Creating a Secure Transactional Blockchain, with Embedded, Unsupervised, Machine Learning for Suspicious Transaction Detection

Krisztián Bálint

Obuda University, Keleti Károly Faculty of Business and Management,
Department of Business Sciences and Digital Skills,
Tavaszmező street 15-17, 1084, Budapest, Hungary
balint.krisztian@kgk.uni-obuda.hu

Abstract: *Blockchain systems suitable for financial transactions, have existed for more than a decade; however, user trust in such systems remains limited due to concerns related to volatility, governance, validator abuse, and the protection of sensitive transaction data. This paper presents the design and implementation of a transactional blockchain, called KGK–UML, that aims to strengthen technical trust through the integration of multiple security mechanisms and an embedded unsupervised anomaly detection module. The proposed architecture combines SHA-256-based chain integrity, Merkle-root transaction integrity, ECDSA-based validator authentication, Proof-of-Authority consensus with deterministic round-robin leader selection, slashing for improper validator behavior, timestamp drift control, and block-level AES-256-CBC encryption with PBKDF2-SHA256-derived keys. In addition, the blockchain integrates an internal K-Means-based anomaly detector that evaluates transaction patterns using normalized amount and time-of-day features to flag potentially suspicious transactions without sharing data with external third-party AI services. The contribution of this paper is not a new standalone cryptographic primitive or a new anomaly detection algorithm, but a system-level architecture that combines governance, confidentiality, integrity, validator discipline and embedded unsupervised analytics, within a closed blockchain environment. This paper demonstrates the feasibility of implementing such a blockchain and argues that this integrated approach can improve the security posture of transactional blockchains and support stronger user confidence. The limitations of the current study and directions for future empirical validation are also discussed.*

Keywords: blockchain technology; financial transactions; unsupervised machine learning; anomaly detection; secure data storage; proof of authority

1 Introduction

Among blockchains capable of conducting financial transactions, Bitcoin remains the best-known example. Its historical importance is undeniable, yet the broader issue of trust in blockchain-based financial systems remains unresolved. For many

users, distrust is driven not only by market volatility, but also by the lack of institutional guarantees, concerns over abuse by privileged actors, uncertainty regarding the confidentiality of stored data and the absence of transparent security governance [1][2].

In financial systems, trust can be understood as the expectation that the underlying infrastructure and its operators are reliable, secure, and able to keep their promises. In conventional finance, this expectation is often supported by banks, regulators, and established legal frameworks. Transactional blockchains, by contrast, frequently lack a comparable institutional safety layer. Therefore, one possible way to strengthen trust, is to improve the technical safeguards of the blockchain itself, especially in relation to integrity, validator accountability, confidentiality and the early detection of suspicious activity [3][4].

This paper presents the design and implementation of a transactional blockchain called KGK–UML. The proposed blockchain is intended for secure financial transaction processing in a closed environment. Its central idea is that user trust can be supported by combining several established security and governance mechanisms with an embedded unsupervised machine learning module that continuously monitors transactions for anomalous patterns. The anomaly detection task is performed internally so that blockchain data does not need to be transferred to an external analytics provider [5][10].

The blockchain uses a Proof-of-Authority (PoA) consensus mechanism to reduce energy consumption and achieve deterministic validator behavior. Because PoA gives special responsibilities to validators, the design also integrates ECDSA-based signing, leader selection in a round-robin order, slashing for improper behavior, timestamp drift control, block-level encryption, and transaction-level integrity verification based on the Merkle root [11][13].

1.1 Scientific Contribution and Novelty

The scientific contribution of this work lies primarily in the integration of several well-established mechanisms into a unified transactional blockchain architecture designed specifically to enhance technical trust in a closed financial transaction environment [11][13].

The individual components applied in KGK–UML, such as PoA consensus, ECDSA signatures, Merkle trees, slashing, AES-based encryption, and K-Means clustering, are all known in the literature. Accordingly, the novelty of the paper is not algorithmic. Instead, the contribution is system-oriented: the proposed model links validator accountability, transaction confidentiality, block integrity, and built-in unsupervised suspicious transaction detection without requiring external AI data processing.

Thus, the value of the paper lies in demonstrating how these components can be orchestrated within a single operational framework to support secure transactional blockchain operation and to create stronger technical conditions for user trust.

