PySchemia-Crypto Repository Structure & Deployment Standard

Overview

The `pyschemia-crypto` package implements the **OBINexus Cryptographic Interoperability Standard v1.0** through a non-monolithic architecture that separates cryptographic primitive versions while maintaining backward compatibility and forward evolution paths.

Directory Structure

```
pyschemia_crypto/
├── legacy/
│   ├── rsa_2048/
│   │   ├── __init__.py
│   │   ├── config.py          # Canonical pattern definitions
│   │   └── pattern.py         # Enforcement logic implementation
│   ├── aes_128/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── pattern.py
│   └── __init__.py
├── stable/
│   ├── rsa_3072/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── pattern.py
│   ├── aes_256/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── pattern.py
│   └── __init__.py
├── experimental/
│   ├── rsa_4096/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── pattern.py
│   ├── ecdsa_p384/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   └── pattern.py
│   └── __init__.py
├── common/
│   ├── __init__.py
│   ├── normalize.py           # USCN implementation
│   ├── automaton.py          # State machine core
│   ├── audit.py              # Secure logging framework
│   └── validators.py         # Cross-language regex validation
├── tests/
│   ├── __init__.py
│   ├── test_patterns.py      # Pattern matching validation
│   ├── test_normalization.py # USCN compliance tests
│   ├── test_audit.py         # Audit trail verification
│   └── test_cross_language.py # Python/Lua/C compatibility
├── .github/
│   └── workflows/
│       ├── ci.yml            # Continuous integration
```

```
│       ├── release.yml       # PyPI deployment automation
│       └── security.yml      # Security compliance checks
├── docs/
│   ├── api_reference.md
│   ├── implementation_guide.md
│   └── compliance_matrix.md
├── setup.py                  # Package configuration
├── setup.cfg                 # Build configuration
├── pyproject.toml           # Modern Python packaging
├── .gitignore
├── .gitattributes
├── README.md
├── CHANGELOG.md
├── LICENSE
└── __init__.py
```

Package Version Strategy

PyPI Release Channels

Version Pattern	Target Audience	Deployment Channel
`1.0.0-legacy`	Legacy system compatibility	PyPI stable
`1.0.0-stable`	Production deployments	PyPI stable
`1.0.0-experimental`	Research & development	PyPI pre-release
`1.0.0-rc1`	Release candidates	PyPI pre-release

Git Branch Strategy

Branch	Purpose	Automation
`main`	Stable release source	Auto-deploy to PyPI stable
`legacy`	Legacy compatibility	Manual PyPI deployment
`experimental`	Development features	Auto-deploy to PyPI pre-release
`release/*`	Release preparation	Automated testing & staging

Implementation Architecture

Core Pattern Definition Structure

Each cryptographic primitive follows this standardized implementation pattern:

python

```python
# Example: stable/rsa_3072/config.py
"""RSA-3072 Cryptographic Primitive Configuration"""

from dataclasses import dataclass
from typing import Pattern
import re

@dataclass(frozen=True)
class RSA3072Config:
    """RSA-3072 primitive configuration following OBINexus v1.0 standard."""

    algorithm: str = "RSA-3072"
    key_size: int = 3072
    pattern: Pattern = re.compile(r"^RSA-3072:[a-f0-9]{768}$")
    security_level: str = "stable"
    compatibility_matrix: dict = None

    def __post_init__(self):
        if self.compatibility_matrix is None:
            object.__setattr__(self, 'compatibility_matrix', {
                'legacy': True,
                'stable': True,
                'modern': True,
                'experimental': False
            })

# Pattern validation constants
CANONICAL_PATTERN = "RSA-3072:[a-f0-9]{768}"
DIGEST_LENGTH = 768
SECURITY_POLICY = "FIPS_140_2_LEVEL_3"
```

Enforcement Logic Framework

python

```python
# Example: stable/rsa_3072/pattern.py
"""RSA-3072 Pattern Enforcement Implementation"""

from typing import Union, Optional
from enum import Enum
from ..common.automaton import ValidationResult
from .config import RSA3072Config

class PatternEnforcer:
    """Implements mandatory if/else control logic per OBINexus standard."""

    def __init__(self):
        self.config = RSA3072Config()

    def enforce_primitive_pattern(self,
                                  primitive_digest: str,
                                  context: str) -> ValidationResult:
        """
        Mandatory pattern enforcement for RSA-3072 primitives.
        Implements Section 2.1 of OBINexus Standard v1.0.
        """
        # Phase 1: Pattern Recognition
        if not self.config.pattern.match(primitive_digest):
            return ValidationResult.REJECT_UNKNOWN_PATTERN

        # Phase 2: Security Level Validation
        if context not in self._get_allowed_contexts():
            return ValidationResult.REJECT_CONTEXT_VIOLATION

        # Phase 3: Compatibility Matrix Check
        if not self._validate_compatibility(context):
            return ValidationResult.REJECT_INCOMPATIBLE_VERSION

        return ValidationResult.ACCEPT_VALIDATED

    def _get_allowed_contexts(self) -> set:
        """Define allowed operational contexts for RSA-3072."""
        return {
            "key_generation",
            "signature_verification",
            "legacy_migration",
            "audit_verification"
        }

    def _validate_compatibility(self, context: str) -> bool:
```

```
        """Validate against compatibility matrix."""
        return self.config.compatibility_matrix.get(context, False)
```

