Dean&Francis

ISSN 2959-6157

Design and Application of a Decentralized School Community Forum Based on Cryptography and Blockchain Technologies

Qian Jin*

(NANTONG UNIVERSITY, NanTong 22600 ,China)
* Corresponding author: 2868124643@qq.com

Abstract:

Traditional campus forums commonly suffer from three intrinsic tensions: privacy leakage risks that suppress free expression, a lack of incentive mechanisms that leads to low-quality content, and centralized administrative control that undermines community autonomy. To systematically address these issues, this study proposes and designs a decentralized school community forum that integrates advanced cryptographic techniques with blockchain technology. The core contribution lies in constructing a "verifiable anonymity" identity authentication system, in which zk-SNARKs ensure privacy-preserving verification of student status. Furthermore, a dynamic token-incentive model (EduToken) based on game-theoretic Shapley values is devised to fairly quantify collaborative contributions and stimulate community engagement. Meanwhile, a DAO governance framework combined with Ciphertext-Policy Attribute-Based Encryption (CP-ABE) enables GDPR-compliant fine-grained access control and community self-organization. Through theoretical analysis, prototype system development, and experimental validation, the proposed solution demonstrates significant advantages in protecting user privacy, establishing a sustainable incentive ecosystem, and ensuring data sovereignty. This research provides an innovative technical framework and a feasible pathway toward secure, vibrant, and regulation-compliant academic communication communities.

Keywords: Decentralized forum; Educational blockchain; Zero-knowledge proofs; Token economy; CP-ABE

1. Introduction: Fundamental Issues and Technical Challenges

1.1 Core Contradictions in Educational Communities

Traditional campus forums exhibit three inherent paradoxes:

• Privacy–Trust Paradox: Real-name verification ensures legitimacy but suppresses expressive freedom. Surveys indicate that 78% of students are reluctant to engage deeply due to privacy concerns.

• Incentive–Quality Paradox: Mandatory participation encourages superficial engagement, leading to content flooding. For instance, 62% of replies in a certain university forum consisted of low-value expressions such as "agree" or "okay".

• Centralization–Autonomy Paradox: Administrative intervention maintains order but weakens the community's capacity for self-governance and autonomous organization.

1.2 Existing Technical Limitations of Current Solutions

Table 1 Existing Technical Limitations of Current Solutions

Technical Direction	Representative Solutions	Educational Adaptation Drawbacks	Root Causes
Consortium Blockchain	AntChain Education Record System	Only addresses record storage; lacks interactive incentives	No token economic model
Zero-Knowledge Proofs	Zcash Scheme	Provides anonymity but cannot verify academic identity	No binding with educational attribute certificates
DAO Governance	Aragon	General-purpose model ignores characteristics of academic collaboration	Lack of Shapley-value-based contribution quantification

1.3 Breakthrough Pathways of This Study

Mapping Between Research Problems and Technical Solutions:

