UAIP Protocol

API Reference

Complete Endpoint & SDK Documentation

Version: 1.0.0
Last Updated: January 2025
Base URL: `http://localhost:8000` (development) | `https://gateway.uaip.io` (production)
Repository: https://github.com/jahanzaibahmad112-dotcom/UAIP-Protocol

---

Table of Contents

1. Gateway REST API
2. SDK Reference (Python)
3. Compliance Auditor API
4. Settlement Engine API
5. Privacy Module API
6. Error Codes
7. Rate Limits
8. Authentication

Gateway REST API

Base URL

Development: `http://localhost:8000`
Production: `https://gateway.uaip.io`

Authentication

All endpoints except `/health` require cryptographic authentication:

- **Ed25519 Signature** in request body
- **Public Key** in request headers
- **Nonce** for replay protection

Endpoints

1. Agent Registration

Endpoint: `POST /v1/register`

Description: Register a new agent in the UAIP network

Request Headers:

```
Content-Type: application/json
```

Request Body:

```
{
  "registration_data": {
    "agent_id": "did:uaip:acme-corp:abc123",
    "zk_commitment": 1234567890,
    "public_key": "a3f5b2c8d1e4f9a7b6c5d8e1f2a3b4c5",
    "timestamp": 1705334400.0
  },
  "signature": "ed25519_signature_hex",
  "public_key": "a3f5b2c8d1e4f9a7b6c5d8e1f2a3b4c5"
}
```

Request Fields:

Field	Type	Required	Description
`registration_data`	object	Yes	Agent registration information
`registration_data.agent_id`	string	Yes	DID (max 500 chars)
`registration_data.zk_commitment`	integer	Yes	Zero-Knowledge commitment
`registration_data.public_key`	string	Yes	Ed25519 public key (hex)
`registration_data.timestamp`	float	Yes	Unix timestamp
`signature`	string	Yes	Ed25519 signature (hex)
`public_key`	string	Yes	Ed25519 public key (hex)

Success Response (200):

```
{
  "status": "REGISTERED",
  "agent_id": "did:uaip:acme-corp:abc123"
}
```

Error Responses:

400 - Missing Fields:

```
{
  "detail": "Missing required fields: agent_id, zk_commitment"
}
```

401 - Invalid Signature:

```
{
  "detail": "Invalid signature"
}
```

429 - Rate Limited:

```
{
  "detail": "Rate limit exceeded"
}
```

Example (cURL):

```
curl -X POST http://localhost:8000/v1/register \
  -H "Content-Type: application/json" \
  -d '{
    "registration_data": {
      "agent_id": "did:uaip:acme:abc123",
      "zk_commitment": 123456,
      "public_key": "a3f5b2c8...",
      "timestamp": 1705334400.0
    },
    "signature": "ed25519_sig...",
    "public_key": "a3f5b2c8..."
  }'
```

---

2. Execute Transaction

Endpoint: `POST /v1/execute`

Description: Execute a governed agent transaction

Request Headers:

```
Content-Type: application/json
```

Request Body:

```json
{
  "sender_id": "did:uaip:acme-corp:abc123",
  "task": "process_invoice",
  "amount": "50.00",
  "chain": "BASE",
  "intent": "Q1 vendor payment processing",
  "data": {
    "task": "process_invoice",
    "amount": "50.00",
    "intent": "Q1 vendor payment",
    "nonce": "a3f5b2c8-d1e4-f9a7-b6c5-d8e1f2a3b4c5",
    "timestamp": 1705334400.0
  },
  "signature": "ed25519_signature_hex",
  "public_key": "a3f5b2c8d1e4f9a7b6c5d8e1f2a3b4c5",
  "nonce": "a3f5b2c8-d1e4-f9a7-b6c5-d8e1f2a3b4c5",
  "timestamp": 1705334400.0,
  "zk_proof": {
    "r": 98765432109876543210,
    "s": 12345678901234567890,
    "timestamp": 1705334400
  }
}
```

Request Fields:

Field	Type	Required	Description
`sender_id`	string	Yes	Agent DID (max 500 chars)
`task`	string	Yes	Task description (max 5000 chars)
`amount`	string	Yes	Amount in USD (decimal string)
`chain`	string	Yes	Blockchain: BASE, SOLANA, ETHEREUM, POLYGON
`intent`	string	Yes	Human-readable intent (max 2000 chars)

Field	Type	Required	Description
`data`	object	Yes	Transaction data to sign
`signature`	string	Yes	Ed25519 signature of data (hex)
`public_key`	string	Yes	Ed25519 public key (hex)
`nonce`	string	Yes	Unique UUID (replay protection)
`timestamp`	float	Yes	Unix timestamp (±30s tolerance)
`zk_proof`	object	Yes	Zero-Knowledge proof
`zk_proof.r`	integer	Yes	Proof r value
`zk_proof.s`	integer	Yes	Proof s value
`zk_proof.timestamp`	integer	Yes	Proof timestamp

Success Response (200) - Immediate:

```
{
  "status": "SUCCESS",
  "request_id": "uaip_tx_a3f5b2c8",
  "settlement": {
    "status": "SUCCESS",
    "tx_id": "uaip_tx_a3f5b2c8",
    "amount": 50.0,
    "fee": 0.50,
    "payout": 49.50,
    "chain": "BASE",
    "currency": "USDC",
    "timestamp": 1705334400.0,
    "processing_time_ms": 145.23
  }
}
```