Deployment Automation

GitHub Actions CI/CD Pipeline

The repository implements comprehensive automation for testing, validation, and deployment across all cryptographic primitive versions.

Primary CI Workflow (`ci.yml`)

yaml

```yaml
name: PySchemia-Crypto CI

on:
  push:
    branches: [ main, legacy, experimental, 'release/*' ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    strategy:
      matrix:
        python-version: ["3.8", "3.9", "3.10", "3.11"]
        crypto-layer: ["legacy", "stable", "experimental"]

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v4
      with:
        python-version: ${{ matrix.python-version }}

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -e .[dev]
        pip install pytest pytest-cov cryptography

    - name: Validate pattern compliance
      run: |
        python -m pytest tests/test_patterns.py::test_${{ matrix.crypto-layer }}_compliance

    - name: Cross-language validation
      run: |
        python tests/test_cross_language.py --layer=${{ matrix.crypto-layer }}

    - name: Security compliance check
      run: |
        python -m pytest tests/test_audit.py --layer=${{ matrix.crypto-layer }}

    - name: Generate coverage report
      run: |
        pytest --cov=pyschemia_crypto --cov-report=xml
```

```yaml
      - name: Upload coverage to Codecov
        uses: codecov/codecov-action@v3
```

PyPI Deployment Workflow (`release.yml`)

yaml

```yaml
name: PyPI Release

on:
  release:
    types: [published]
  workflow_dispatch:
    inputs:
      version_type:
        description: 'Version type'
        required: true
        default: 'stable'
        type: choice
        options:
        - legacy
        - stable
        - experimental

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'

    - name: Install build dependencies
      run: |
        python -m pip install --upgrade pip
        pip install build twine

    - name: Configure version suffix
      run: |
        VERSION_SUFFIX=""
        if [ "${{ github.event.inputs.version_type }}" = "legacy" ]; then
          VERSION_SUFFIX="-legacy"
        elif [ "${{ github.event.inputs.version_type }}" = "experimental" ]; then
          VERSION_SUFFIX="-experimental"
        fi
        echo "VERSION_SUFFIX=$VERSION_SUFFIX" >> $GITHUB_ENV

    - name: Build package
      run: python -m build
```

```
    - name: Publish to PyPI
      env:
        TWINE_USERNAME: __token__
        TWINE_PASSWORD: ${{ secrets.PYPI_API_TOKEN }}
      run: |
        twine upload dist/*
```

Testing Framework

Pattern Compliance Testing

The testing framework validates all aspects of the OBINexus standard implementation:

python

```python
# tests/test_patterns.py
"""Pattern compliance testing for all cryptographic layers."""

import pytest
from pyschemia_crypto.legacy.rsa_2048 import PatternEnforcer as LegacyRSA
from pyschemia_crypto.stable.rsa_3072 import PatternEnforcer as StableRSA
from pyschemia_crypto.experimental.rsa_4096 import PatternEnforcer as ExperimentalRSA

class TestPatternCompliance:
    """Validate pattern enforcement across all cryptographic layers."""

    @pytest.mark.parametrize("layer,enforcer_class", [
        ("legacy", LegacyRSA),
        ("stable", StableRSA),
        ("experimental", ExperimentalRSA),
    ])
    def test_pattern_recognition(self, layer, enforcer_class):
        """Test mandatory pattern recognition per OBINexus Section 2.1."""
        enforcer = enforcer_class()

        # Valid pattern should be accepted
        valid_digest = self._generate_valid_digest(layer)
        result = enforcer.enforce_primitive_pattern(valid_digest, "key_generation")
        assert result == ValidationResult.ACCEPT_VALIDATED

        # Invalid pattern should be rejected
        invalid_digest = "INVALID-PATTERN:xyz123"
        result = enforcer.enforce_primitive_pattern(invalid_digest, "key_generation")
        assert result == ValidationResult.REJECT_UNKNOWN_PATTERN

    def test_canonical_normalization(self):
        """Test isomorphic reduction implementation per OBINexus Section 4."""
        from pyschemia_crypto.common.normalize import normalize_primitive_input

        test_cases = [
            ("rsa-3072:abc123", "RSA-3072:ABC123"),
            ("RSA_3072:abc123", "RSA-3072:ABC123"),
            ("rsa3072:abc123", "RSA-3072:ABC123"),
        ]

        for input_variant, expected_canonical in test_cases:
            result = normalize_primitive_input(input_variant)
            assert result == expected_canonical

    def _generate_valid_digest(self, layer: str) -> str:
        """Generate valid digest for testing purposes."""
```

```
    digest_lengths = {
        "legacy": 512,     # RSA-2048
        "stable": 768,     # RSA-3072
        "experimental": 1024  # RSA-4096
    }

    length = digest_lengths[layer]
    algorithm = f"RSA-{length * 4}"  # Convert to key size
    hex_digest = "a" * length

    return f"{algorithm}:{hex_digest}"
```