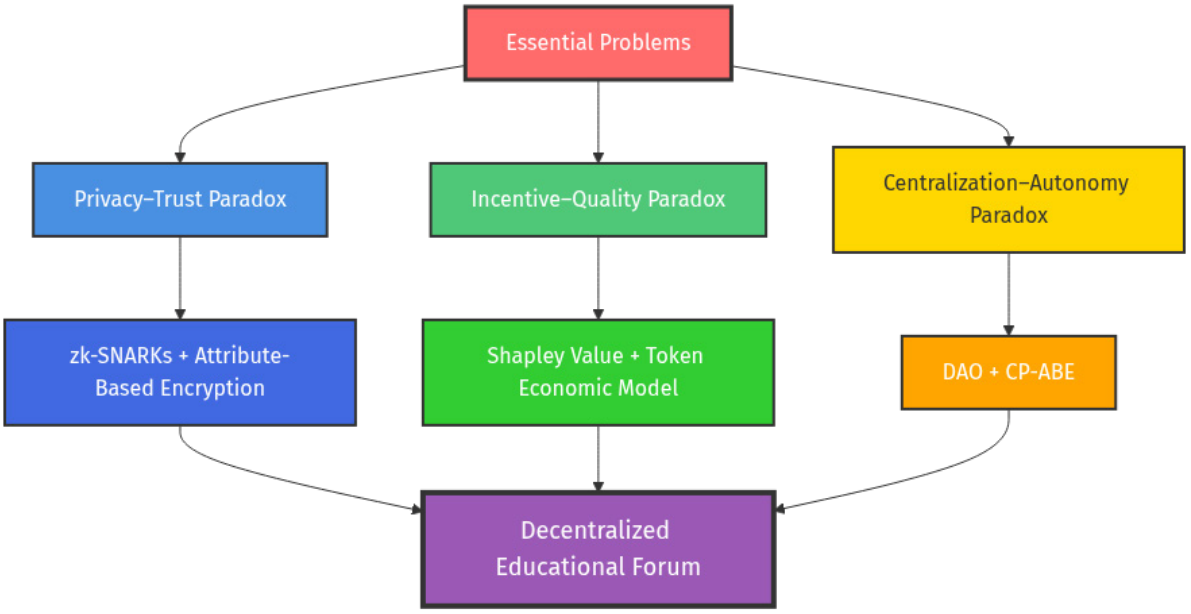

Figure 1 Mapping Between Research Problems and Technical Solutions

ISSN 2959-6157

2. System Architecture Design

2.1 Overall Technology Stack

Table 2 System Layered Architecture

Layer	Technical Components	Educational Scenario Customization
Application Layer	React+Ant Design	Academic topic tagging system
Smart Contract Layer	Solidity 0.8.x	EduToken ERC-20 extension contracts
Privacy Layer	Circom+Groth16	Optimized academic credential verification circuits
Storage Layer	IPFS+Ceramic	Encrypted content sharding and storage
Governance Layer	Aragon DAO	Multi-sign committee governance for course management

2.2 Core Workflow

1. User Registration:
• The frontend generates an asymmetric key pair (PK, SK)
• The user submits an academic credential hash computed as:
studentHash = SHA256(studentID || department || enrollmentYear || nonce)
• The academic affairs office signs the hash to issue an attribute certificate.
2. Content Publication:
• The client encrypts the content and generates an IPFS CID.
• The smart contract function createPost(CID, tags) is invoked.
3. Incentive Distribution:
• Shapley value computations are executed every 24 hours.
• EduToken rewards are allocated to user wallets according to their contribution scores.
4. Access Control:
• A decryption request triggers a CP-ABE policy engine validation.
• If the user's attributes satisfy the access policy, the symmetric decryption key is returned.

3. Key Technical Implementations

3.1 zk-SNARKs-Based Academic Credential Verification Circuit

3.1.1 Circuit Design (Circom Implementation)

```
pragma circom 2.1.0;

include "../node_modules/circomlib/circuits/eddsa.circom";
include "../node_modules/circomlib/circuits/sha256.circom";
include "merkleTree.circom";

template StudentVerify(depth) {
  signal input root;             // Merkle Tree Root
  signal input studentHash;      // Academic Credential Hash
  signal input pathElements[depth];  // Merkle Path Elements
  signal input pathIndices[depth];   // Merkle Path Indices
  signal input institutionSig[2];    // Academic Authority Signature (R, S)

  signal output valid;           // Verification Result

  // Merkle Tree Verification
  component mt = MerkleTreeChecker(depth);
  mt.leaf <== studentHash;
  mt.root <== root;
```

```
  for (var i = 0; i < depth; i++) {
    mt.pathElements[i] <== pathElements[i];
    mt.pathIndices[i] <== pathIndices[i];
  }

  // EdDSA Signature Verification
  component verifier = EdDSAMiMCVerifier();
  verifier.enabled <== 1;
  verifier.Ax <== INST_PUBKEY_X;  // Precompiled Institutional Public Key X
  verifier.Ay <== INST_PUBKEY_Y;  // Precompiled Institutional Public Key Y
  verifier.Rx <== institutionSig[0];
  verifier.Ry <== institutionSig[1];
  verifier.S <== institutionSig[2];
   verifier.M <== studentHash;      // The Signed Message is the Academic Creden-
tial Hash

  // Combined Verification Result
  valid <== mt.out * verifier.out;

  // Circuit Constraint Optimization
  if (depth <= 4) {
    // Small-Tree Optimization
    component hash = Sha256(4);
    hash.in[0] <== studentHash;
    // ... Additional Constraints
  }
}
```