1.2 Research Question

Is it possible to create a blockchain suitable for financial transactions that uses built-in unsupervised machine learning to detect suspicious transactions using a K-Means anomaly detector?

This work is structured as follows:

- KGK – UML chain integrity
- Merkle-Root integration into the blockchain header
- Authenticity and authorization verification within the PoA (Proof of Authority) mechanism extended with ECDSA (Elliptic Curve Digital Signature Algorithm) signature, Creating a PoA leader (round-robin) system
- Application of sanctions in case of incorrect behavior in the KGK-UML blockchain
- Time policy, timestamp manipulation, and latency control creation
- Network mTLS + pinning (stub), creating a signed broadcast
- Encryption solutions in the blockchain called KGK-UML
- Creating Unsupervised Machine Learning in the KGK–UML blockchain to detect suspicious transactions

2 KGK–UML Blockchain Architecture Structure

The spread of digital technologies has increased the complexity of modern business and financial infrastructures. In such systems, stable and reliable operation depends on precise logical rules and strong data protection mechanisms. Blockchain technology is suitable for such use cases because it combines cryptographic integrity, distributed validation, and transparent auditability [6][8].

The KGK–UML blockchain was designed as a secure transactional blockchain in which user trust is strengthened by combining multiple technical safeguards. The architecture is based on three main layers. The first layer applies advanced blockchain security solutions, including SHA-256 chain integrity, Merkle-root transaction integrity, ECDSA signatures, PoA validator governance, slashing,

timestamp control, and block-level encryption. The second layer adds an embedded unsupervised machine learning component for suspicious transaction detection. The third layer enforces proper validator behavior and sanctions abuse where necessary [9][10].

Figure 1 makes the research logic of the KGK-UML blockchain more visible. The upper layer groups the advanced security mechanisms applied at the block and network level, the middle layer shows the embedded K-Means anomaly detector, and the lower layer focuses on validator discipline through leader selection, slashing, and double-sign detection. Presenting the architecture in this layered form helps emphasize that the contribution of the paper is system-level: trust is addressed simultaneously through confidentiality, integrity, behavioral control, and internal suspicious-transaction screening [5,8,11]. Figure 1 below shows the structure of the KGK-UML blockchain from a security perspective.

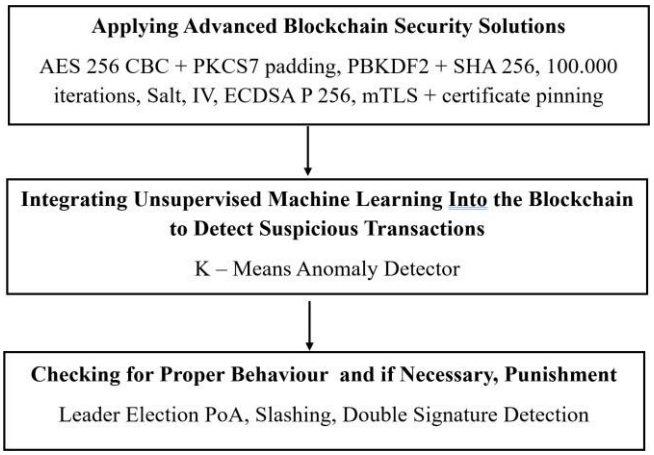


Figure 1. KGK-UML blockchain structure for building user trust and security

Table 1 summarizes the main design elements and their intended protection goals. In revised form, it clarifies how the prototype connects concrete mechanisms - such as Merkle-root verification, validator governance, timestamp control, block-level encryption, and anomaly detection - to the practical security role each element plays inside the KGK-UML blockchain.

Table 1
Principal design elements of the KGK–UML blockchain and their protection goals

Function	Solution	Protection
Chain integrity	SHA-256 + PreviousHash	Detects block-level modification
Transaction integrity	Merkle root in the header	Supports transaction-level integrity and partial verification
Credibility	ECDSA (P-256) signature on the header	Accepts only valid blocks from authorized validators
Eligibility	PoA leader (round-robin)	Provides a deterministic block author
Governance	On-chain validator add/remove with majority signature	Supports secure validator set changes
Sanction	Slashing for double-signing and lateness	Punishes improper validator behavior
Time policy	± 2-minute drift with slot time	Limits timestamp manipulation and excessive delay
DoS protection	Transaction count and size limits	Reduces resource-depletion attempts
Confidentiality	AES-256-CBC per block with PBKDF2-SHA256	Keeps block bodies readable only with a password
Network	mTLS + certificate pinning (architectural stub)	Improves protection against spoofing and MITM
Anomaly detection	K-Means (k = 2), p90 + minority cluster rule	Flags unusual or rare transactions

2.1 Threat Model

This study adopts an explicit threat model describing the main assets, adversaries, assumptions, and mitigation mechanisms considered in the proposed architecture.