Success Response (200) - Pending Approval:

```
{
  "status": "PENDING_APPROVAL",
  "request_id": "uaip_tx_a3f5b2c8",
  "message": "High-value transaction requires human approval"
}
```

Error Responses:

400 - Invalid Amount:

```
{
  "detail": "Amount must be at least 0.01"
}
```

401 - Identity Verification Failed:

```
{
  "detail": "IDENTITY_VERIFICATION_FAILED"
}
```

403 - Replay Attack:

```
{
  "detail": "REPLAY_ATTACK_DETECTED"
}
```

403 - Blacklisted:

```
{
  "detail": "BLACKLISTED: Policy violation"
}
```

451 - Compliance Violation:

```
{
  "detail": {
    "error": "COMPLIANCE_VIOLATION",
    "audit": {
      "audit_id": "AUDIT-ABC123",
      "status": "TERMINATE",
      "verification_reasoning": "Prohibited keyword detected",
      "grounded_law": "AML/KYC Regulations"
    }
  }
}
```

500 - Settlement Failed:

```
{
  "detail": "Settlement processing failed"
}
```

Example (cURL):

```
curl -X POST http://localhost:8000/v1/execute \
  -H "Content-Type: application/json" \
  -d '{
    "sender_id": "did:uaip:acme:abc123",
    "task": "process_invoice",
    "amount": "50.00",
    "chain": "BASE",
    "intent": "Q1 vendor payment",
    "data": {...},
    "signature": "...",
    "public_key": "...",
    "nonce": "...",
    "timestamp": 1705334400.0,
    "zk_proof": {...}
  }'
```

---

3. Check Transaction Status

Endpoint: `GET /v1/check/{request_id}`

Description: Check status of pending transaction

Path Parameters:

Parameter	Type	Required	Description
`request_id`	string	Yes	Transaction request ID (UUID format)

Success Response (200):

```
{
  "status": "WAITING",
  "request_id": "uaip_tx_a3f5b2c8"
}
```

Possible Status Values:

- `WAITING` - Pending admin approval

- `APPROVED` - Approved, settlement processing
- `REJECTED` - Rejected by admin
- `NOT_FOUND` - Request ID not found

Example (cURL):

```
curl http://localhost:8000/v1/check/uaip_tx_a3f5b2c8
```

---

4. Manual Decision (Admin Only)

Endpoint: `POST /v1/decision/{request_id}/{choice}`

Description: Approve or deny pending transaction (admin only)

Path Parameters:

Parameter	Type	Required	Description
`request_id`	string	Yes	Transaction request ID (UUID)
`choice`	string	Yes	Decision: `allow` or `deny`

Request Headers:

```
X-Admin-Key: your-admin-key
```

Success Response (200) - Approved:

```
{
  "status": "SETTLED",
  "settlement": {
    "tx_id": "uaip_tx_a3f5b2c8",
    "amount": 50000.0,
    "fee": 260.0,
    "payout": 49740.0
  }
}
```

Error Responses:

401 - Unauthorized:

```
{
  "detail": "Unauthorized"
}
```

404 - Not Found:

```
{
  "detail": "Transaction not found or already processed"
}
```

Example (cURL):

```
curl -X POST http://localhost:8000/v1/decision/uaip_tx_abc123/allow \
  -H "X-Admin-Key: your-secure-admin-key"
```

---

5. Dashboard (Web UI)

Endpoint: `GET /`

Description: Real-time monitoring dashboard (HTML)

Response: HTML dashboard with:

- Transaction statistics

- Recent transaction list
- Approval buttons for pending transactions
- Auto-refresh every 10 seconds

Access: Open in browser at http://localhost:8000

---

6. Health Check

Endpoint: `GET /health`

Description: System health status

Success Response (200):

```
{
  "status": "healthy",
  "version": "1.0.0"
}
```

Error Response (503):

```
{
  "detail": "Service unhealthy"
}
```

Example (cURL):

```
curl http://localhost:8000/health
```

SDK Reference (Python)

Installation

```
pip install -e .  # Install from local repository
```

Or copy `sdk.py` to your project.

Class: UAIP_Enterprise_SDK

Constructor

```
UAIP_Enterprise_SDK(
    agent_name: str,
    company_name: str,
    secret_code: int,
    gateway_url: str = "http://localhost:8000",
    auto_register: bool = True,
    verify_ssl: bool = True
)
```

Parameters:

Parameter	Type	Required	Description
`agent_name`	str	Yes	Agent name (alphanumeric, max 100 chars)
`company_name`	str	Yes	Company name (alphanumeric, max 100 chars)

Parameter	Type	Required	Description
`secret_code`	int	Yes	Secret for ZK-proofs (keep secure!)
`gateway_url`	str	No	Gateway URL (default: localhost:8000)
`auto_register`	bool	No	Auto-register on init (default: True)
`verify_ssl`	bool	No	Verify SSL certs (default: True)

Returns: SDK instance

Raises:

- `ValueError` - Invalid inputs
- `RuntimeError` - Registration failed

Example:

```
from sdk import UAIP_Enterprise_SDK

agent = UAIP_Enterprise_SDK(
    agent_name="FinanceBot",
    company_name="Acme Corp",
    secret_code=12345678,
    gateway_url="https://gateway.uaip.io"
)

print(agent.did)  # did:uaip:acme-corp:a3f5b2c8
```

Method: register()

```
agent.register() -> Dict[str, Any]
```

Description: Register agent with gateway (called automatically if `auto_register=True`)

Returns:

```
{
    "status": "REGISTERED",
    "agent_id": "did:uaip:acme-corp:abc123"
}
```

Raises:

- `RuntimeError` - Registration failed

Example:

```
# Manual registration (if auto_register=False)
response = agent.register()
print(response['status'])  # "REGISTERED"
```

Method: call_agent()

```
agent.call_agent(
    task: str,
    amount: Union[int, float, Decimal, str],
    intent: str,
    chain: str = "BASE",
    metadata: Optional[Dict[str, Any]] = None,
    wait_for_approval: bool = True
) -> Dict[str, Any]
```

Description: Execute governed transaction

Parameters:

Parameter	Type	Required	Description
`task`	str	Yes	Task description (3-5000 chars)
`amount`	number/str	Yes	Amount in USD (0.01 - 1B)
`intent`	str	Yes	Human-readable intent (3-2000 chars)
`chain`	str	No	BASE, SOLANA, ETHEREUM, POLYGON
`metadata`	dict	No	Additional metadata (max 10KB)
`wait_for_approval`	bool	No	Wait for manual approval if needed

Returns (Success):

```
{
    "status": "SUCCESS",
    "request_id": "uaip_tx_abc123",
    "settlement": {
        "tx_id": "uaip_tx_abc123",
        "amount": 50.0,
        "fee": 0.50,
        "payout": 49.50,
        "chain": "BASE",
        "currency": "USDC",
        "timestamp": 1705334400.0,
        "processing_time_ms": 145.23
    }
}
```

Returns (Pending):

```
{
    "status": "PENDING_APPROVAL",
    "request_id": "uaip_tx_abc123",
    "message": "High-value transaction requires human approval"
}
```

Raises:

- `ValueError` - Invalid input
- `RuntimeError` - Transaction failed

Example:

```
# Simple transaction
result = agent.call_agent(
    task="process_invoice",
    amount=50.00,
    intent="Q1 vendor payment",
    chain="BASE"
)

if result['status'] == 'SUCCESS':
    print(f"□ Success! Fee: ${result['settlement']['fee']}")
elif result['status'] == 'PENDING_APPROVAL':
    print(f"□ Waiting for approval: {result['request_id']}")

# With metadata
result = agent.call_agent(
    task="data_analysis",
    amount=100.00,
    intent="Customer sentiment analysis",
    chain="SOLANA",
    metadata={"customer_id": "CUST-123", "dataset": "Q4-2024"}
)

# High-value transaction (will wait for approval)
result = agent.call_agent(
    task="enterprise_license",
    amount=50000.00,
    intent="Annual software license renewal",
    chain="ETHEREUM",
    wait_for_approval=True  # Blocks until approved
)
```

Method: get_statistics()

```
agent.get_statistics() -> Dict[str, Any]
```

Description: Get SDK usage statistics

Returns:

```
{
    "agent_did": "did:uaip:acme-corp:abc123",
    "agent_name": "FinanceBot",
    "company_name": "Acme Corp",
    "gateway_url": "http://localhost:8000",
    "total_requests": 42,
    "successful_requests": 38,
    "failed_requests": 4,
    "success_rate": 90.48,
    "total_amount_processed": 12500.00,
    "registrations": 1
}
```

Example:

```
stats = agent.get_statistics()
print(f"Success rate: {stats['success_rate']}%")
print(f"Total processed: ${stats['total_amount_processed']:,.2f}")
```

Method: health_check()

```
agent.health_check() -> Dict[str, Any]
```

Description: Check gateway connectivity

Returns (Healthy):

```
{
    "gateway_status": "healthy",
    "gateway_url": "http://localhost:8000",
    "response": {
        "status": "healthy",
        "version": "1.0.0"
    }
}
```

Returns (Unhealthy):

```
{
    "gateway_status": "unhealthy",
    "gateway_url": "http://localhost:8000",
    "error": "Connection refused"
}
```

Example:

```
health = agent.health_check()
if health['gateway_status'] == 'healthy':
    print("□ Gateway operational")
else:
    print(f"□ Gateway down: {health['error']}")
```

Method: close()

```
agent.close()
```

Description: Cleanup resources (HTTP connections)

Example:

```
# Context manager (automatic cleanup)
with UAIP_Enterprise_SDK("Bot", "Corp", 123) as agent:
    result = agent.call_agent(task="...", amount=10, intent="...")
# agent.close() called automatically

# Manual cleanup
agent = UAIP_Enterprise_SDK("Bot", "Corp", 123)
try:
    result = agent.call_agent(...)
finally:
    agent.close()
```

