Product Requirements Document (PRD)

Keeper Secrets Manager VS Code Extension

Document Information

- **Product**: Keeper Secrets Manager VS Code Extension
- **Version**: 1.0.0
- **Date**: July 2025
- **Author**: Product Team
- **Status**: Draft

---

1. Executive Summary

1.1 Product Vision

Create the most intuitive and powerful secrets management VS Code extension that seamlessly integrates with developers' workflows while maintaining enterprise-grade security. The extension will leverage Keeper's advanced notation system and robust SDK to deliver a superior developer experience that surpasses all existing competitors.

1.2 Key Success Metrics

- **Adoption**: 50,000+ active users within 6 months
- **User Satisfaction**: 4.8+ star rating on VS Code Marketplace
- **Performance**: <100ms response time for secret retrieval
- **Security**: Zero security incidents in production

1.3 Competitive Advantage

- **Zero External Dependencies**: No CLI installation required (vs 1Password)
- **Advanced Notation System**: More powerful than any competitor
- **Native SDK Integration**: Direct integration vs CLI wrappers
- **Comprehensive Security**: Enterprise-grade with offline capabilities

---

2. Market Analysis & Competitive Landscape

2.1 Current Market Leaders

Extension	Install Base	Key Strengths	Limitations
1Password	500K+	Simple UX, Good branding	Requires CLI, Limited notation
Doppler	100K+	Two-way sync, Autocomplete	Cloud-only, Environment vars only
HashiCorp Vault	50K+	Enterprise features	Complex setup, Self-hosted required

2.2 Market Opportunity

- **Gap**: No extension offers both simplicity AND enterprise features
- **Opportunity**: Developers want powerful secrets management without complexity
- **Timing**: Growing focus on security in development workflows

---

3. Product Requirements

3.0 MVP Features (Phase 0)

3.0.1 Basic Extension Setup

User Story: As a developer, I want to install and configure the Keeper VS Code extension with minimal setup.

Requirements:

- **Extension Installation**: Install from VS Code Marketplace
- **Basic Authentication**: One-Time Token (OTT) authentication only
- **Single Profile**: Support for one configuration profile
- **Basic Settings**: Simple configuration via VS Code settings
- **Connection Test**: Verify connection to Keeper servers

Success Criteria:

- Extension installs and activates successfully
- User can authenticate with OTT in <2 minutes
- Basic connection validation works

3.0.2 Secret Retrieval & Display

User Story: As a developer, I want to view and retrieve my secrets in VS Code.

Requirements:

- **Secret List**: Display all accessible secrets in a simple list
- **Secret Details**: Show basic secret information (title, type, fields)
- **Copy to Clipboard**: Copy field values to clipboard
- **Basic Search**: Simple text search across secret titles
- **Refresh**: Manual refresh of secret list

Success Criteria:

- Display up to 100 secrets efficiently
- Copy functionality works reliably
- Search finds secrets by title

3.0.3 Basic Commands

User Story: As a developer, I want basic commands to interact with secrets.

Requirements:

- **Command Palette**: Basic commands accessible via Command Palette
- **Authenticate Command**: `Keeper: Authenticate`
- **List Secrets Command**: `Keeper: List Secrets`
- **Search Command**: `Keeper: Search Secrets`
- **Copy Secret Command**: `Keeper: Copy Secret Field`

Success Criteria:

- All commands work without errors
- Command palette integration functions properly
- Basic error handling for failed operations

3.1 Core Features (Phase 1)

3.1.1 Authentication & Configuration

User Story: As a developer, I want to quickly authenticate with Keeper without complex setup.

Requirements:

- **One-Time Token (OTT)** authentication flow
- **Regional support** (US, EU, AU, GOV, JP, CA)
- **Multi-profile management** (dev, staging, prod)
- **Multiple auth methods**: Config file, Base64 config, In-memory config
- **Environment variable support** (`KSM_CONFIG` , `KEEPER_SECRETS_MANAGER_CONFIG_BASE64`)
- **Secure credential storage** using VS Code's SecretStorage API
- **SSL certificate verification** with configurable validation
- **Automatic token refresh** and rotation

Success Criteria:

- Setup completion in <2 minutes
- Support for all Keeper regions
- Secure token management with zero plaintext storage
- Fallback authentication methods for different environments

3.1.2 Secret Discovery & Management

User Story: As a developer, I want to browse and manage my secrets directly in VS Code.

Requirements:

- **Secret Explorer**: Tree view showing folders and secrets
- **Record Operations**: Complete CRUD operations (Create, Read, Update, Delete)
- **Bulk Operations**: Multi-select for batch actions on multiple records
- **Search Interface**: Basic filtering by title and UID
- **Quick Actions**: Copy, edit, delete secrets with contextual menus
- **Metadata Display**: Show secret age, last modified, record type
- **Hierarchical Folder Support**: Nested folder structure with parent-child relationships

- **Record Type Support**: Login, Bank Account, Server Credentials, SSH Keys, etc.

Success Criteria:

- Load 1000+ secrets in <3 seconds
- Intuitive folder navigation with unlimited nesting
- Basic search by title and UID
- Support for all 30+ record types from KSM

3.1.3 Keeper Notation Integration

User Story: As a developer, I want to use Keeper notation in my code with full IDE support.