2.1.1 Protected assets

- The integrity of the chain and block history
- The integrity of transactions stored in each block
- The authenticity of validator-generated blocks
- The confidentiality of encrypted transaction payloads
- The availability of the blockchain against basic resource-exhaustion attempts
- The correctness and timeliness of validator behavior during block creation

2.1.2 Adversary Model

- A malicious validator attempting to sign invalid blocks, double-sign blocks, or delay block creation
- A network-level attacker attempting to spoof or perform packet manipulation during inter-node communication
- An unauthorized external party attempting to inspect block contents without the correct password
- A participant generating unusual transaction patterns that may indicate suspicious financial behavior

2.1.3 Assumptions

The model assumes that standard cryptographic primitives such as SHA-256, ECDSA P-256, AES-256-CBC, and PBKDF2-SHA256 are implemented correctly and remain computationally secure under normal usage assumptions. It is also assumed that validator public keys are known to the network and that the validator set is managed through the governance logic of the blockchain [13].

2.1.4 Threats and Mitigations

Chain tampering is mitigated by hash chaining using SHA-256 and the PreviousHash field. Transaction-level manipulation is mitigated by storing the Merkle root in the block header. Unauthorized block creation is mitigated by ECDSA-based validator authentication and deterministic PoA leader selection. Improper validator behavior, including double-signing and excessive lateness, is mitigated through slashing. Timestamp manipulation is constrained by the ±2-minute drift policy. Confidentiality of block payloads is protected through per-block AES-256-CBC encryption with PBKDF2-SHA256 key derivation, random

salt, and random IV generation. Finally, unusual transaction behavior is addressed by the built-in unsupervised anomaly detector based on K-Means clustering [11][13].

2.2 Scope

The present work focuses on implementation-level protections and architectural defenses. It does not claim to provide formal cryptographic proofs, resistance against all advanced adaptive adversaries, or a production-grade network stack. The network mTLS and certificate pinning layer is currently introduced as a stub-level architectural element rather than as a fully benchmarked deployment-ready subsystem.

3 Core Security and Governance Mechanisms

3.1 Blockchain Integrity

Chain integrity is a mandatory property of every blockchain. In KGK–UML, the header includes ChainId, protocol version, block index, timestamp, previous hash, and the Merkle root. This design ensures that modifying a block, changes both its own hash and the integrity of subsequent blocks. The PreviousHash field preserves chronological linkage across the chain [12].

3.2 Merkle-Root Integration

Transactions are represented through a Merkle tree, whose root is stored in the block header. This approach offers two advantages. First, the signed header protects all transactions indirectly through the root hash. Second, clients can verify specific transactions using simplified payment verification, because only the block header and the relevant Merkle proof are needed, not the full block body [12].

3.3 Authenticity and Authorization with PoA and ECDSA

The network follows a Proof-of-Authority logic in which exactly one validator is authorized to sign a block at a given height. The header is signed using ECDSA P-256. The private key remains with the validator, while other participants verify the signature using the public key. This mechanism strengthens authenticity, non-repudiation and integrity [13].

3.4 Deterministic Leader Selection

The leader selection logic follows a round-robin strategy. At each block height, the next validator in the ordered validator list becomes the designated block author. This reduces ambiguity, prevents validator races, and supports a lower-energy consensus model than Proof-of-Work [11].

3.5 Slashing and Validator Discipline

Because PoA places considerable trust in validators, the system includes a slashing mechanism to punish improper behavior. The current design reacts to double-signing and excessive lateness. A slashed validator experiences a reduction in stake and receives a recorded behavioral penalty. The main goal is not merely punishment, but the deterrence of behavior that erodes trust in the blockchain [11].

3.6 Timestamp Drift Control

Timestamp manipulation is constrained by a time policy with a permitted ±2-minute drift. If a newly created block falls outside the accepted time window, the block is rejected. This policy limits deliberate delay and helps maintain predictable block production [11].