Properties

```
agent.did                   # str: Agent's Decentralized Identifier
agent.agent_name            # str: Agent name
agent.company_name          # str: Company name
agent.gateway               # str: Gateway URL
agent.pk                    # str: Ed25519 public key (hex)
agent.zk_commitment         # int: Zero-Knowledge commitment
```

Example:

```
print(f"Agent DID: {agent.did}")
print(f"Public Key: {agent.pk[:16]}...")
print(f"ZK Commitment: {agent.zk_commitment}")
```

Compliance Auditor API

Class: ComplianceAuditor

Constructor

```
ComplianceAuditor(
    log_dir: str = ".",
    log_filename: str = "uaip_forensic_records.json"
)
```

Parameters:

Parameter	Type	Required	Description
`log_dir`	str	No	Directory for audit logs (default: current)
`log_filename`	str	No	Audit log filename (default: uaip_forensic_records.json)

Example:

```
from compliance import ComplianceAuditor

auditor = ComplianceAuditor(
    log_dir="./audit_logs",
    log_filename="production_audits.json"
)
```

Method: run_active_audit()

```
auditor.run_active_audit(action_log: Dict[str, Any]) -> Tuple[str, Dict[str, Any]]
```

Description: Run compliance audit on transaction

Parameters:

Field	Type	Required	Description
`action_log`	dict	Yes	Transaction details
`action_log.sender`	str	Yes	Agent DID
`action_log.task`	str	Yes	Task description
`action_log.amount`	number	Yes	Amount in USD
`action_log.intent`	str	No	Intent description
`action_log.chain`	str	No	Blockchain name
`action_log.timestamp`	float	No	Unix timestamp

Returns: Tuple of `(status, audit_report)`

Status Values:

- `"TERMINATE"` - Transaction blocked
- `"PENDING_ENFORCED"` - Requires human approval
- `"PASSED"` - Transaction approved

Audit Report:

```
{
    "audit_id": "AUDIT-ABC123",
    "timestamp": "2025-01-15T10:30:00Z",
    "agent": "did:uaip:acme:abc123",
    "task": "process_invoice",
    "amount": "50.00",
    "chain": "BASE",
    "status": "PASSED",
    "verification_reasoning": "Standard nano-transaction verified.",
    "grounded_law": "UAIP Policy: Routine transaction logging.",
    "model_metadata": "Llama-3-Legal-14B-RAG (Simulated)",
    "audit_duration_ms": 12,
    "disclaimer": "AI-generated audit. Always verify with human counsel."
}
```

Example:

```
from compliance import ComplianceAuditor

auditor = ComplianceAuditor()

# Audit transaction
status, report = auditor.run_active_audit({
    "sender": "did:uaip:acme:abc123",
    "task": "process_invoice",
    "amount": 50.00,
    "intent": "Q1 vendor payment",
    "chain": "BASE",
    "timestamp": 1705334400.0
})

if status == "PASSED":
    print(f"□ Approved: {report['verification_reasoning']}")
elif status == "PENDING_ENFORCED":
    print(f"□ Needs approval: {report['verification_reasoning']}")
elif status == "TERMINATE":
    print(f"□ Blocked: {report['verification_reasoning']}")
```

Method: get_statistics()

```
auditor.get_statistics() -> Dict[str, int]
```

Description: Get audit statistics

Returns:

```
{
    "total_audits": 1000,
    "blocked": 5,
    "pending": 12,
    "passed": 983,
    "validation_errors": 0
}
```

Method: reset_statistics()

```
auditor.reset_statistics()
```

Description: Reset statistics (for testing)

Method: health_check()

```
auditor.health_check() -> Dict[str, Any]
```

Description: Check auditor health

Returns:

```
{
    "status": "healthy",
    "log_directory": "/path/to/logs",
    "log_file": "uaip_forensic_records.json",
    "log_writable": True,
    "statistics": {...},
    "total_keywords": 25
}
```

Settlement Engine API

Class: UAIPFinancialEngine

Constructor

```
UAIPFinancialEngine(
    log_dir: str = ".",
    treasury_did: Optional[str] = None,
    enable_self_payment: bool = False
)
```

Parameters:

Parameter	Type	Required	Description
`log_dir`	str	No	Directory for settlement logs
`treasury_did`	str	No	Override treasury DID (testing)
`enable_self_payment`	bool	No	Allow self-payments (default: False)

Example:

```
from settlement import UAIPFinancialEngine

engine = UAIPFinancialEngine(
    log_dir="./settlements",
    treasury_did="did:uaip:treasury:production"
)
```

Method: process_settlement()

```
engine.process_settlement(
    payer_did: str,
    amount_usd: Union[int, float, Decimal, str],
    payee_did: str,
    chain: str,
    idempotency_key: Optional[str] = None,
    metadata: Optional[Dict[str, Any]] = None
) -> Dict[str, Any]
```

Description: Process financial settlement

Parameters:

Parameter	Type	Required	Description
`payer_did`	str	Yes	Paying agent DID
`amount_usd`	number	Yes	Amount in USD
`payee_did`	str	Yes	Receiving agent DID
`chain`	str	Yes	BASE, SOLANA, ETHEREUM, POLYGON
`idempotency_key`	str	No	Prevent duplicate processing
`metadata`	dict	No	Additional metadata

