UAIP Protocol

Security & Compliance Guide

Production Deployment & Regulatory Framework

Version: 1.0.0
Last Updated: January 2025
Repository: https://github.com/jahanzaibahmad112-dotcom/UAIP-Protocol

---

Table of Contents

1. Security Best Practices
2. Production Deployment
3. Compliance Frameworks
4. Incident Response
5. Monitoring & Alerting
6. Disaster Recovery

Security Best Practices

Secret Management

Admin Keys

❌ NEVER:

```
# Hardcoded in code
ADMIN_KEY = "my-admin-password"

# In version control
git commit -m "Added admin key"

# In plain text files
echo "admin_key=secret123" > config.txt
```

✅ ALWAYS:

```
# Environment variables
export ADMIN_KEY=$(openssl rand -hex 32)

# Secrets manager (production)
aws secretsmanager get-secret-value --secret-id uaip/admin-key

# HashiCorp Vault
vault kv get secret/uaip/admin-key
```

Generation:

```
import secrets

# Generate secure admin key (64 characters)
admin_key = secrets.token_urlsafe(48)
print(f"ADMIN_KEY={admin_key}")
```

Agent Secret Keys

Storage:

```
# ❌ NEVER in code
agent = UAIP_Enterprise_SDK(secret_code=12345678)

# ✅ From environment
import os
secret = int(os.getenv("AGENT_SECRET_KEY"))
agent = UAIP_Enterprise_SDK(secret_code=secret)

# ✅ From secrets manager
from boto3 import client
sm = client('secretsmanager')
secret = int(sm.get_secret_value(SecretId='agent-secret')['SecretString'])
```

Rotation:

```
# Rotate secret every 90 days
def rotate_agent_secret():
    # Generate new secret
    new_secret = ZK_Privacy.generate_secret_key()
    new_commitment = ZK_Privacy.generate_commitment(new_secret)

    # Update in registry
    agent.update_commitment(new_commitment)

    # Store new secret securely
    store_secret(new_secret)
```

Network Security

TLS Configuration

nginx Configuration:

```
server {
    listen 443 ssl http2;
    server_name gateway.uaip.io;

    # TLS 1.3 only
    ssl_protocols TLSv1.3;

    # Strong ciphers
    ssl_ciphers 'TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256';

    # Certificate
    ssl_certificate /etc/letsencrypt/live/gateway.uaip.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/gateway.uaip.io/privkey.pem;

    # HSTS
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    # Proxy to gateway
    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
    }
}
```

Firewall Rules

```
# Allow HTTPS only
ufw allow 443/tcp

# Allow SSH (restrict to specific IPs)
ufw allow from 203.0.113.0/24 to any port 22

# Block all other incoming
ufw default deny incoming

# Enable
ufw enable
```

Database Security

File Permissions

```
# Database file: read/write owner only
chmod 600 uaip_vault.db

# Log directory: read/write/execute owner only
chmod 700 ./logs

# Backup: read owner only
chmod 400 backup.db.gz
```

Encryption at Rest

```
# Encrypt database file
openssl enc -aes-256-cbc -salt \
  -in uaip_vault.db \
  -out uaip_vault.db.enc \
  -pass file:./db_key

# Decrypt for use
openssl enc -d -aes-256-cbc \
  -in uaip_vault.db.enc \
  -out uaip_vault.db \
  -pass file:./db_key
```

Input Validation

Strict Validation

```
from pydantic import BaseModel, Field, validator

class TransactionRequest(BaseModel):
    amount: Decimal = Field(ge=0.01, le=1000000000)
    task: str = Field(min_length=3, max_length=5000)
    intent: str = Field(min_length=3, max_length=2000)

    @validator('task')
    def validate_task(cls, v):
        # No SQL injection characters
        if any(c in v for c in ["'", '"', ';', '--']):
            raise ValueError("Invalid characters in task")
        return v
```

Rate Limiting

Production Configuration

```
 # Rate limits per endpoint
RATE_LIMITS = {
    '/v1/register': (10, 60),      # 10 per minute
    '/v1/execute': (100, 60),      # 100 per minute
    '/v1/check': (200, 60),        # 200 per minute
    '/v1/decision': (50, 60)       # 50 per minute (admin)
}

# IP-based rate limiting
from slowapi import Limiter
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)

@app.post("/v1/execute")
@limiter.limit("100/minute")
async def execute(request: Request, req: UAIPPacket):
    ...
```

Production Deployment

Infrastructure Setup

Server Requirements

Minimum (Single Server):

- CPU: 4 cores
- RAM: 8 GB
- Disk: 100 GB SSD
- Network: 100 Mbps

Recommended (Production):