3.7 Confidentiality and Encryption

The KGK–UML blockchain protects block payload confidentiality through per-block AES-256-CBC encryption. In this design, the payload is encrypted before the block is added to the chain, and the encrypted content is stored in the EncryptedPayloadB64 field. The corresponding decryption step requires the correct user-supplied password, which means that the plaintext block body is not directly exposed to other participants, without authorization.

To strengthen password-based protection, the encryption key is derived using PBKDF2-SHA256 rather than being used directly from the password itself. The derivation process includes a random 16-byte salt for each block, which helps defend against precomputed dictionary and rainbow-table attacks. In addition, a random 16-byte initialization vector is generated for every block. As a result, even if the plaintext and password remain the same, the ciphertext differs across blocks.

This confidentiality layer is important because the KGK–UML blockchain is intended for sensitive financial transaction scenarios. The purpose of the encryption design is not to replace all possible data-protection controls, but to add

an internal confidentiality safeguard that complements integrity protection, validator authentication, and behavioral enforcement [13].

3.8 Network Protection

The architectural design includes mutual TLS and certificate pinning to reduce the risk of spoofing and man-in-the-middle attacks. In the present paper, this layer is introduced as an architectural deployment stub rather than as a fully benchmarked production-ready subsystem [13].

4 Prototype Implementation of KGK–UML

This section presents selected implementation details of the KGK–UML blockchain prototype. While the previous section summarized the core security and governance mechanisms at a conceptual level, the present section illustrates how these mechanisms were translated into working program logic. Only the most relevant code excerpts are retained in the main text in order to support technical transparency without reducing readability.

4.1 Merkle-Root Integration into the Header

The blockchain stores transactions in a Merkle-based structure. In contrast to a simple approach that hashes an entire serialized transaction list as one text block, the KGK–UML design hashes each transaction separately, builds a binary hash tree, and places the Merkle root in the block header. This allows block signing to protect all transactions indirectly and enables simplified payment verification with only the block header and the relevant Merkle path [12]. Figure 2 below illustrates the canonical byte representation of the header.

```
public byte[] HeaderBytes()
{
   var                                                    header                                                    =
$"{ChainId}|v{ProtoVersion}|{Index}|{Timestamp:O}|{PreviousHash}|{MerkleRoot}";
   return Encoding.UTF8.GetBytes(header); // MerkleRoot in the header
}
```

Figure 2. Canonical byte representation of the header

The header is then hashed and signed to create a secure block header, which is shown in Figure 3.

```
block.Hash = block.CalculateHash(); // SHA-256 in the header
block.Signature = SignHeader(block, leader); // ECDSA
```

Figure 3. Block hashing and signing

Using this method, the Merkle root becomes part of the signed header. Therefore, a client only needs the block header and the Merkle path of a given transaction to recompute the root locally and compare it with the Merkle root stored in the header.

4.2 Authenticity and Authorization Verification with ECDSA

The network follows a PoA logic in which the defined leader at each height signs the header bytes using its private key, while the other participants verify the signature using the corresponding public key. The implementation relies on ECDSA P-256 [13]. Figure 4 below illustrates the validator keys (ECDSA) solution.

```
public class Validator
{
   public string Name { get; set; }
   public ECDsa Key { get; private set; } // private key (signature)
   public byte[] PublicKeyDer { get; private set; } // public key (verification)
   public Validator(string name)
   {
      Name = name;
      Key = ECDsa.Create(ECCurve.NamedCurves.nistP256);
      PublicKeyDer = Key.ExportSubjectPublicKeyInfo();
   }
}
```

Figure 4. Validator keys (ECDSA)

In the KGK–UML blockchain, each validator has its own ECDSA P-256 key pair. Figure 5 below shows header signing with ECDSA.

```
var leader = GetLeaderForBlock(block.Index);
block.Signer = leader.Name;
block.Hash = block.CalculateHash(); // SHA-256 from the header
block.Signature = SignHeader(block, leader); // ECDSA signature
```

Figure 5. Header signing with ECDSA

After signature generation, the block can be re-verified. Figure 6 below illustrates a signature verification ECDSA solution.

```
private bool VerifyHeader(Block block, Validator v)
{
   using var ecdsa = ECDsa.Create();
   ecdsa.ImportSubjectPublicKeyInfo(v.PublicKeyDer, out _);
   return ecdsa.VerifyData(block.HeaderBytes(), block.Signature, HashAlgorithmName.SHA256);
}
```