Returns:

```
{
    "status": "SUCCESS",
    "tx_id": "uaip_tx_abc123",
    "amount": 50.0,
    "fee": 0.50,
    "payout": 49.50,
    "fee_percentage": 1.0,
    "tier": "B",
    "chain": "BASE",
    "currency": "USDC",
    "timestamp": 1705334400.0,
    "processing_time_ms": 25.5
}
```

Example:

```
result = engine.process_settlement(
    payer_did="did:uaip:acme:abc123",
    amount_usd=100.00,
    payee_did="did:uaip:provider:def456",
    chain="BASE",
    idempotency_key="invoice_2025_001"
)

print(f"Settlement: ${result['amount']}")
print(f"Fee: ${result['fee']} ({result['fee_percentage']:.2f}%)")
print(f"Payout: ${result['payout']}")
```

Method: calculate_projected_fee()

```
engine.calculate_projected_fee(amount: Union[int, float, Decimal, str]) -> Dict[str, Any]
```

Description: Calculate fee without processing

Returns:

```
{
    "amount": 1500.0,
    "fee": 15.0,
    "payout": 1485.0,
    "fee_percentage": 1.0,
    "tier": "B",
    "tier_name": "Mid-Range",
    "tier_description": "1.0% fee for transactions $10 - $10,000"
}
```

Example:

```
 projection = engine.calculate_projected_fee(1500.00)
print(f"Tier: {projection['tier_name']}")
print(f"Fee: ${projection['fee']} ({projection['fee_percentage']}%)")
```

Method: get_statistics()

```
engine.get_statistics() -> Dict[str, Any]
```

Description: Get settlement statistics

Returns:

```
{
    "total_transactions": 1000,
    "successful_transactions": 995,
    "failed_transactions": 5,
    "total_volume_usd": 125000.0,
    "total_fees_collected_usd": 1250.0,
    "average_transaction_usd": 125.0,
    "average_fee_usd": 1.25,
    "average_fee_percentage": 1.0,
    "tier_breakdown": {
        "tier_a": 50,
        "tier_b": 900,
        "tier_c": 50
    },
    "idempotency_cache_size": 342
}
```

Privacy Module API

Class: ZK_Privacy

Method: generate_secret_key()

```
ZK_Privacy.generate_secret_key() -> int
```

Description: Generate cryptographically secure secret key

Returns: Random integer secret key

Example:

```
from privacy import ZK_Privacy

secret = ZK_Privacy.generate_secret_key()
print(f"Secret: {secret}")  # Keep this secure!
```

Method: generate_commitment()

```
ZK_Privacy.generate_commitment(secret_key: int) -> int
```

Description: Create public commitment from secret

Parameters:

Parameter	Type	Required	Description
`secret_key`	int	Yes	Secret key (1 to 2^255-20)

Returns: Public commitment integer

Example:

```
secret = ZK_Privacy.generate_secret_key()
commitment = ZK_Privacy.generate_commitment(secret)
print(f"Commitment: {commitment}")  # Safe to share publicly
```

Method: create_proof()

```
ZK_Privacy.create_proof(
    secret_key: int,
    public_commitment: int,
    include_timestamp: bool = True,
    check_rate_limit: bool = True
) -> Dict[str, Any]
```

Description: Generate Zero-Knowledge proof

Parameters:

Parameter	Type	Required	Description
`secret_key`	int	Yes	Your secret key
`public_commitment`	int	Yes	Your public commitment
`include_timestamp`	bool	No	Add timestamp (default: True)
`check_rate_limit`	bool	No	Enforce rate limit (default: True)

Returns:

```
{
    "r": 98765432109876543210,
    "s": 12345678901234567890,
    "timestamp": 1705334400
}
```

Example:

```
secret = ZK_Privacy.generate_secret_key()
commitment = ZK_Privacy.generate_commitment(secret)
proof = ZK_Privacy.create_proof(secret, commitment)

# Proof can be shared - doesn't reveal secret!
```

Method: verify_proof()

```
ZK_Privacy.verify_proof(
    proof: Dict[str, Any],
    public_commitment: int,
    check_freshness: bool = True
) -> bool
```

Description: Verify Zero-Knowledge proof

Parameters:

Parameter	Type	Required	Description
`proof`	dict	Yes	Proof from create_proof()
`public_commitment`	int	Yes	Public commitment to verify
`check_freshness`	bool	No	Check timestamp (default: True)

Returns: `True` if valid, `False` otherwise

Example:

```
# Create and verify
secret = ZK_Privacy.generate_secret_key()
commitment = ZK_Privacy.generate_commitment(secret)
proof = ZK_Privacy.create_proof(secret, commitment)

# Verify (anyone can do this without knowing secret)
is_valid = ZK_Privacy.verify_proof(proof, commitment)
print(f"Valid: {is_valid}")  # True
```

Method: get_security_parameters()

```
ZK_Privacy.get_security_parameters() -> Dict[str, Any]
```

