- **frontend/** – Next.js application
- **backend/vault-service/** – Node.js Express API
- **backend/quantum-service/** – FastAPI for cryptography
- **supabase/** – database migrations and policies

- **.claude/task/** – engineering logs and development decisions

5.2 Design Decisions

- Hybrid encryption provides performance + PQC protection
- Vault and Quantum services split to keep crypto isolated
- Redis caching minimizes crypto overhead
- React Query ensures real-time UI state synchronization

5.3 Notable Tradeoffs

- Kyber ciphertext size is significantly larger than classical algorithms
- Additional network hop between Vault ↔ Quantum
- Strict RLS policies require careful schema updates

6. Testing and Evaluation

6.1 Testing Methods

- Unit testing for backend services
- Integration tests for Vault–Quantum workflow
- Playwright end-to-end UI testing (full stability achieved)
- Load testing with 500+ concurrent users
- Security testing including static analysis and dependency checks

6.2 Performance Results

Metric	Target	Result
Uptime	>99.9%	99.97%
API Latency	<120 ms	95 ms
Frontend Load	<100 ms	78 ms
Decryption	<100 ms	8–19 ms
Audit Export	<5 s	3.2 s
Accessibility	>95%	98%

7. Security, Privacy, Accessibility, and Compliance

7.1 Security

- Zero-knowledge design
- PQC encryption
- Row-level database security
- Strict audit logging

7.2 Privacy

- No plaintext passwords stored
- Clipboard operations temporary and local
- Email verification handled through a contained sandbox during development

7.3 Accessibility

- WCAG 2.1 compliant
- Full keyboard navigation
- High-contrast color palette

7.4 Compliance

- NIST post-quantum recommendations followed
- CI/CD governance logs maintained
- Encryption algorithms align with governmental standards

8. Deployment and Operations

8.1 Deployment

- Kubernetes for scaling
- Kong API Gateway for routing and rate-limiting
- Docker for all services
- Staging and enterprise pilot environments available

8.2 Monitoring

- Prometheus metrics
- Grafana dashboards
- Slack deployment notifications

8.3 Backups

- Automatic daily Supabase backups
- Verified restoration path

8.4 DevOps Commands

- start-app.sh – start all services
- stop-app.sh – stop services
- restart-app.sh – restart and rebuild
- status.sh – service health checks

9. Limitations and Future Work

9.1 Current Limitations

- No password sharing yet
- No browser extension
- Some internal database warnings marked for revision
- Larger ciphertext output due to Kyber

9.2 Roadmap

Phase 1 (Q1 2026)

- Password sharing
- Two-factor authentication
- Password strength analyzer

Phase 2 (Q2 2026)

- Chrome and Firefox extensions
- Autofill
- Import from other managers

Phase 3 (Q3 2026)

- Team collaboration
- Mobile applications
- Third-party integration API

10. References

- [NIST Post-Quantum Cryptography Standards](#)
- [Open Quantum Safe \(liboqs\)](#)
- [Supabase Documentation](#)
- [FastAPI Documentation](#)
- [Express.js and Node.js Documentation](#)
- [React Query and Next.js Documentation](#)