Figure 6. Signature verification (ECDSA verify)

4.3 PoA Leader (Round-Robin) System

The choice of PoA was motivated by the intention to reduce energy consumption and maintain deterministic validator behavior. For each block, exactly one validator is authorized to create and sign the block. The order changes in round-robin form so that each validator receives a turn [11]. Figure 7 below shows the KGK–UML method that calculates the next leader.

```
public Validator GetLeaderForBlock(int blockIndex)
{
   int n = Validators.Count;
   if (n == 0) throw new InvalidOperationException("No validators.");
   int leaderIdx = (genesisLeaderIndex + blockIndex) % n;
   return Validators[leaderIdx];
}
```

Figure 7. KGK–UML method that calculates the next leader

4.4 Slashing Based on Incorrect Behavior

Slashing is a core disciplinary mechanism in the blockchain. Because KGK–UML uses a PoA model, trust partly depends on the proper behavior of validators. The design therefore monitors double-signing and lateness. In the case of improper behavior, the validator is penalized [11]. Figure 8 below, shows the validator penalty execution.

```
private void Slash(Validator v, string reason)
{
   if (v == null) return;
   v.Stake = Math.Max(0, v.Stake - 10); // cash/stake withdrawal
   v.IsSlashed = true;
   if (reason == "double-sign")
      doubleSignCount[v.Name] = doubleSignCount.GetValueOrDefault(v.Name) + 1;
   else if (reason == "lateness")
      lateCount[v.Name] = lateCount.GetValueOrDefault(v.Name) + 1;
   Console.WriteLine($"⚠ SLASH: {v.Name} for {reason}. New stake={v.Stake}");
}
```

Figure 8. Validator penalty execution

4.5 Time Policy, Timestamp Manipulation and Latency Control

A further protection mechanism addresses timestamp manipulation and block delay. The first step is to define the allowed drift. Figure 9 below shows the ±2-minute drift setting.

```
public const string ChainId = "KGK-UML";
public const int ProtocolVersion = 2;
// Time / slots
public static readonly TimeSpan SlotDuration = TimeSpan.FromSeconds(15);
public static readonly TimeSpan AllowedClockDrift = TimeSpan.FromMinutes(2);
```

Figure 9. Setting the ±2-minute drift time

The timestamp is then checked against the allowed policy. Figure 10 below illustrates the Timestamp check.

```
if (Math.Abs((block.Timestamp - DateTime.UtcNow).TotalSeconds) >
   ChainConfig.AllowedClockDrift.TotalSeconds)
{
   Console.WriteLine("Block timestamp outside allowed drift.");
   return false;
}
```

Figure 10. Timestamp check

4.6 Encryption of Block Payloads

Before a block is added to the chain, its transaction payload can be encrypted using AES-256-CBC with a PBKDF2-SHA256-derived key, random salt, and random IV [13]. Figure 11 below shows the method of encrypting block data transactions by block.

```
public static string EncryptToB64(string plaintext, string password, out string saltB64, out string ivB64)
{
   saltB64 = GenerateSaltB64(16);
   byte[] salt = Convert.FromBase64String(saltB64);
   byte[] key = DeriveKey(password, salt);
   byte[] iv = new byte[IvBytes];
   RandomNumberGenerator.Fill(iv);
   ivB64 = Convert.ToBase64String(iv);
   using var aes = Aes.Create();
   aes.KeySize = 256;
   aes.BlockSize = 128;
   aes.Mode = CipherMode.CBC;
   aes.Padding = PaddingMode.PKCS7;
   aes.Key = key;
   aes.IV = iv;
   using var enc = aes.CreateEncryptor();
   byte[] pt = Encoding.UTF8.GetBytes(plaintext);
   byte[] ct = enc.TransformFinalBlock(pt, 0, pt.Length);
   return Convert.ToBase64String(ct);
}
```

Figure 11. Encrypting block transaction data block by block

5 Embedded Unsupervised Machine Learning for Suspicious Transaction Detection

The KGK–UML blockchain contains an embedded anomaly detector that continuously evaluates transaction data. The purpose of this module is not to replace a full fraud analytics platform, but to provide an internal, privacy-preserving, first-layer screening mechanism inside the blockchain [10].