Security Compliance Framework

Audit Trail Implementation

The package implements secure audit logging per OBINexus Section 6:

python

```python
# pyschemia_crypto/common/audit.py
"""Secure audit trail implementation per OBINexus Standard v1.0."""

import hashlib
from datetime import datetime
from dataclasses import dataclass
from typing import Optional

@dataclass
class SecureAuditNode:
    """Secure audit trail node for primitive operations."""

    timestamp: datetime
    primitive_hash: str
    pattern_hash: str
    operation_context: str
    compliance_level: str = "OBINexus-v1.0"

    @classmethod
    def create_audit_entry(cls,
                           primitive_digest: str,
                           pattern: str,
                           context: str) -> 'SecureAuditNode':
        """Create audit entry with secure hash references."""
        return cls(
            timestamp=datetime.utcnow(),
            primitive_hash=f"PRIM_{hashlib.sha256(primitive_digest.encode()).hexdigest()[:16]}"
            pattern_hash=f"PAT_{hashlib.sha256(pattern.encode()).hexdigest()[:16]}",
            operation_context=context
        )

    def to_audit_record(self) -> dict:
        """Generate compliant audit record."""
        return {
            "timestamp": self.timestamp.isoformat(),
            "primitive_ref": self.primitive_hash,
            "pattern_ref": self.pattern_hash,
            "context": self.operation_context,
            "compliance_level": self.compliance_level
        }
```

PyPI Package Configuration

Setup Configuration (`setup.py`)

python

```python
"""PySchemia-Crypto package setup configuration."""

from setuptools import setup, find_packages
import os

# Read version from environment or default
version = os.getenv('PACKAGE_VERSION', '1.0.0')
version_suffix = os.getenv('VERSION_SUFFIX', '')

setup(
    name="pyschemia-crypto",
    version=f"{version}{version_suffix}",
    packages=find_packages(),
    include_package_data=True,

    # Package metadata
    description="OBINexus Cryptographic Interoperability Layer",
    long_description=open('README.md').read(),
    long_description_content_type="text/markdown",

    author="OBINexus Computing",
    author_email="nnamdi@obinexus.com",
    url="https://github.com/obinexus/pyschemia-crypto",

    # Dependencies
    install_requires=[
        "cryptography>=3.4.8",
        "regex>=2021.8.3",
        "typing-extensions>=4.0.0",
    ],

    extras_require={
        "dev": [
            "pytest>=6.0",
            "pytest-cov>=2.12",
            "black>=21.0",
            "mypy>=0.910",
            "flake8>=3.9",
        ],
        "docs": [
            "sphinx>=4.0",
            "sphinx-rtd-theme>=0.5",
        ],
    },

    # Classifiers
```

```
    classifiers=[
        "Development Status :: 4 - Beta",
        "Intended Audience :: Developers",
        "License :: OSI Approved :: MIT License",
        "Programming Language :: Python :: 3",
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Programming Language :: Python :: 3.10",
        "Programming Language :: Python :: 3.11",
        "Topic :: Security :: Cryptography",
        "Topic :: Software Development :: Libraries :: Python Modules",
    ],

    python_requires=">=3.8",

    # Entry points for CLI tools
    entry_points={
        "console_scripts": [
            "pyschemia-validate=pyschemia_crypto.cli:validate_command",
            "pyschemia-audit=pyschemia_crypto.cli:audit_command",
        ],
    },
)
```

Repository Initialization Script

The complete repository can be initialized using the provided automation script, ensuring consistent structure across deployments and maintaining compliance with the formal specification framework.