- CPU: 8 cores
- RAM: 16 GB
- Disk: 500 GB NVMe SSD
- Network: 1 Gbps

Docker Deployment

Dockerfile:

```
FROM python:3.10-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY . .

# Create non-root user
RUN useradd -m -u 1000 uaip && chown -R uaip:uaip /app
USER uaip

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s \
  CMD curl -f http://localhost:8000/health || exit 1

# Run
CMD ["uvicorn", "gateway:app", "--host", "0.0.0.0", "--port", "8000"]
```

docker-compose.yml:

```
version: '3.8'

services:
  gateway:
    build: .
    ports:
      - "8000:8000"
    environment:
      - ADMIN_KEY=${ADMIN_KEY}
      - DB_PATH=/data/uaip_vault.db
    volumes:
      - ./data:/data
      - ./logs:/app/logs
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./certs:/etc/nginx/certs
    depends_on:
      - gateway
    restart: unless-stopped
```

Kubernetes Deployment

deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: uaip-gateway
spec:
  replicas: 3
  selector:
    matchLabels:
      app: uaip-gateway
  template:
    metadata:
      labels:
        app: uaip-gateway
    spec:
      containers:
      - name: gateway
        image: uaip-protocol:1.0.0
        ports:
        - containerPort: 8000
        env:
        - name: ADMIN_KEY
          valueFrom:
            secretKeyRef:
              name: uaip-secrets
              key: admin-key
        - name: DATABASE_URL
          value: postgresql://uaip:pass@postgres:5432/uaip
        resources:
          requests:
            memory: "2Gi"
            cpu: "1000m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 10
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 10
```

Database Migration

SQLite to PostgreSQL

```
import sqlite3
import psycopg2

# Export from SQLite
sqlite_conn = sqlite3.connect('uaip_vault.db')
sqlite_conn.row_factory = sqlite3.Row

# Import to PostgreSQL
pg_conn = psycopg2.connect(
    "dbname=uaip user=uaip password=secure host=localhost"
)

# Migrate inventory
for row in sqlite_conn.execute('SELECT * FROM inventory'):
    pg_conn.execute(
        'INSERT INTO inventory (did, pk, zk_commitment, created_at, updated_at) VALUES (%s, %s, %s, %s, %s)',
        (row['did'], row['pk'], row['zk_commitment'], row['created_at'], row['updated_at'])
    )

pg_conn.commit()
```

Backup Strategy

Automated Backups

```
#!/bin/bash
# backup.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups"

# Stop writes (WAL checkpoint)
sqlite3 uaip_vault.db "PRAGMA wal_checkpoint(FULL);"

# Create backup
cp uaip_vault.db "${BACKUP_DIR}/uaip_vault_${DATE}.db"

# Compress
gzip "${BACKUP_DIR}/uaip_vault_${DATE}.db"

# Upload to S3
aws s3 cp "${BACKUP_DIR}/uaip_vault_${DATE}.db.gz" \
  s3://uaip-backups/

# Keep last 30 days locally
find ${BACKUP_DIR} -name "*.gz" -mtime +30 -delete
```

Cron Schedule:

```
# Backup every 6 hours
0 */6 * * * /opt/uaip/backup.sh
```

Compliance Frameworks

EU AI Act Compliance

Article 14: Human Oversight

Requirement: High-risk AI systems must have human oversight

Implementation:

```
# Transactions ≥$1,000 require human approval
if amount >= Decimal("1000"):
    return "PENDING_ENFORCED", {
        "reasoning": "High-value transaction requires human oversight",
        "grounded_law": "EU AI Act Article 14: Mandatory human oversight for high-risk autonomous spending"
    }
```

Dashboard: http://localhost:8000

- Admins review pending transactions
- Approve/deny with logged reasoning
- Complete audit trail maintained

Article 10: Data Quality

Requirement: Training data must be relevant, representative, and free of errors

Implementation:

- Compliance rules maintained in version control
- Regular review of prohibited keywords
- Test suite validates rule accuracy

Article 13: Transparency

Requirement: Users must be informed when interacting with AI

Implementation:

```
audit_report = {
    "disclaimer": "AI-generated audit. Always verify with human counsel.",
    "model_metadata": "Llama-3-Legal-14B-RAG",
    "verification_reasoning": "...",
    "grounded_law": "..."
}
```

SOC2 Compliance

CC7.2: Monitoring

Requirement: Continuous monitoring of system components

Implementation:

- Real-time dashboard monitoring
- Alert on failed transactions
- Statistics tracking
- Log aggregation

Metrics Tracked:

```
stats = {
    'total_transactions': 1000,
    'failed_transactions': 5,
    'compliance_blocks': 2,
    'average_processing_time_ms': 345
}
```

CC6.1: Logical Access

Requirement: Restrict logical access through use of access credentials

Implementation:

- Admin endpoints require ADMIN_KEY
- Agents require cryptographic proof
- No shared credentials
- Regular key rotation

CC7.3: Change Management

Requirement: Track changes to system components

Implementation:

- Git version control
- Semantic versioning (1.0.0)
- Change log in database
- Audit trail of code changes

GDPR Compliance

Article 5: Data Minimization

Requirement: Collect only necessary data

Implementation:

```
# Only store essential data
inventory_record = {
    "did": agent_id,              # Identifier (not PII)
    "pk": public_key,             # Public data
    "zk_commitment": commitment,  # Public data
    # NO email, name, or other PII
}
```

Article 15: Right of Access

Requirement: Users can request their data

Implementation:

```
@app.get("/v1/agent/{did}/data")
async def get_agent_data(did: str, admin_key: str = Header(None)):
    # Verify admin or agent owner
    verify_access(admin_key, did)

    # Return all data for this agent
    data = conn.execute('SELECT * FROM inventory WHERE did=?', (did,)).fetchone()
    logs = conn.execute('SELECT * FROM action_logs WHERE sender=?', (did,)).fetchall()

    return {"agent_data": data, "transaction_history": logs}
```

Article 17: Right to Erasure

Requirement: Users can request data deletion

Implementation:

```
@app.delete("/v1/agent/{did}")
async def delete_agent(did: str, admin_key: str = Header(None)):
    verify_access(admin_key, did)

    # Delete from inventory
    conn.execute('DELETE FROM inventory WHERE did=?', (did,))

    # Anonymize logs (keep for audit but remove PII)
    conn.execute(
        'UPDATE action_logs SET sender=? WHERE sender=?',
        (f"DELETED_{hash(did)}", did)
    )

    return {"status": "deleted"}
```

Article 32: Security Measures

Requirement: Implement appropriate technical and organizational measures

Implementation:

- Encryption in transit (TLS 1.3)
- Encryption at rest (database file encryption)
- Access controls (admin key, cryptographic auth)
- Regular security audits
- Incident response plan

Incident Response

Incident Types

1. Data Breach

Detection:

- Unauthorized access to database
- Leaked credentials
- Suspicious admin activity

Response:

```
 # 1. Immediately rotate all keys
export NEW_ADMIN_KEY=$(openssl rand -hex 32)

# 2. Lock down access
ufw deny all

# 3. Analyze logs
grep "401\|403" uaip_gateway.log > unauthorized_access.log

# 4. Notify affected parties
python notify_breach.py --affected-dids did:uaip:acme:123

# 5. Document incident
echo "Breach detected at $(date)" >> incidents.log
```

---

2. Service Outage

Detection:

- Health check failures
- Gateway not responding
- Database locked

Response:

```
# 1. Check service status
systemctl status uaip-gateway

# 2. Check logs
tail -n 100 uaip_gateway.log

# 3. Restart if needed
systemctl restart uaip-gateway

# 4. Verify recovery
curl http://localhost:8000/health

# 5. Post-mortem
python generate_incident_report.py --type=outage
```

---

3. Compliance Violation

Detection:

- Audit log shows blocked transaction should have passed
- Compliance rules incorrect
- False positive rate too high

Response:

```
# 1. Review compliance rules
auditor = ComplianceAuditor()
print(auditor.INSTANT_BLOCK_KEYWORDS)

# 2. Update rules if needed
# Add to compliance.py:
# INSTANT_BLOCK_KEYWORDS.append("new_keyword")

# 3. Reprocess affected transactions
for tx_id in affected_transactions:
    reprocess_transaction(tx_id)

# 4. Notify affected agents
notify_agents(affected_dids, "Compliance issue resolved")
```

Incident Response Team

Roles & Responsibilities

Incident Commander:

- Declares incident
- Coordinates response
- Makes final decisions

Technical Lead:

- Investigates root cause
- Implements fixes
- Validates recovery

Communications Lead:

- Notifies stakeholders
- Updates status page
- Handles external communications

Compliance Officer:

- Assesses regulatory impact
- Documents for auditors
- Files required reports

Incident Severity Levels

Level	Description	Response Time	Example
P0 - Critical	Complete service outage	< 15 minutes	Gateway down
P1 - High	Major feature broken	< 1 hour	Compliance auditor failing
P2 - Medium	Minor feature impacted	< 4 hours	Dashboard not loading
P3 - Low	Cosmetic issue	< 1 day	Typo in logs

Monitoring & Alerting

Metrics to Monitor

Application Metrics

```
from prometheus_client import Counter, Histogram, Gauge

# Transaction counters
tx_total = Counter('uaip_transactions_total', 'Total transactions')
tx_success = Counter('uaip_transactions_success', 'Successful transactions')
tx_failed = Counter('uaip_transactions_failed', 'Failed transactions')