Requirements:

- **Enhanced Notation System**: Full `${keeper://[UID]/field/[field_name]}` syntax support with template substitution
- **Field Access Patterns**: Direct field value retrieval with indexing
- **Custom Field Access**: Custom field addressing with label-based lookup
- **File Access Notation**: File attachment notation (`${keeper://[UID]/file/[filename]}`)
- **Index Support**: Array/list indexing (`${keeper://[UID]/field/url[0]}`)
- **Template Substitution**: Process notation within strings and configuration files
- **Syntax Highlighting**: Highlight `${keeper://...}` notation in files
- **Autocomplete**: Basic suggestions for UIDs and field names when typing `${keeper://`
- **Hover Provider**: Show secret metadata and field values on hover over `${keeper://...}`
- **Link Provider**: Navigate to Keeper vault with deep linking
- **Validation**: Basic notation syntax validation for `${keeper://...}` expressions
- **Multi-language Support**: Works across Python, JavaScript, Java, configuration files

Success Criteria:

- Notation autocomplete in <200ms
- 95% accuracy in secret resolution
- Support for core notation formats
- Basic validation with error messages

3.1.4 Secret Detection & Replacement

User Story: As a developer, I want to automatically detect hardcoded secrets and replace them with Keeper notation.

Requirements:

- **Pattern Recognition**: Detect API keys, passwords, tokens
- **CodeLens Suggestions**: Show "Save in Keeper" above detected secrets
- **Bulk Detection**: Scan entire project for secrets
- **Smart Replacement**: Suggest appropriate `${keeper://...}` notation format
- **Undo Support**: Revert notation to original values

Success Criteria:

- 95% accuracy in secret detection
- <5% false positive rate
- Support for 20+ secret patterns

3.2 Advanced Features (Phase 2)

3.2.1 Comprehensive Field Type Support

User Story: As a developer, I want to work with all types of secrets and structured data.

Requirements:

- **30+ Field Types**: Text, URL, Login, Password, Email, Phone, Date, Pin Code, etc.
- **Complex Field Types**: Address, Payment Card, Bank Account, Key Pair, OTP, Passkey
- **Security Fields**: Security Questions, One-Time Codes, Hidden Fields, Secure Notes
- **Enterprise Fields**: License Numbers, Account Numbers, PAM credentials
- **Structured Data**: Multi-value fields with indexing and property access
- **Field Validation**: Type-specific validation and formatting
- **Field Templates**: Pre-configured field layouts for common use cases

3.2.2 Advanced File Management

User Story: As a developer, I want to manage secret files and certificates in VS Code.

Requirements:

- **File Upload**: Drag-and-drop file attachment to secrets
- **File Preview**: View file contents in VS Code (certificates, keys, configs)
- **Download**: Export files to local filesystem
- **Multiple File Attachments**: Support for multiple files per record
- **File Metadata**: Display file properties and information
- **Certificate Management**: Special handling for SSL/TLS certificates
- **Key File Management**: SSH keys, API keys, cryptographic keys
- **File Type Detection**: Automatic file type recognition and handling

3.2.3 Password Generation & Security

User Story: As a developer, I want to generate secure passwords with custom policies.

Requirements:

- **Custom Policies**: Length, character sets, complexity rules
- **Quick Generation**: Generate from command palette
- **Bulk Generation**: Create multiple passwords at once
- **Policy Templates**: Save and reuse generation policies
- **Strength Indicator**: Visual password strength feedback
- **TOTP/OTP Support**: Time-based and HMAC-based one-time passwords
- **Passkey Integration**: WebAuthn credential management

3.2.4 Advanced Folder & Organization

User Story: As a team lead, I want to organize secrets efficiently with advanced folder features.

Requirements:

- **Nested Folder Support**: Unlimited folder hierarchy depth
- **Folder Permissions**: Granular access control per folder
- **Bulk Folder Operations**: Mass folder creation, deletion, reorganization
- **Folder Templates**: Pre-configured folder structures
- **Smart Folders**: Dynamic folders based on queries/filters
- **Folder Metadata**: Tags, descriptions, and custom properties

3.2.5 Template & Code Generation

User Story: As a developer, I want to generate templates and sample code for KSM integrations.

Requirements:

- **KSM SDK Sample Code**: Generate sample code for all KSM SDKs
  - JavaScript/TypeScript samples (authentication, CRUD, notation)
  - Python samples (CLI-style operations, SDK usage)
  - Java samples (Spring Boot integration, basic operations)
  - .NET samples (ASP.NET Core, console applications)
  - Go samples (web service integration, CLI tools)
- **CI/CD Templates**: Generate infrastructure templates
  - GitHub Actions workflows using KSM GitHub Action
  - Terraform files using terraform-provider-secretsmanager
  - Ansible playbooks using keeper_secrets_manager_ansible
  - Docker/docker-compose files with KSM integration
  - Kubernetes manifests with external-secrets integration
- **Integration Patterns**: Common integration scenarios
  - Web application authentication flows
  - Database connection string management
  - API key management patterns
  - Certificate and SSH key handling
- **Template Customization**: Modify templates before insertion
- **Language Detection**: Auto-detect project language and suggest appropriate samples

Success Criteria:

- Generate working code samples for all supported SDKs
- Templates use correct KSM notation and configuration
- Code samples follow language-specific best practices
- Easy customization and insertion into projects

3.3 Enhanced Features (Phase 3)

3.3.1 Advanced File Management

User Story: As a developer, I want to manage certificates and key files efficiently.