Description: Get cryptographic parameters

Returns:

```
{
    "protocol": "Schnorr NIZK (Non-Interactive Zero-Knowledge)",
    "generator": 2,
    "prime_modulus": 57896044618658097711785492504343953926634992332820282019728792003956564819949,
    "prime_modulus_hex": "0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffed",
    "group_order": 57896044618658097711785492504343953926634992332820282019728792003956564819948,
    "prime_bits": 255,
    "security_level_bits": 128,
    "security_level_description": "128-bit (Curve25519 equivalent)",
    "hash_function": "SHA-256",
    "domain_separator": "UAIP-ZK-SCHNORR-v1.0",
    "proof_validity_seconds": 300,
    "max_clock_skew_seconds": 60,
    "rate_limit_proofs_per_minute": 100,
    "standards": [
        "NIST SP 800-186: DL-Based Cryptography",
        "Fiat-Shamir Heuristic",
        "RFC 8032 (Ed25519) parameter compatibility"
    ]
}
```

Convenience Functions

```python
from privacy import generate_identity, create_and_verify_proof, batch_verify_proofs

# Generate new identity
secret, commitment = generate_identity()

# Quick test
is_valid = create_and_verify_proof(secret, commitment)

# Batch verification
identities = [generate_identity() for _ in range(10)]
proofs = [ZK_Privacy.create_proof(s, c) for s, c in identities]
commitments = [c for s, c in identities]
results = batch_verify_proofs(proofs, commitments)
print(f"Success rate: {results['success_rate'] * 100}%")
```

Error Codes

HTTP Status Codes

Code	Meaning	Description
200	Success	Request completed successfully
400	Bad Request	Invalid input parameters
401	Unauthorized	Authentication failed
403	Forbidden	Access denied (blacklist, replay attack)
404	Not Found	Resource doesn't exist
429	Too Many Requests	Rate limit exceeded
451	Unavailable For Legal Reasons	Compliance violation

Code	Meaning	Description
500	Internal Server Error	Server error
503	Service Unavailable	System unhealthy

Application Error Codes

Gateway Errors

Error	Code	Description	Solution
Missing fields	`400`	Required field missing	Check request body
Invalid signature	`401`	Ed25519 verification failed	Verify signing process
Replay attack	`403`	Nonce already used	Generate new nonce
Blacklisted	`403`	Agent blacklisted	Contact support
Compliance violation	`451`	Policy violation	Review task/intent
Settlement failed	`500`	Blockchain error	Retry or contact support

SDK Errors

Exception	Description	Solution
`ValueError`	Invalid input	Check parameter values
`RuntimeError`	Operation failed	Check gateway health
`ConnectionError`	Cannot reach gateway	Verify gateway URL
`TimeoutError`	Request timeout	Check network, retry

Rate Limits

Default Limits

Endpoint	Limit	Window	Notes
`/v1/register`	10 requests	60 seconds	Per IP address
`/v1/execute`	100 requests	60 seconds	Per agent DID
`/v1/check`	200 requests	60 seconds	Per IP address
`/v1/decision`	50 requests	60 seconds	Per admin key

Rate Limit Headers

Response Headers:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 87
X-RateLimit-Reset: 1705334460
```

Rate Limit Response

429 Too Many Requests:

```
{
  "detail": "Rate limit exceeded",
  "retry_after": 42
}
```

Handling Rate Limits:

```
import time

try:
    result = agent.call_agent(task="...", amount=10, intent="...")
except RuntimeError as e:
    if "Rate limit" in str(e):
        print("Rate limited, waiting 60 seconds...")
        time.sleep(60)
        result = agent.call_agent(task="...", amount=10, intent="...")
```

Authentication

Overview

UAIP uses **multi-layer cryptographic authentication**:

1. **Ed25519 Digital Signatures** - Non-repudiation
2. **Zero-Knowledge Proofs** - Privacy-preserving identity
3. **Nonce Tracking** - Replay attack prevention

Authentication Flow

```
1. Agent generates request data
       │
       ▼
2. Agent signs data with Ed25519 private key
       │
       ▼
3. Agent creates ZK-proof of identity
       │
       ▼
4. Agent sends: data + signature + ZK-proof + nonce
       │
       ▼
5. Gateway verifies Ed25519 signature
       │
       ▼
6. Gateway checks nonce (not already used)
       │
       ▼
7. Gateway verifies ZK-proof
       │
       ▼
8. Gateway processes request
```

Ed25519 Signature

Signing:

```
import nacl.signing
import json

# Create signing key
signing_key = nacl.signing.SigningKey.generate()

# Data to sign (canonical JSON)
data = {"task": "process", "amount": "50.00"}
message = json.dumps(data, sort_keys=True, separators=(',', ':')).encode()

# Sign
signed = signing_key.sign(message)
signature = signed.signature.hex()
```

Verification:

```
# Get verify key from public key
verify_key = nacl.signing.VerifyKey(public_key_hex, encoder=nacl.encoding.HexEncoder)

# Verify signature
verify_key.verify(message, bytes.fromhex(signature))
```

Zero-Knowledge Proof

Creation:

```
from privacy import ZK_Privacy

secret = 12345678
commitment = ZK_Privacy.generate_commitment(secret)
proof = ZK_Privacy.create_proof(secret, commitment)
```

Verification:

```
is_valid = ZK_Privacy.verify_proof(proof, commitment)
```

Nonce Generation

Requirements:

- Cryptographically random (use `uuid.uuid4()`)
- Globally unique
- Used only once
- Tracked by gateway

Example:

```
import uuid

nonce = uuid.uuid4().hex  # e.g., "a3f5b2c8d1e4f9a7b6c5d8e1f2a3b4c5"
```

Request/Response Examples

Complete Transaction Example

Request

```
curl -X POST http://localhost:8000/v1/execute \
  -H "Content-Type: application/json" \
  -d '{
    "sender_id": "did:uaip:acme-corp:abc123",
    "task": "process_invoice",
    "amount": "150.00",
    "chain": "BASE",
    "intent": "Q1 vendor payment for invoice #12345",
    "data": {
      "task": "process_invoice",
      "amount": "150.00",
      "intent": "Q1 vendor payment",
      "nonce": "a3f5b2c8-d1e4-f9a7-b6c5-d8e1f2a3b4c5",
      "timestamp": 1705334400.0
    },
    "signature": "ed25519_signature_in_hex_format",
    "public_key": "a3f5b2c8d1e4f9a7b6c5d8e1f2a3b4c5",
    "nonce": "a3f5b2c8-d1e4-f9a7-b6c5-d8e1f2a3b4c5",
    "timestamp": 1705334400.0,
    "zk_proof": {
      "r": 98765432109876543210,
      "s": 12345678901234567890,
      "timestamp": 1705334400
    }
  }'
```

Response (Success)

```
{
  "status": "SUCCESS",
  "request_id": "uaip_tx_a3f5b2c8",
  "settlement": {
    "status": "SUCCESS",
    "tx_id": "uaip_tx_a3f5b2c8",
    "amount": 150.0,
    "fee": 1.5,
    "payout": 148.5,
    "fee_percentage": 1.0,
    "tier": "B",
    "chain": "BASE",
    "currency": "USDC",
    "timestamp": 1705334400.5,
    "processing_time_ms": 234.56
  }
}
```

Response (Pending)

```
{
  "status": "PENDING_APPROVAL",
  "request_id": "uaip_tx_a3f5b2c8",
  "message": "High-value transaction requires human approval"
}
```

Response (Blocked)

```
{
  "detail": {
    "error": "COMPLIANCE_VIOLATION",
    "audit": {
      "audit_id": "AUDIT-ABC123",
      "timestamp": "2025-01-15T10:30:00Z",
      "status": "TERMINATE",
      "verification_reasoning": "HARD_RULE_OVERRIDE: Prohibited keyword detected: 'laundering'",
      "grounded_law": "AML/KYC Regulations (FATF Recommendations 10-16): Transaction contains prohibited activities or keywords."
    }
  }
}
```

SDK Usage Examples

Basic Usage

```python
from sdk import UAIP_Enterprise_SDK

# Initialize agent
agent = UAIP_Enterprise_SDK(
    agent_name="FinanceBot",
    company_name="Acme Corp",
    secret_code=12345678
)

# Execute transaction
result = agent.call_agent(
    task="process_invoice",
    amount=50.00,
    intent="Q1 vendor payment"
)

print(f"Status: {result['status']}")
```

With Error Handling

```python
from sdk import UAIP_Enterprise_SDK
import time

agent = UAIP_Enterprise_SDK("Bot", "Corp", 123)

try:
    result = agent.call_agent(
        task="process_payment",
        amount=75.00,
        intent="Service payment",
        chain="BASE"
    )

    if result['status'] == 'SUCCESS':
        print(f"□ Paid ${result['settlement']['payout']}")
        print(f"Fee: ${result['settlement']['fee']}")

    elif result['status'] == 'PENDING_APPROVAL':
        print(f"□ Awaiting approval: {result['request_id']}")

except ValueError as e:
    print(f"□ Invalid input: {e}")

except RuntimeError as e:
    if "Rate limit" in str(e):
        print("Rate limited, waiting...")
        time.sleep(60)
    else:
        print(f"□ Transaction failed: {e}")

finally:
    agent.close()
```

Context Manager

```python
from sdk import UAIP_Enterprise_SDK

with UAIP_Enterprise_SDK("Bot", "Corp", 123) as agent:
    # Multiple transactions
    for invoice in invoices:
        result = agent.call_agent(
            task=f"process_invoice_{invoice.id}",
            amount=invoice.amount,
            intent=f"Payment for invoice {invoice.number}"
        )
        print(f"Invoice {invoice.id}: {result['status']}")

# Automatic cleanup when exiting context
```

Batch Processing

```python
from sdk import UAIP_Enterprise_SDK
import time

agent = UAIP_Enterprise_SDK("BatchBot", "Corp", 123)

invoices = [
    {"id": "INV-001", "amount": 100.00},
    {"id": "INV-002", "amount": 250.00},
    {"id": "INV-003", "amount": 50.00}
]

results = []

for invoice in invoices:
    try:
        result = agent.call_agent(
            task=f"process_{invoice['id']}",
            amount=invoice['amount'],
            intent=f"Payment for {invoice['id']}"
        )
        results.append({
            "invoice": invoice['id'],
            "status": result['status'],
            "request_id": result['request_id']
        })

    except Exception as e:
        results.append({
            "invoice": invoice['id'],
            "status": "ERROR",
            "error": str(e)
        })

    time.sleep(0.1)  # Rate limiting courtesy

# Summary
success = sum(1 for r in results if r['status'] == 'SUCCESS')
print(f"Processed: {success}/{len(invoices)} invoices")
```

Testing

Unit Testing Example