# Processing time
tx_duration = Histogram('uaip_transaction_duration_seconds', 'Transaction duration')

# Current state
pending_approvals = Gauge('uaip_pending_approvals', 'Pending approvals')

# In code
tx_total.inc()
with tx_duration.time():
    result = process_transaction()
if result['status'] == 'SUCCESS':
    tx_success.inc()
```

System Metrics

```
 # CPU usage
top -bn1 | grep "Cpu(s)" | awk '{print $2}'

# Memory usage
free -m | awk 'NR==2{printf "%.2f%%\n", $3*100/$2 }'

# Disk usage
df -h | grep "/dev/sda1" | awk '{print $5}'

# Database size
du -h uaip_vault.db
```

Alert Configuration

PagerDuty Integration

```
import requests

def send_alert(severity, message):
    requests.post('https://events.pagerduty.com/v2/enqueue', json={
        'routing_key': PAGERDUTY_KEY,
        'event_action': 'trigger',
        'payload': {
            'summary': message,
            'severity': severity,
            'source': 'uaip-gateway',
            'timestamp': datetime.utcnow().isoformat()
        }
    })

# Trigger on errors
if failed_attempts >= MAX_FAILED_ATTEMPTS:
    send_alert('critical', f'IP {ip} exceeded failed attempts')
```

Alert Rules

```
# alerts.yml
groups:
  - name: uaip
    rules:
      - alert: HighErrorRate
        expr: rate(uaip_transactions_failed[5m]) > 0.1
        annotations:
          summary: "High error rate detected"

      - alert: DatabaseLocked
        expr: uaip_db_lock_errors > 0
        annotations:
          summary: "Database lock detected"

      - alert: LowDiskSpace
        expr: disk_usage_percent > 90
        annotations:
          summary: "Disk usage above 90%"
```

Disaster Recovery

Recovery Time Objective (RTO)

Target: 4 hours

Steps:

1. Provision new server (1 hour)
2. Restore from backup (1 hour)
3. Verify integrity (1 hour)
4. Resume operations (1 hour)

Recovery Point Objective (RPO)

Target: 6 hours (max data loss)

Implementation:

- Backups every 6 hours
- WAL replication for lower RPO

Disaster Scenarios

Complete Data Loss

```
# 1. Provision new server
terraform apply

# 2. Install UAIP
git clone https://github.com/jahanzaibahmad112-dotcom/UAIP-Protocol
cd UAIP-Protocol
pip install -r requirements.txt

# 3. Restore latest backup
aws s3 cp s3://uaip-backups/latest.db.gz .
gunzip latest.db.gz
mv latest.db uaip_vault.db

# 4. Verify
python -c "import sqlite3; conn = sqlite3.connect('uaip_vault.db'); print(conn.execute('SELECT COUNT(*) FROM inventory').fetchone())

# 5. Start service
python gateway.py
```

Compromised Keys

```
# 1. Generate new admin key
export NEW_ADMIN_KEY=$(openssl rand -hex 32)

# 2. Update all configs
sed -i "s/ADMIN_KEY=.*/ADMIN_KEY=${NEW_ADMIN_KEY}/" .env

# 3. Notify all agents
python scripts/notify_key_rotation.py

# 4. Document in incident log
echo "Admin key rotated due to compromise" >> incidents.log
```

Testing Disaster Recovery

Quarterly DR Test:

```
#!/bin/bash
# dr_test.sh

echo "Starting DR test..."

# 1. Create test backup
python backup.py --test

# 2. Provision test environment
terraform apply -var="environment=dr-test"

# 3. Restore backup
./restore.sh --backup=latest-test

# 4. Run test suite
pytest tests/ --dr-test

# 5. Verify metrics
python verify_recovery.py

# 6. Tear down test environment
terraform destroy -var="environment=dr-test"

echo "DR test complete. See report.html for details."
```

Conclusion

UAIP Protocol is designed with security and compliance as first-class concerns:

 Security by Design: Multi-layer defense, constant-time operations, input validation
 Regulatory Compliance: EU AI Act, SOC2, GDPR frameworks built-in
 Production Ready: Docker/Kubernetes deployment, monitoring, alerting
 Incident Response: Documented procedures, severity levels, escalation paths
 Disaster Recovery: 4-hour RTO, automated backups, tested recovery procedures

For Production Deployment:

1. Review all security best practices
2. Configure monitoring and alerting
3. Test disaster recovery procedures
4. Complete compliance documentation
5. Conduct security audit
6. Train incident response team

For Enterprise Support:

- Security audits and penetration testing
- Custom compliance frameworks
- 24/7 incident response
- SLA-backed uptime guarantees

Contact: [your-email] | [your-linkedin]

Document Version: 1.0.0
Last Updated: January 15, 2025
UAIP Protocol Project