Requirements:

- **Certificate Management**: View and manage SSL/TLS certificates
- **Key File Management**: SSH keys, API keys, cryptographic keys
- **File Preview**: View file contents in VS Code when safe
- **Multiple File Support**: Handle multiple file attachments per record
- **File Metadata**: Display file properties and information
- **Download Integration**: Save files to local filesystem

3.3.2 Advanced Code Generation

User Story: As a developer, I want advanced code generation with real secret data.

Requirements:

- **Live Code Generation**: Generate code using actual secrets from vault
- **Custom Code Templates**: Create and save custom code templates
- **Multi-Language Support**: Generate equivalent code across different languages
- **Integration Testing**: Generate test code with mock data

- **Documentation Generation**: Auto-generate API documentation from secrets
- **Migration Tools**: Generate migration scripts from other secret managers
- **Code Validation**: Validate generated code syntax and structure

3.3.3 Developer Experience

User Story: As a developer, I want an intuitive and helpful development experience.

Requirements:

- **Smart Defaults**: Sensible default configurations
- **Guided Setup**: Step-by-step configuration wizard
- **Contextual Help**: Inline help and tooltips
- **Error Recovery**: Helpful error messages and recovery suggestions
- **Quick Actions**: Keyboard shortcuts and quick commands
- **Status Indicators**: Clear connection and sync status

3.3.4 SDK Integration Showcase

User Story: As a developer, I want to see comprehensive examples of how to use KSM across different technologies.

Requirements:

- **Interactive Examples**: Live, runnable code examples in the extension
- **Best Practices**: Showcase recommended patterns for each SDK
- **Performance Examples**: Demonstrate efficient KSM usage patterns
- **Error Handling**: Show robust error handling across all SDKs
- **Security Examples**: Demonstrate secure implementation patterns
- **Framework Integration**: Show integration with popular frameworks (Express, Spring, Django, etc.)
- **Real-world Scenarios**: Complete application examples using KSM

3.4 Enterprise Features (Phase 4)

3.4.1 Enterprise Template Management

User Story: As an enterprise architect, I want to manage templates across the organization.

Requirements:

- **Template Distribution**: Centralized template management
- **Organization Standards**: Enforce organizational coding standards
- **Template Approval**: Review and approval workflow for templates
- **Usage Analytics**: Track template usage and adoption
- **Template Governance**: Version control and compliance tracking

3.4.2 Advanced Configuration Management

User Story: As an enterprise admin, I want centralized configuration management.

Requirements:

- **Configuration Policies**: Centralized configuration enforcement
- **Multi-Workspace Support**: Manage multiple VS Code workspaces
- **Team Settings**: Shared extension settings for teams
- **Configuration Backup**: Backup and restore configurations
- **Update Management**: Controlled extension updates and rollbacks

---

4. Technical Architecture

4.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────┐
│                    VS Code Extension                    │
├─────────────────────────────────────────────────────────┤
│  UI Layer                                               │
│  ├─ Commands & Menus                                    │
│  ├─ Tree Views & Panels                                 │
│  ├─ Language Providers                                  │
│  └─ Status Bar & Notifications                          │
├─────────────────────────────────────────────────────────┤
│  Service Layer                                          │
│  ├─ KSM Service Wrapper                                 │
│  ├─ Configuration Manager                               │
│  ├─ Cache Manager                                       │
│  └─ Security Manager                                    │
├─────────────────────────────────────────────────────────┤
│  KSM JavaScript SDK                                     │
│  ├─ SecretManager Client                                │
│  ├─ Notation Parser                                     │
│  ├─ Crypto Operations                                   │
│  └─ Storage Abstraction                                 │
├─────────────────────────────────────────────────────────┤
│  Storage Layer                                          │
│  ├─ VS Code Settings                                    │
│  ├─ Secure Storage API                                  │
│  └─ Local Cache                                         │
└─────────────────────────────────────────────────────────┘
```

4.2 Key Components

4.2.1 KSM Service Wrapper

```
class KSMService {
  private client: SecretsManager;
  private cache: SecretCache;
  private config: ConfigurationManager;

  async authenticate(token: string): Promise<void>
  async getSecrets(filter?: SecretFilter): Promise<KeeperRecord[]>
  async createSecret(data: SecretData): Promise<string>
  async updateSecret(uid: string, data: SecretData): Promise<void>
  async deleteSecret(uid: string): Promise<void>
  async resolveNotation(notation: string): Promise<string>
}
```

4.2.2 Configuration Manager