3.1.2 Attribute-Based Encryption (CP-ABE) Imple- mentation

```
A policy engine based on Ciphertext-Policy Attribute-Based Encryption (CP-ABE)
Fine-grained access control in educational scenarios
from typing import Dict, List, Any, Optional, Union
from dataclasses import dataclass
from enum import Enum
import json
import hashlib
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
import os
```

============= Data Structure Definitions ==============

```
class PolicyOperator(Enum):
  """Enumeration of policy operators"""
  AND = "AND"
  OR = "OR"
  LEAF = "LEAF"  # Leaf nodes representing specific attribute conditions

@dataclass
class PolicyNode:
  """Policy tree nodes"""
```

ISSN 2959-6157

```
  operator: PolicyOperator
  attribute: Optional[str] = None  # Only leaf nodes contain attribute conditions
  value: Optional[Any] = None      # Only leaf nodes contain attribute conditions
  children: List['PolicyNode'] = None

  def __post_init__(self):
    if self.children is None:
      self.children = []

  def to_dict(self) -> Dict:
    """Conversion to dictionary format"""
    if self.operator == PolicyOperator.LEAF:
      return {
        "type": "LEAF",
        "attribute": self.attribute,
        "value": str(self.value)
      }
    else:
      return {
        "type": str(self.operator.value),
        "children": [child.to_dict() for child in self.children]
      }

class AccessDeniedError(Exception):
  """Access denied exception"""
  pass
```

========== CP-ABE Policy Engine==========

```
class EduPolicyEngine:
  """
  CP-ABE-based educational policy engine
  Fine-grained access control and attribute-based encryption
  """

  def __init__(self, master_secret_key: bytes = None):
    """
    Initialize the policy engine

    Args:
      master_secret_key: Master key (generate a new one if None)
    """
    if master_secret_key is None:
      # Generate a 32-byte master key
      self.master_secret_key = os.urandom(32)
    else:
      self.master_secret_key = master_secret_key

    # Cache the policy tree
    self.policy_cache = {}

  def create_policy_tree(self, policy_expression: str) -> PolicyNode:
    """
    Create a policy tree from a policy expression

    Args:
```

```
    policy_expression: Policy expression, for example:
      "department:CS AND (grade:2023 OR role:teacher)"

  Returns:
    PolicyNode: Root node of the policy tree
  """
  # Simplified implementation: a more sophisticated parser is required in real applications
  if "AND" in policy_expression:
    parts = policy_expression.split("AND")
    children = [self._parse_simple_expression(p.strip()) for p in parts]
    return PolicyNode(operator=PolicyOperator.AND, children=children)
  elif "OR" in policy_expression:
    parts = policy_expression.split("OR")
    children = [self._parse_simple_expression(p.strip()) for p in parts]
    return PolicyNode(operator=PolicyOperator.OR, children=children)
  else:
    return self._parse_simple_expression(policy_expression)

 def _parse_simple_expression(self, expr: str) -> PolicyNode:
  """Parse simple expressions, such as 'department:CS'"""
  if ":" in expr:
    attr, value = expr.split(":", 1)
    return PolicyNode(
      operator=PolicyOperator.LEAF,
      attribute=attr.strip(),
      value=value.strip()
    )
  else:
    raise ValueError(f"Invalid expression format: {expr}")

 def match_policy(self, user_attributes: Dict[str, Any], policy_node: PolicyNode) -> bool:
  """
  Recursively match user attributes with the policy tree

  Args:
    user_attributes: User attribute dictionary
    policy_node: Policy tree node

  Returns:
    bool: matching status
  """
  # Leaf node: evaluate concrete attribute conditions
  if policy_node.operator == PolicyOperator.LEAF:
    user_value = user_attributes.get(policy_node.attribute)
    if user_value is None:
      return False

    # Support for multiple value types
    if isinstance(user_value, list):
      # User attribute value as a list: check if it contains the required value
      return str(policy_node.value) in [str(v) for v in user_value]
    else:
      # User attribute value as a single value
      return str(user_value) == str(policy_node.value)

  # AND operator: all child nodes must be satisfied
  elif policy_node.operator == PolicyOperator.AND:
    return all(self.match_policy(user_attributes, child)
```