```python
 import unittest
from sdk import UAIP_Enterprise_SDK
from privacy import ZK_Privacy
from compliance import ComplianceAuditor

class TestUAIPSDK(unittest.TestCase):

    def setUp(self):
        self.agent = UAIP_Enterprise_SDK(
            agent_name="TestBot",
            company_name="TestCorp",
            secret_code=12345678,
            auto_register=False
        )

    def tearDown(self):
        self.agent.close()

    def test_agent_creation(self):
        """Test agent DID generation"""
        self.assertIsNotNone(self.agent.did)
        self.assertTrue(self.agent.did.startswith("did:uaip:"))

    def test_zk_proof(self):
        """Test Zero-Knowledge proof generation and verification"""
        secret = ZK_Privacy.generate_secret_key()
        commitment = ZK_Privacy.generate_commitment(secret)
        proof = ZK_Privacy.create_proof(secret, commitment)

        # Valid proof should verify
        self.assertTrue(ZK_Privacy.verify_proof(proof, commitment))

        # Invalid proof should fail
        wrong_commitment = commitment + 1
        self.assertFalse(ZK_Privacy.verify_proof(proof, wrong_commitment))

    def test_compliance_audit(self):
        """Test compliance auditor"""
        auditor = ComplianceAuditor()

        # Clean transaction should pass
        status, report = auditor.run_active_audit({
            "sender": "did:uaip:test:123",
            "task": "process_invoice",
            "amount": 50.00,
            "intent": "Payment"
        })
        self.assertEqual(status, "PASSED")

        # Prohibited keyword should block
        status, report = auditor.run_active_audit({
            "sender": "did:uaip:test:123",
            "task": "money laundering",  # Prohibited
            "amount": 50.00,
            "intent": "Payment"
        })
        self.assertEqual(status, "TERMINATE")

if __name__ == '__main__':
    unittest.main()
```

Changelog

Version 1.0.0 (January 2025)

Initial Release

- □ 5-layer security architecture
- □ Zero-Knowledge proof implementation (Schnorr protocol)
- □ Multi-chain settlement (Base, Solana, Ethereum, Polygon)
- □ RAG-powered compliance auditing
- □ Ed25519 digital signatures
- □ Complete REST API
- □ Python SDK
- □ Real-time dashboard
- □ Comprehensive audit logging
- □ Rate limiting and DoS protection
- □ Connection pooling
- □ Idempotency protection

Support & Resources

Documentation

- **Quick Start Guide:** See Quick-Start-Guide.pdf
- **Architecture Deep Dive:** See Architecture-Deep-Dive.pdf (upcoming)
- **Security & Compliance:** See Security-Compliance-Guide.pdf (upcoming)

Community

- **GitHub:** https://github.com/jahanzaibahmad112-dotcom/UAIP-Protocol
- **Issues:** https://github.com/jahanzaibahmad112-dotcom/UAIP-Protocol/issues
- **Twitter:** @UAIPProtocol
- **Email:** [your-email]

Enterprise Support

For production deployments, security audits, or custom integrations:

- **Email:** [your-email]
- **LinkedIn:** [your-linkedin-profile]

Contributing

See CONTRIBUTING.md in the repository for:

- Development setup
- Code style guidelines
- Testing requirements
- Pull request process

Appendix

Glossary

Agent - Autonomous AI system capable of making decisions and executing transactions

DID (Decentralized Identifier) - Globally unique identifier following W3C standard: `did:uaip:company:hash`

Ed25519 - Elliptic curve signature scheme providing 128-bit security

Zero-Knowledge Proof - Cryptographic method to prove knowledge without revealing information

Schnorr Protocol - Mathematical protocol for generating ZK-proofs

Nonce - Number used once, prevents replay attacks

RAG (Retrieval-Augmented Generation) - AI technique combining retrieval and generation for compliance

USDC - USD Coin, stablecoin pegged 1:1 to US Dollar

Settlement - Final processing of financial transaction on blockchain

Idempotency - Property ensuring same operation produces same result when repeated

Supported Chains

Chain	Type	Currency	Block Time	Finality
Base	L2 (Optimistic Rollup)	USDC	2 seconds	12 blocks
Solana	L1	USDC	0.4 seconds	32 blocks
Ethereum	L1	USDC	12 seconds	12 blocks
Polygon	L2 (Plasma/PoS)	USDC	2 seconds	128 blocks

Fee Tiers Detail

Tier A: Nano Transactions

- Range: $0.01 - $10.00
- Fee: $0.01 flat
- Reasoning: Percentage fees prohibitive at this scale
- Best for: API calls, micro-services, small data exchanges

Tier B: Mid-Range

- Range: $10.01 - $10,000.00
- Fee: 1.0% of amount
- Reasoning: Industry standard for payment processing
- Best for: Standard B2B transactions, service payments

Tier C: Enterprise

- Range: $10,000.01+
- Fee: $10 flat + 0.5% of amount
- Reasoning: Hybrid model reduces percentage burden on large amounts
- Best for: Enterprise licenses, large contracts, treasury operations

---

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