```
class ConfigurationManager {
  private profiles: Map<string, ProfileConfig>;
  private activeProfile: string;

  async createProfile(name: string, config: ProfileConfig): Promise<void>
  async switchProfile(name: string): Promise<void>
  async getActiveProfile(): Promise<ProfileConfig>
  async validateConfiguration(): Promise<boolean>
}
```

4.2.3 Language Providers

```
class KeeperNotationProvider implements vscode.CompletionItemProvider {
  provideCompletionItems(document: vscode.TextDocument, position: vscode.Position): vscode.CompletionItem[]
}

class KeeperHoverProvider implements vscode.HoverProvider {
  provideHover(document: vscode.TextDocument, position: vscode.Position): vscode.Hover
}
```

4.3 Performance Requirements

Metric	Target	Measurement
Extension Load Time	<2 seconds	From activation to ready state
Secret Retrieval	<100ms	Single secret fetch
Notation Resolution	<50ms	Hover provider response
Search Response	<200ms	Search results display
Memory Usage	<50MB	Peak memory consumption

4.4 Security Architecture

4.4.1 Authentication Flow

1. **Token Input**: User provides one-time token
2. **Token Validation**: Verify token with Keeper servers
3. **Key Exchange**: Establish secure session keys
4. **Credential Storage**: Store in VS Code SecretStorage
5. **Token Refresh**: Automatic token renewal

4.4.2 Secret Handling

- **In-Memory Only**: Secrets never persisted to disk
- **Encrypted Communication**: All API calls use TLS 1.3
- **Secure Clipboard**: Automatic clipboard clearing
- **Audit Logging**: Track all secret access
- **Zero Trust**: Verify every operation

---

5. Primary Use Case: Python Developer Scenario

5.1 The Main Problem We're Solving

Current State: Python developers have hardcoded secrets scattered throughout their code, creating security vulnerabilities and maintenance nightmares.

Our Solution: Transform hardcoded secrets into secure, maintainable Keeper notation with minimal friction.

5.2 Complete Python Developer Workflow

5.2.1 Before: Insecure Code with Hardcoded Secrets

```
# config.py - BAD: Hardcoded secrets everywhere
DATABASE_URL = "postgresql://user:mypassword123@localhost:5432/myapp"
STRIPE_API_KEY = "sk_test_51H7x2xABC123..."
JWT_SECRET = "super-secret-jwt-key-dont-share"
REDIS_PASSWORD = "redis-password-here"

# app.py - More hardcoded secrets
import stripe
import psycopg2

stripe.api_key = "sk_test_51H7x2xABC123..."  # Hardcoded again!
conn = psycopg2.connect("postgresql://user:mypassword123@localhost:5432/myapp")
```

5.2.2 Step 1: Extension Detects Hardcoded Secrets

- **Visual Indicators**: Hardcoded secrets highlighted in red
- **CodeLens Actions**: `Save in Keeper` appears above each detected secret
- **Bulk Detection**: Scan entire project with `Keeper: Scan for Secrets` command

5.2.3 Step 2: Developer Saves Secrets to Keeper

1. Click `Save in Keeper` above `"mypassword123"`
2. Extension dialog: "Save 'mypassword123' to Keeper?"
3. Developer selects folder: "Development > Database"
4. Extension creates new record with password field
5. Returns UID: `k8x9m2n1-abc3-def4-ghi5-jkl6mno7pqr8`

5.2.4 Step 3: Auto-Replacement with Secure Notation

```
# config.py - GOOD: Using Keeper notation
DATABASE_URL = "postgresql://user:${keeper://k8x9m2n1-abc3-def4-ghi5-jkl6mno7pqr8/field/password}@localhost:5432/myapp"
STRIPE_API_KEY = "${keeper://a1b2c3d4-efgh-ijkl-mnop-qrstuvwxyz12/field/password}"
JWT_SECRET = "${keeper://f7g8h9i0-jklm-nopq-rstu-vwxyz1234567/field/password}"
REDIS_PASSWORD = "${keeper://m3n4o5p6-qrst-uvwx-yz12-3456789abcde/field/password}"
```

5.2.5 Step 4: Enhanced Development Experience

```
# app.py - With KSM Python SDK integration
from keeper_secrets_manager_core import SecretsManager
import os

# Extension can generate this boilerplate code
secrets_manager = SecretsManager(config=os.environ['KSM_CONFIG'])

# Hover over notation shows actual values in VS Code
stripe_key = secrets_manager.resolve_notation("${keeper://a1b2c3d4-efgh-ijkl-mnop-qrstuvwxyz12/field/password}")
stripe.api_key = stripe_key