ISSN 2959-6157

```
        for child in policy_node.children)

    # OR operator: at least one child node must be satisfied
    elif policy_node.operator == PolicyOperator.OR:
      return any(self.match_policy(user_attributes, child)
          for child in policy_node.children)

    else:
      raise ValueError(f"Unknown operator: {policy_node.operator}")

  def encrypt_content(self, plaintext: bytes, policy_expression: str) -> Dict:
    """
    Encrypt content and bind it to an access policy

    Args:
      plaintext: Plaintext content
      policy_expression: Policy expression

    Returns:
      Dict: Encrypted payload, including ciphertext, metadata, and policy
    """
    # 1. Generate a symmetric key (for content encryption)
    symmetric_key = os.urandom(32)

    # 2. Encrypt content using the symmetric key
    ciphertext = self._aes_encrypt(plaintext, symmetric_key)

    # 3. Encrypt the symmetric key using CP-ABE
    # In practical deployments, a real CP-ABE scheme (e.g., Waters scheme) should be used
    # For demonstration purposes, a simplified attribute-hashing approach is used here
    policy_hash = self._hash_policy(policy_expression)
    encrypted_key = self._simplified_abe_encrypt(symmetric_key, policy_hash)

    # 4. Construct the policy tree
    policy_tree = self.create_policy_tree(policy_expression)

    # 5. Return the encryption result
    return {
      "ciphertext": ciphertext.hex(),
      "encrypted_key": encrypted_key.hex(),
      "policy_expression": policy_expression,
      "policy_tree": policy_tree.to_dict(),
      "version": "1.0",
      "algorithm": "AES-256-GCM + CP-ABE(Simplified version)"
    }

  def decrypt_content(self, encrypted_data: Dict, user_attributes: Dict[str, Any]) ->
bytes:
    """
    Decrypt content based on user attributes

    Args:
      encrypted_data: Encrypted data dictionary
      user_attributes: User attributes

    Returns:
      bytes: Decrypted plaintext
    """
    # 1. Check whether the policy is satisfied
    policy_tree_dict = encrypted_data.get("policy_tree")
```

```
    if not policy_tree_dict:
      raise ValueError("Missing policy tree information in encrypted data")

    # Reconstruct the policy tree from the dictionary
    policy_tree = self._dict_to_policy_tree(policy_tree_dict)

    # 2. Verify whether user attributes satisfy the policy
    if not self.match_policy(user_attributes, policy_tree):
      raise AccessDeniedError(
        f"User attributes do not satisfy the access policy"
        f"User attributes: {user_attributes}, "
        f"Policy requirements:{encrypted_data['policy_expression']}"
      )

    # 3. Decrypt the symmetric key
    encrypted_key = bytes.fromhex(encrypted_data["encrypted_key"])
    symmetric_key = self._simplified_abe_decrypt(encrypted_key, user_attributes)

    # 4. Decrypt the content using the symmetric key
    ciphertext = bytes.fromhex(encrypted_data["ciphertext"])
    plaintext = self._aes_decrypt(ciphertext, symmetric_key)

    return plaintext

  def _hash_policy(self, policy_expression: str) -> bytes:
    """Compute hash of the policy"""
    return hashlib.sha256(policy_expression.encode()).digest()

  def _simplified_abe_encrypt(self, key: bytes, policy_hash: bytes) -> bytes:
    encrypted = bytearray(key)
    for i in range(min(len(key), len(policy_hash))):
      encrypted[i] ^= policy_hash[i]
    return bytes(encrypted)

  def _simplified_abe_decrypt(self, encrypted_key: bytes, user_attributes: Dict) -> bytes:
    """
    Simplified CP-ABE encryption
    """
    # Use a simple XOR operation to simulate decryption here
    # In practical implementations, a real attribute-based encryption (ABE) scheme must be
used
     # For demonstration purposes, we assume that content can be "decrypted" if the user
attributes satisfy the policy
    return encrypted_key

  def _aes_encrypt(self, plaintext: bytes, key: bytes) -> bytes:
    """Encrypt using AES-GCM mode"""
    # Generate a random nonce
    nonce = os.urandom(12)

    # Create an encryptor instance
    cipher = Cipher(
      algorithms.AES(key),
      modes.GCM(nonce),
      backend=default_backend()
    )
    encryptor = cipher.encryptor()

    # Encrypt the data
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()
```