# Direct notation resolution
DATABASE_URL = secrets_manager.resolve_notation("postgresql://user:${keeper://k8x9m2n1-abc3-def4-ghi5-jkl6mno7pqr8/field/password}@l
conn = psycopg2.connect(DATABASE_URL)
```

5.3 Developer Experience Benefits

Before	After
30+ hardcoded secrets in files	All secrets in Keeper vault
Secrets committed to git	`${keeper://...}` notation is git-safe
Manual secret updates in multiple places	Update once in Keeper, used everywhere

Before	After
No secret visibility control	Keeper permissions and sharing
Security vulnerabilities	Enterprise-grade security
Difficult onboarding	New developers get Keeper access

5.4 VS Code Integration Features

5.4.1 Visual Feedback

- **Syntax Highlighting**: `${keeper://...}` highlighted in distinct color
- **Hover Provider**: Show actual secret values on hover (with permissions)
- **Status Indicators**: Connection status and secret sync status

5.4.2 Developer Productivity

- **Autocomplete**: Type `${keeper://` and get UID/field suggestions
- **Quick Actions**: CodeLens to edit secret, copy value, or navigate to vault
- **Validation**: Real-time validation with error highlighting
- **Bulk Operations**: Process multiple secrets at once

5.4.3 Code Generation

- **SDK Templates**: Generate Python KSM SDK boilerplate
- **Integration Patterns**: Common patterns for Flask, Django, FastAPI
- **Test Code**: Generate test code with mock secrets

5.5 Success Stories & User Testimonials

5.5.1 Python Developer Success Story

"As a Python developer working on a fintech application, I had over 40 hardcoded secrets scattered across my Flask application. The KSM VS Code extension detected all of them in seconds, and I was able to secure every single one with just a few clicks. Now my code is git-safe, and I can update database passwords without touching the code. The `${keeper://...}` notation makes it clear where secrets come from, and the hover feature lets me verify values instantly. Setup took 2 minutes, and I secured my entire codebase in under 20 minutes. This extension transformed how I handle secrets."

5.5.2 DevOps Engineer Success Story

"As a DevOps engineer managing 15+ microservices, I was constantly dealing with secret sprawl across Docker files, Kubernetes manifests, and configuration files. The KSM extension's template generation feature is a game-changer - I can generate properly configured Terraform files, Ansible playbooks, and GitHub Actions workflows that use KSM out of the box. The `${keeper://...}` notation works seamlessly in YAML, JSON, and environment files. What used to take hours of manual configuration now takes minutes with generated templates."

5.5.3 Security Engineer Success Story

"As a security engineer, I was constantly finding hardcoded secrets in code reviews and security scans. The KSM VS Code extension eliminated this problem entirely. Developers can't accidentally commit secrets because they're automatically converted to safe notation. The extension respects our Keeper permissions, provides audit trails, and integrates with our existing security workflows. False positives dropped to near zero, and our security posture improved dramatically."

5.5.4 Team Lead Success Story

"As a team lead managing 8 developers across 3 time zones, secret management was a constant headache. Different developers handled secrets differently, and onboarding new team members took days. With the KSM extension, we have standardized secret management across all projects. New developers install the extension, authenticate with their Keeper account, and they're productive immediately. The shared templates ensure consistency, and the notation system makes code reviews much easier."

5.5.5 Full-Stack Developer Success Story

"As a full-stack developer working with React, Node.js, and Python, I was juggling secrets across multiple languages and frameworks. The KSM extension's multi-language support is incredible - the same `${keeper://...}` notation works in my JavaScript configs, Python environment files, and Docker compose files. The SDK code generation feature gave me working examples for each language, and the integration patterns helped me implement best practices across my entire stack."

5.6 Success Metrics for Python Developers

- **Security**: Zero secrets in git commits
- **Productivity**: <30 seconds to secure a hardcoded secret
- **Maintenance**: 90% reduction in secret-related bugs
- **Onboarding**: New developers productive in <5 minutes
- **Adoption**: 80% of hardcoded secrets converted within first week

---

6. User Experience Design

5.1 Ease of Use Principles

Based on competitor analysis, our extension will prioritize:

5.1.1 Zero-Friction Setup

Problem: 1Password requires CLI installation, complex setup **Solution**:

- Single token authentication
- Automatic region detection
- Guided setup wizard
- Pre-configured templates

5.1.2 Intuitive Discovery

Problem: Doppler limits to environment variables **Solution**:

- Visual secret browser
- Smart search with filters
- Folder-based organization
- Recent secrets quick access

5.1.3 Contextual Integration

Problem: Vault extension requires external knowledge **Solution**:

- Inline CodeLens suggestions
- Hover-based secret preview
- Automatic pattern recognition
- Smart autocomplete

5.2 User Interface Design

5.2.1 Secret Explorer Panel

```
┌─────────────────────────────────────┐
│ 🔐 KEEPER SECRETS                   │
├─────────────────────────────────────┤
│ 📁 Development/                     │
│   ├─ 🔑 API Keys                    │
│   │   ├─ GitHub Token               │
│   │   └─ Stripe API Key             │
│   └─ 🗄 Database                    │
│       └─ PostgreSQL Credentials     │
├─────────────────────────────────────┤
│ 📁 Production/                      │
│   ├─ 🔑 API Keys                    │
│   └─ 🗄 Database                    │
├─────────────────────────────────────┤
│ [🔍 Search] [⚙ Settings] [🔄 Sync] │
└─────────────────────────────────────┘
```

5.2.2 Command Palette Integration

- `Keeper: Authenticate` - Initial setup
- `Keeper: Search Secrets` - Find secrets
- `Keeper: Insert Notation` - Add secret reference
- `Keeper: Generate Password` - Create new password
- `Keeper: Scan for Secrets` - Detect hardcoded secrets

5.2.3 Status Bar Integration