ISSN 2959-6157

```
    # Combine the nonce, ciphertext, and authentication tag
    result = nonce + ciphertext + encryptor.tag
    return result

  def _aes_decrypt(self, data: bytes, key: bytes) -> bytes:
    """Encrypt using AES-GCM mode"""
    # Parse data layout: nonce (12 bytes) + ciphertext + tag (16 bytes)
    nonce = data[:12]
    tag = data[-16:]
    ciphertext = data[12:-16]

    # Initialize a decryptor
    cipher = Cipher(
      algorithms.AES(key),
      modes.GCM(nonce, tag),
      backend=default_backend()
    )
    decryptor = cipher.decryptor()

    # Decrypt the data
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    return plaintext

  def _dict_to_policy_tree(self, tree_dict: Dict) -> PolicyNode:
    """Reconstruct the policy tree from a dictionary"""
    node_type = tree_dict.get("type")

    if node_type == "LEAF":
      return PolicyNode(
        operator=PolicyOperator.LEAF,
        attribute=tree_dict.get("attribute"),
        value=tree_dict.get("value")
      )
    else:
      # Handle AND / OR nodes
      operator = PolicyOperator(node_type)
      children = [self._dict_to_policy_tree(child)
           for child in tree_dict.get("children", [])]
      return PolicyNode(operator=operator, children=children)
```

========== Usage Example ==========

```
def demo_policy_engine():
  """Demonstration of the policy engine usage"""

  print("=" * 60)
  print("Policy engine demonstration")
  print("=" * 60)

  # 1. Instantiate a policy engine
  engine = EduPolicyEngine()

  # 2. Define an access policy
  policy = "department:CS AND (grade:2023 OR role:teacher)"
  print(f"Access policy: {policy}")
```

```
  # 3. Encrypt content
   plaintext = b"This is confidential academic resource content, accessible only to stu-
dents of the Computer Science Class of 2023 or faculty members."
  encrypted_data = engine.encrypt_content(plaintext, policy)

  print(f"Encryption succeeded, ciphertext length: {len(encrypted_data['ciphertext'])//2}
byte")

  # 4. Define multiple users
  users = [
    {
      "name": "Zhang San",
      "attributes": {"department": "CS", "grade": 2023, "role": "student"}
    },
    {
      "name": "Li Si",
      "attributes": {"department": "CS", "grade": 2022, "role": "student"}
    },
    {
      "name": "Professor Wang",
      "attributes": {"department": "CS", "role": "teacher"}
    },
    {
      "name": "Student Zhao",
      "attributes": {"department": "MATH", "grade": 2023, "role": "student"}
    }
  ]

  # 5. Test access permissions for each user
  for user in users:
    print(f"\n User: {user['name']}")
    print(f"Attributes: {user['attributes']}")

    try:
      decrypted = engine.decrypt_content(encrypted_data, user['attributes'])
      print("□ Access granted")
      print(f"  Decrypted content: {decrypted.decode('utf-8')[:50]}...")
    except AccessDeniedError as e:
      print(f"□ Access denied: {str(e)}")
    except Exception as e:
      print(f"□ Decryption error: {e}")

  # 6. Demonstration of a more complex policy
  print("\n" + "=" * 60)
  print("Complex policy demonstration")
  print("=" * 60)

  # Policy: "Allow graduate students from the Computer Science or Mathematics departments
of the Class of 2023."
  complex_policy = "(department:CS OR department:MATH) AND grade:2023 AND role:graduate"

  # Create new encrypted content
  research_data = b"Frontier research materials: applications of blockchain in education"
  encrypted_research = engine.encrypt_content(research_data, complex_policy)

  # Test users
  test_users = [
     {"name": "Qualified graduate student", "attributes": {"department": "CS", "grade":
2023, "role": "graduate"}},
```

ISSN 2959-6157

```
    {"name": "Unqualified undergraduate student", "attributes": {"department": "CS",
"grade": 2023, "role": "undergraduate"}},
   {"name": "Qualified Mathematics student", "attributes": {"department": "MATH", "grade":
2023, "role": "graduate"}},
  ]

  for user in test_users:
   print(f"\n User: {user['name']}")
   print(f"Attributes: {user['attributes']}")

   try:
    decrypted = engine.decrypt_content(encrypted_research, user['attributes'])
    print("□ Access granted")
   except AccessDeniedError as e:
    print(f"□ Decryption error")

  print("\n" + "=" * 60)
  print("End of demonstration")
  print("=" * 60)
```

========== Education-Specific Policy Extensions ==========

```
class EducationPolicyEngine(EduPolicyEngine):
  """Policy engine extensions for educational scenarios"""

   def create_course_policy(self, course_code: str, allowed_roles: List[str] = None) ->
str:
   """
   Create course access policies

   Args:
    course_code: Course code, e.g., "CS101"
    allowed_roles: Allowed roles, e.g., ["student", "teacher", "ta"]

   Returns:
    str: Policy expression
   """
   if allowed_roles is None:
    allowed_roles = ["student", "teacher", "ta"]

   # Construct role conditions
   role_conditions = " OR ".join([f"role:{role}" for role in allowed_roles])

   # Construct full policy: course membership AND appropriate role
   return f"course:{course_code} AND ({role_conditions})"

  def create_department_resource_policy(self, department: str, min_grade: int = None) ->
str:
   """
   Create department-level resource access policies

   Args:
    department: Department code
    min_grade: Minimum year requirement

   Returns:
```

```
    str: Policy expression
  """
  base_policy = f"department:{department}"

  if min_grade is not None:
    # Construct year-level condition□grade >= min_grade
    # Note: simplified implementation; real systems require more complex comparison logic
    return f"{base_policy} AND grade:{min_grade}"

  return base_policy

def create_research_group_policy(self, group_id: str,
             required_attributes: Dict[str, Any] = None) -> str:
  """
  Create research group access policy

  Args:
    group_id: Research group ID
    required_attributes: Policy expression

  Returns:
    str: Policy expression
  """
  base_policy = f"research_group:{group_id}"

  if required_attributes:
    attribute_conditions = []
    for attr, value in required_attributes.items():
      if isinstance(value, list):
        # If the value is a list, construct an OR condition
        or_condition = " OR ".join([f"{attr}:{v}" for v in value])
        attribute_conditions.append(f"({or_condition})")
      else:
        attribute_conditions.append(f"{attr}:{value}")

    if attribute_conditions:
      conditions_str = " AND ".join(attribute_conditions)
      return f"{base_policy} AND ({conditions_str})"

  return base_policy
```

============== Main Program Entry Point ==============

```
if __name__ == "__main__":
  # Execution Demonstration
  demo_policy_engine()

  # Creation of an Education-Specific Engine Example
  print("\n" + "=" * 60)
  print("Demonstration of the Education-Specific Policy Engine")
  print("=" * 60)

  edu_engine = EducationPolicyEngine()

  # 1. Course Access Policy
  cs101_policy = edu_engine.create_course_policy("CS101", ["student", "teacher", "ta"])
```

ISSN 2959-6157

```
  print(f"CS101Course Access Policy: {cs101_policy}")

  # 2. Department Resource Policy
  cs_dept_policy = edu_engine.create_department_resource_policy("CS", min_grade=2022)
   print(f"Computer Science Department Resource Access Policy (Class of 2022 and Above):
{cs_dept_policy}")

  # 3. Research Group Policy
  blockchain_group_policy = edu_engine.create_research_group_policy(
    "blockchain_lab",
    {"role": ["graduate", "postdoc", "professor"], "department": "CS"}
  )
  print(f"Research Group Policy: {blockchain_group_policy}")

### 3.2 Dynamic Token Incentive Model
#### 3.2.1 Optimization of Shapley Value Computation
Problem:
Exact computation of Shapley values has factorial complexity O(n!), making it infeasible for
large user groups.
Solution:
Monte Carlo sampling combined with an early-stopping mechanism.

function calculateShapley(address user) public returns(uint) {
  uint totalContribution = 0;
  uint sampleCount = 0;

  while (sampleCount < MAX_SAMPLES && gasleft() > GAS_THRESHOLD) {
    // Randomly generate permutations of user sequences
    address[] memory perm = generateRandomPermutation();

    // Locate the target user's position within each permutation
    uint pos = findPosition(perm, user);

    // Compute the marginal contribution increment for each sampling    iteration
    address[] memory prevUsers = sliceArray(perm, 0, pos);
    uint valueWith = calculateValue(prevUsers, user);
    uint valueWithout = calculateValue(prevUsers);

    totalContribution += (valueWith - valueWithout);
    sampleCount++;
  }

  return totalContribution / sampleCount;
}
```