```
KSM: Connected (Dev) | 🔐 3 secrets found
```

5.3 Workflow Examples

5.3.1 First-Time Setup

1. Install extension from marketplace
2. Run "Keeper: Authenticate" command

3. Paste one-time token
4. Select region (auto-detected)
5. Choose default profile
6. Extension ready for use

5.3.2 Working with Secrets

1. Open JavaScript file
2. Type `process.env.`
3. See autocomplete with secret suggestions
4. Select secret, notation inserted
5. Hover over notation to see value
6. Click to navigate to Keeper vault

5.3.3 Secret Detection

1. Open file with hardcoded API key
2. CodeLens appears: "Save in Keeper"
3. Click to save secret
4. Choose folder and field type
5. Original value replaced with notation
6. Secret safely stored in Keeper

---

6. Technical Specifications

6.1 Development Stack

6.1.1 Core Technologies

- **Language**: TypeScript 5.0+
- **Framework**: VS Code Extension API 1.80+
- **SDK**: Keeper Secrets Manager JavaScript SDK
- **Build**: webpack 5.0 with TypeScript loader
- **Testing**: Jest + VS Code Test Runner
- **Linting**: ESLint + Prettier

6.1.2 Key Dependencies

```
{
  "dependencies": {
    "@keeper-security/secrets-manager-core": "^1.0.0",
    "vscode": "^1.80.0"
  },
  "devDependencies": {
    "@types/vscode": "^1.80.0",
    "typescript": "^5.0.0",
    "webpack": "^5.0.0",
    "jest": "^29.0.0"
  }
}
```

6.2 File Structure

```
src/
├── commands/           # Command implementations
│   ├── authenticate.ts
│   ├── search.ts
│   └── generate.ts
├── providers/          # Language providers
│   ├── completion.ts
│   ├── hover.ts
│   └── codelens.ts
├── services/           # Core services
│   ├── ksm.ts
│   ├── config.ts
│   └── cache.ts
├── ui/                 # User interface
│   ├── explorer.ts
│   ├── panels.ts
│   └── statusbar.ts
├── utils/              # Utilities
│   ├── notation.ts
│   ├── patterns.ts
│   └── security.ts
└── extension.ts        # Main entry point
```

6.3 Extension Manifest