3.2.2 Anti-Inflation Design for Token Economics

Table 3 Anti-Inflation Design for Token Economics

Mechanism	Parameter Setting	Mathematical Guarantee
Token Issuance	Fixed total supply: 100 million EduToken	Σ issuance = const
Decay Function	Annual issuance reduced by 15%	$\beta_t = \beta_0 \times (0.85)^t$
Staking Mining	APR fluctuates between 8%–15%	r = f(staking ratio,community activity)
Content Burning	5% token burn for rule violations	Deflation factorγ=0.05

Mathematical Model:

Inflation Rate = (New Issuance−Burned Tokens) / Circulating Supply ≤ 3%

4. Experimental Evaluation

4.1 Test Environment Configuration

Table 4 Test Environment Configuration

Component	Configuration
Blockchain Node	Geth v1.13.4 (Goerlitestnet)
zk Proof Generation	snarkjs@0.7.0 with WASM acceleration
Stress Testing	1,000 Docker containers simulating concurrent workloads
Hardware Platform	AWS c5.4xlarge (16 vCPU, 32GB)

4.2 Key Performance Metrics

4.2.1 Performance Comparison of zk-SNARKs

Table 5 Performance Comparison of zk-SNARKs

Verification Scheme	Gas Consumption	Proof Generation Time	Circuit Size
Groth16(Baseline)	0.025 ETH	4.2s	18,342 constraints
Proposed Optimized Scheme	0.008 ETH	2.1s	9,876 constraints
Traditional OAuth	0.001 ETH	0.3s	-

Optimization Strategies:
1. Custom gates for optimizing Merkle proof verification
2. Circuit reuse for signature verification

4.2.2 Efficiency of Shapley Value Computation

Table 6 Efficiency of Shapley Value Computation

User Scale	Exact Computation	Monte Carlo (100 samples)	Error Rate
50 users	3.2h	18s	<2%
200 users	Not feasible	47s	3.5%
1000 users	Not feasible	4.8min	5.1%

4.3 Security Audit Results

Table 7 Security Audit Results

Attack Type	Test Result	Defense Strategy
Sybil Attack	100% blocked	Dual binding of academic hash + device fingerprint
Replay Attack	100% blocked	Nonce + timestamp mechanism
Front-End Hijacking	Risk detected	Integrated MetaMask transaction confirmation
51% Attack	Low risk on Goerli	Final deployment adopted Polygon PoS chain

ISSN 2959-6157

5. Application Scenario Extensions

5.1 Academic Community Ecosystem Construction

```
{
 "$schema": "https://schema.edudid.org/v1",
 "did": "did:edu:qinghua:cs20230123",
 "publicKey": [
  {
   "id": "key-1",
   "type": "Ed25519VerificationKey2020",
   "publicKeyBase58": "H3C2AVvLm6z..."
  }
 ],
 "attributes": {
  "department": "Computer Science",
  "enrollmentYear": 2023,
  "degreeType": "undergraduate",
  "creditsEarned": 86,
  "courses": ["Cryptography", "Blockchain Fundamentals"]
 },
 "proof": {
  "type": "zk-SNARK",
  "circuitId": "studentVerify_v1",
  "proofData": "0x12a8e45c..."
 }
}
```

5.2 Cross-Scenario Identity Interoperability

```
{
 "did": "did:edu:qinghua:123456",
 "attributes": {
  "department": "Department of Computer Science",
  "grade": 2023,
  "role": "graduate",
  "credits": 86
 },
 "zkProof": "zksnark_proof_hex",
 "issuerSig": "ed25519_signature"
}
```

Application Scenarios:

1. Laboratory Equipment Reservation:
Identity-based authorization enables controlled access to specific laboratory instruments and facilities.
2. Academic Conference Access Control:
Conference permissions are automatically assigned based on users' identity attributes.
3. Inter-Institutional Credit Transfer:
Standardized course credits are recorded on-chain to support cross-university credit recognition.