```
{
  "name": "keeper-secrets-manager",
  "displayName": "Keeper Secrets Manager",
  "description": "Enterprise-grade secrets management for VS Code",
  "version": "1.0.0",
  "publisher": "keeper-security",
  "engines": {
    "vscode": "^1.80.0"
  },
  "categories": ["Other"],
  "activationEvents": [
    "onLanguage:javascript",
    "onLanguage:typescript",
    "onLanguage:python",
    "onCommand:keeper.authenticate"
  ],
  "contributes": {
    "commands": [
      {
        "command": "keeper.authenticate",
        "title": "Authenticate",
        "category": "Keeper"
      }
    ],
    "views": {
      "explorer": [
        {
          "id": "keeperSecrets",
          "name": "Keeper Secrets",
          "when": "keeper:authenticated"
        }
      ]
    }
  }
}
```

7. Quality Assurance

7.1 Testing Strategy

7.1.1 Unit Testing

- **Coverage**: 90%+ code coverage
- **Framework**: Jest with TypeScript
- **Mocking**: Mock VS Code API and KSM SDK
- **CI/CD**: Automated testing on PR

7.1.2 Integration Testing

- **End-to-End**: Test complete workflows
- **VS Code Integration**: Test extension lifecycle
- **SDK Integration**: Test KSM SDK operations
- **Multi-Platform**: Test on Windows, macOS, Linux

7.1.3 Security Testing

- **Penetration Testing**: Third-party security audit
- **Vulnerability Scanning**: Automated dependency scanning
- **Secret Detection**: Ensure no secrets in code
- **Compliance**: SOC2, HIPAA validation

7.2 Performance Testing

7.2.1 Load Testing

- **Large Vaults**: Test with 10,000+ secrets
- **Memory Usage**: Monitor memory consumption
- **Response Times**: Measure API latency
- **Concurrent Users**: Test multiple profiles

7.2.2 Stress Testing

- **Network Failures**: Test offline scenarios
- **API Failures**: Test error handling
- **Resource Limits**: Test memory/CPU limits
- **Recovery**: Test disaster recovery

---

8. Deployment & Distribution

8.1 Release Strategy

8.1.1 Beta Release

- **Audience**: Internal teams + select customers
- **Duration**: 4 weeks
- **Features**: Core MVP features
- **Feedback**: Collect user feedback and metrics

8.1.2 Public Release

- **Platform**: VS Code Marketplace
- **Marketing**: Developer blog posts, demos
- **Support**: Documentation, tutorials
- **Monitoring**: Usage analytics, error tracking

8.2 Maintenance & Updates

8.2.1 Regular Updates

- **Frequency**: Monthly feature updates
- **Security**: Weekly security patches
- **Dependencies**: Keep SDK updated
- **Bug Fixes**: Rapid response to issues

8.2.2 Long-term Support

- **Compatibility**: Support latest VS Code versions

- **Migration**: Smooth upgrade paths
- **Documentation**: Keep docs updated
- **Community**: Active community support

---

9. Success Metrics & KPIs

9.1 Adoption Metrics

- **Downloads**: 50,000+ in first 6 months
- **Active Users**: 10,000+ monthly active users
- **Retention**: 80% month-over-month retention
- **Growth Rate**: 20% monthly growth

9.2 User Satisfaction

- **Marketplace Rating**: 4.8+ stars
- **User Reviews**: 90% positive reviews
- **NPS Score**: 50+ Net Promoter Score
- **Support Tickets**: <2% of users file tickets

9.3 Performance Metrics

- **Load Time**: <2 seconds average
- **API Response**: <100ms average
- **Error Rate**: <0.1% error rate
- **Uptime**: 99.9% availability

9.4 Security Metrics

- **Zero Incidents**: No security breaches
- **Vulnerability Response**: <24 hours
- **Compliance**: 100% audit compliance
- **Penetration Testing**: Pass quarterly tests

---

10. Technical Implementation Notes

10.1 Comprehensive KSM Feature Coverage

10.1.1 Complete Authentication Support

- **All Authentication Methods**: OTT, config file, Base64, in-memory, environment variables
- **Regional Support**: Full support for all Keeper regions (US, EU, AU, GOV, JP, CA)
- **Multi-Profile Management**: Support for multiple environments and configurations
- **Enterprise Security**: SSL verification, key management, digital signatures

10.1.2 Full Record & Field Type Support

- **30+ Field Types**: Complete support for all KSM field types
- **All Record Types**: Login, Server Credentials, SSH Keys, Certificates, PAM, etc.
- **Complex Data Structures**: Multi-value fields, indexing, property access
- **Field Validation**: Type-specific validation and formatting

10.1.3 Advanced Notation System

- **Complete Notation Support**: Full `keeper://` syntax with all addressing patterns
- **Index Support**: Array indexing (`[0]`, `[1]`) and property access
- **File Notation**: Direct file attachment addressing
- **Custom Fields**: Label-based custom field access
- **Escape Sequences**: Special character handling

10.1.4 Comprehensive File Management

- **Multiple File Types**: Support for all file types (certificates, keys, configs, docs)
- **File Operations**: Complete CRUD operations for file attachments
- **File Metadata**: Full file property and information support
- **Bulk Operations**: Mass file upload/download capabilities

10.2 Key Differentiators from Competitors

10.2.1 vs 1Password

- **No CLI Dependency**: Direct SDK integration eliminates installation friction
- **Advanced Notation**: Keeper's notation system is more powerful than `op://`
- **Offline Capability**: Cached secrets work without internet
- **Enterprise Features**: Built-in compliance and audit capabilities
- **30+ Field Types**: vs 1Password's limited field support
- **File Management**: Native file attachment support

10.2.2 vs Doppler

- **Self-Hosted Option**: Support for on-premises deployments
- **File Management**: Native file attachment support
- **Complex Data Types**: Support for structured data beyond env vars
- **Advanced Search**: Powerful filtering and search capabilities
- **Complete Field Types**: 30+ field types vs environment variables only
- **Folder Management**: Hierarchical folder structure with permissions

10.2.3 vs HashiCorp Vault

- **Simple Setup**: No complex infrastructure required
- **User-Friendly**: Developer-focused UX vs admin-focused
- **Native Integration**: Direct VS Code integration
- **Managed Service**: No operational overhead
- **Field Type System**: Rich field types vs key-value only
- **File Support**: Native file attachment handling

10.3 Keeper Notation Syntax Specification

10.3.1 Enhanced Notation Format

The VS Code extension uses an enhanced notation format that wraps the traditional Keeper notation in template substitution syntax:

Format: `${keeper://[UID]/field/[field_name]}`

Key Benefits:

- **Clear Boundaries**: `${}` clearly separates notation from surrounding text
- **Template Substitution**: Runtime processing replaces notation with actual values
- **Multi-language Support**: Works in Python, JavaScript, Java, configuration files
- **IDE Integration**: Easy parsing for syntax highlighting and autocomplete

10.3.2 Notation Examples

Basic Field Access:

```
# Database connection
DATABASE_URL = "postgresql://user:${keeper://k8x9m2n1-abc3-def4-ghi5-jkl6mno7pqr8/field/password}@localhost:5432/myapp"

# API Keys
STRIPE_API_KEY = "${keeper://a1b2c3d4-efgh-ijkl-mnop-qrstuvwxyz12/field/password}"
GITHUB_TOKEN = "${keeper://f7g8h9i0-jklm-nopq-rstu-vwxyz1234567/field/password}"
```

Custom Field Access:

```
// Custom field by label
const apiEndpoint = "${keeper://record-uid/custom_field/API_Endpoint}";
const clientId = "${keeper://record-uid/custom_field/Client_ID}";
```

Array/List Indexing:

```
// Multiple URLs with indexing
String primaryUrl = "${keeper://record-uid/field/url[0]}";
String backupUrl = "${keeper://record-uid/field/url[1]}";
```

File Access:

```
# Docker compose with certificates
version: '3.8'
services:
  web:
    image: nginx
    volumes:
      - "${keeper://cert-record/file/server.crt}:/etc/ssl/certs/server.crt"
      - "${keeper://cert-record/file/server.key}:/etc/ssl/private/server.key"
```

Configuration Files:

```
{
  "database": {
    "host": "localhost",
    "username": "myapp",
    "password": "${keeper://db-record/field/password}",
    "ssl_cert": "${keeper://db-record/file/client.crt}"
  },
  "redis": {
    "password": "${keeper://redis-record/field/password}",
    "host": "${keeper://redis-record/field/host}"
  }
}
```

10.3.3 Processing Flow

1. **Detection**: Extension scans for `${keeper://...}` patterns
2. **Parsing**: Extract UID, field type, and field name
3. **Validation**: Verify notation syntax and secret existence
4. **Resolution**: Replace with actual secret values at runtime
5. **Caching**: Cache resolved values for performance

10.3.4 Runtime Processing

```
# Example SDK integration
from keeper_secrets_manager_core import SecretsManager

secrets_manager = SecretsManager(config=config)

# Original string with notation
original = "postgresql://user:${keeper://k8x9m2n1-abc3-def4-ghi5-jkl6mno7pqr8/field/password}@localhost:5432/myapp"

# Process notation
processed = secrets_manager.resolve_notation(original)
# Result: "postgresql://user:actualpassword123@localhost:5432/myapp"
```

10.3.5 VS Code Integration Features

- **Syntax Highlighting**: `${keeper://...}` highlighted in distinct color
- **Autocomplete**: Trigger on `${keeper://` with UID and field suggestions
- **Hover Provider**: Show secret metadata and field values on hover
- **Validation**: Real-time validation with error highlighting
- **Quick Actions**: CodeLens to edit secret, copy value, or navigate to vault

10.3.6 Security Considerations

- **No Plaintext Storage**: Secrets never stored in plaintext in files
- **Git Safety**: `${keeper://...}` notation is safe to commit
- **Access Control**: Honors Keeper vault permissions
- **Audit Trail**: All secret access logged in Keeper
- **Encryption**: All secret values encrypted in transit and at rest

10.4 Robust Architecture Principles

10.4.1 Error Handling

```
class RobustKSMService {
  async getSecret(uid: string): Promise<KeeperRecord> {
    try {
      return await this.client.getSecret(uid);
    } catch (error) {
      if (error instanceof NetworkError) {
        // Try cache first
        const cached = await this.cache.get(uid);
        if (cached) return cached;

        // Retry with exponential backoff
        return await this.retryWithBackoff(() => this.client.getSecret(uid));
      }

      this.logger.error('Failed to get secret', error);
      throw new KSMError('Secret retrieval failed', error);
    }
  }
}
```

10.3.2 Caching Strategy

- **Multi-Level Cache**: Memory + persistent cache for resilience
- **TTL-Based**: Automatic cache expiration with configurable timeouts
- **Invalidation**: Smart cache invalidation on record updates
- **Offline Support**: Graceful degradation with cached secrets
- **Performance Optimization**: Bulk operations and lazy loading

10.3.3 Security Hardening

- **Input Validation**: Sanitize all user inputs and notation
- **Rate Limiting**: Prevent API abuse and DoS attacks
- **Audit Logging**: Track all operations and access patterns
- **Secure Defaults**: Secure by default configuration
- **Encryption**: End-to-end encryption with zero-knowledge architecture
- **Token Management**: Secure token storage and automatic rotation

10.4 Ease of Use Implementation

10.4.1 Smart Defaults

```
const defaultConfig = {
  region: 'auto-detect',
  profile: 'default',
  cacheTimeout: 300000, // 5 minutes
  notationFormat: 'keeper://',
  secretDetection: true,
  autoComplete: true
};
```

10.4.2 Guided Setup

```
class SetupWizard {
  async runSetup(): Promise<void> {
    const token = await this.promptForToken();
    const region = await this.detectRegion(token);
    const profile = await this.createProfile(region);

    await this.testConnection(profile);
    await this.showWelcome();
  }
}
```

10.4.3 Contextual Help

- **Tooltips**: Explain complex features
- **Inline Help**: Context-sensitive guidance
- **Quick Start**: Interactive tutorials
- **Error Messages**: Helpful error explanations

---

11. Conclusion

This PRD outlines a comprehensive approach to building the most advanced and user-friendly secrets management VS Code extension. By leveraging Keeper's powerful SDK and notation system while focusing on ease of use and robust architecture, we can create a product that significantly surpasses existing competitors.

The key to success will be:

1. **Zero-friction setup** - No external dependencies
2. **Intuitive UX** - Natural developer workflows
3. **Enterprise security** - Uncompromising security standards
4. **Robust performance** - Fast, reliable, and scalable
5. **Comprehensive features** - Beyond basic secret management

With this foundation, the Keeper Secrets Manager VS Code extension will become the definitive tool for developer secret management, driving adoption and strengthening Keeper's position in the enterprise security market.