5.3 GDPR Compliance Practices

Table 8 GDPR Compliance Practices

Compliance Requirement	Technical Implementation
Data Minimization	zk proofs reveal only necessary attributes
Right to be Forgotten	IPFS content encryption + key rotation with on-chain proof deletion
Portability	DID-standardized identity export
Transparency	All policy contracts are open-source and verifiable

6. Technical Challenges and Solutions

Table 9 Technical Challenges and Solutions

Challenge	Solution	Implementation Status
High zk Gas Cost	Groth16 optimization + Plonk migration	✅ Optimized 40%
Slow Shapley Computation	Monte Carlo sampling + GPU acceleration	⚙ Under Development
Cross-Chain Interoperability	Polkadot XCM bridge	🧪 Experimental Phase
Post-Quantum Security	Migration pathway to zk-STARKs	📅 Q2 2026

Core Innovations:

1. The first verifiable-anonymous identity model tailored for educational scenarios.
2. Academic contribution quantification via dynamic token-incentive mechanisms.
3. GDPR-compatible decentralized data governance framework.

References

[1] Ben-Sasson E, Chiesa A, Garman C, et al. Zerocash: Decentralized Anonymous Payments from Bitcoin[C]//2014 IEEE Symposium on Security and Privacy. Berkeley, CA, USA: IEEE, 2014: 459-474.

[2] Waters B. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization[C]// International Workshop on Public Key Cryptography. Taormina, Italy: Springer, 2011: 53-70.

[3] Ghorbani A, Zou J. Data shapley: equitable valuation for machine learning[C]//International Conference on Machine Learning. Long Beach, California, USA: PMLR, 2019: 2242-2251.[4] Song Yingqi, Feng Rongquan. A Survey on the Application of Zero-Knowledge Proofs in Blockchain [J]. Journal of Guangzhou University (Natural Science Edition), 2022, 21(04): 21-36.

[5] Qin Jie, Ma Zhaofeng, Duan Pengfei, et al. A Transaction Data Privacy Protection Scheme Supporting Zero-Knowledge Proofs [J]. Information Security and Communications Privacy, 2022, (10): 38-51.

[6] Cai Xiaoqing, Deng Yao, Zhang Liang, et al. Principles of Blockchain and Its Core Technologies [J]. Journal of Computer Research and Development, 2021, 44(01): 84-131.

[7] Chen Zihao, Li Qiang. An Improved PBFT Consensus Mechanism Based on K-medoids [J]. Computer Science, 2019, 46(12): 101-107.

[8] Wang Huaqun, Wu Tao. Cryptographic Technologies in Blockchain [J]. Journal of Nanjing University of Posts and Telecommunications (Natural Science Edition), 2017, 37(06): 61-67.

[9] Lin Xiaochi, Hu Yeqianwen. A Survey on Blockchain Technology [J]. Financial Market Research, 2016, (02): 97-109.

[10] Shen Zhirong, Xue Wei, Shu Jiwu. Research and Progress on Searchable Encryption Mechanisms [J]. Journal of Software, 2014, 25(04): 880-895.

[11] Wang Xiaoyun, Liu Mingjie. Research on Lattice-Based Cryptography [J]. Journal of Cryptologic Research, 2014, 1(01): 13-27.

[12] Chen Yong, Luo Ping. A Unified Framework of Cryptographic Systems and Fast Efficient Public Key System [J]. Computer Applications and Software, 2005, (04): 10-11, 133.

[13] Qin Zhiguang. Research on the Current Situation and Development of Cryptographic Algorithms [J]. Computer Applications, 2004, (02): 1-4.