The detector uses two features derived from each transaction: the normalized transaction amount and the time of day normalized by the 24-hour cycle. As a result, each transaction is represented as a point in two-dimensional feature space.

The K-Means algorithm is used with k = 2 clusters. During clustering, each transaction is assigned to the nearest centroid according to Euclidean distance. After repeated assignment and centroid updates, the detector computes the distance of each point from its own centroid.

A transaction is flagged as suspicious if at least one of two conditions holds. First, the transaction lies far from the center of its assigned cluster according to the 90^{th} percentile distance threshold. Second, the transaction belongs to a minority cluster whose size is small enough to represent rare behavior. The blockchain then stores a Boolean anomaly label and a numeric anomaly score for each transaction.

5.1 Feature Construction and Data Representation

From a methodological perspective, the detector relies on lightweight behavioral features that can be extracted directly from transaction metadata without sending data to any external analytics service. This choice is consistent with the closed-system philosophy of the KGK–UML blockchain, where confidentiality and local processing are prioritized [10].

The current implementation represents each transaction as a two-dimensional vector. The first feature is the normalized transaction amount produced by min–max scaling, while the second feature is the normalized hour-of-day. This representation was selected because the amount and transaction time are intuitive first-order indicators of abnormal payment behavior and can be computed with very low overhead inside a blockchain environment.

5.2 Operational Interpretation

Operationally, the embedded detector should be interpreted as a supportive control rather than an autonomous decision-maker. Its purpose is to enrich the blockchain with a machine-derived suspiciousness signal that can support validators, auditors, or later compliance modules. A flagged transaction should therefore be understood as unusual or statistically rare, but not automatically fraudulent in a legal or financial sense.

This distinction is essential for scientific interpretation. Unsupervised learning can identify deviation without requiring labeled data, which is a practical advantage in environments where trusted fraud labels are not readily available. However, statistical irregularity and confirmed malicious behavior are not identical, and the output of the module must be interpreted in context [10].

5.3 Discussion of the K-Means Module

The detector has clear advantages: it operates locally, avoids external data sharing, and can be integrated directly into transaction processing logic. At the same time, the current version has important limitations. It relies on only two features, uses a fixed number of clusters, and employs heuristic decision rules based on percentile distance and minority-cluster membership [10].

Therefore, the anomaly label should be interpreted as a suspiciousness signal rather than as definitive proof of fraud. Within the scope of the present study, the detector should therefore be interpreted as a feasibility-level embedded monitoring component that demonstrates architectural integration, rather than as a complete production-grade fraud-detection solution. Figure 12 shows the implementation logic of the KMeansAnomalyDetector module in the blockchain.

```
{
  public class KMeansAnomalyDetector
  private readonly int k;
  private readonly int maxIterations;
  private double[][] centroids;
  public KMeansAnomalyDetector(int k = 2, int maxIterations = 50)
  {
    this.k = k;
    this.maxIterations = maxIterations;
  }
  public void FitAndTag(List<Transaction> txs)
  {
    if (txs == null || txs.Count == 0) return;
    if (txs.Count < 2)
    {
      var t = txs[0];
      t.IsAnomalous = false;
      t.AnomalyScore = 0;
      return;
    }
    var amounts = txs.Select(t => t.Amount).ToArray();
    var minA = amounts.Min();
    var maxA = amounts.Max();
    double Range(double x) => (maxA - minA) == 0 ? 0.0 : (x - minA) / (maxA - minA);
    double[][] X = txs.Select(t =>
    {
      var f1 = Range(t.Amount);
      var f2 = t.Timestamp.Hour / 23.0;
      return new[] { f1, f2 };
```

```
}).ToArray();
var rnd = new Random(42);
int kUse = Math.Min(k, X.Length);
centroids = new double[kUse][];
var used = new HashSet<int>();
for (int i = 0; i < kUse; i++)
{
   int idx;
   do idx = rnd.Next(X.Length); while (!used.Add(idx));
   centroids[i] = (double[])X[idx].Clone();
}
int[] labels = new int[X.Length];
for (int iter = 0; iter < maxIterations; iter++)
{
   bool changed = false;
   for (int i = 0; i < X.Length; i++)
   {
      int best = 0;
      double bestDist = Dist2(X[i], centroids[0]);
      for (int c = 1; c < kUse; c++)
      {
         var d = Dist2(X[i], centroids[c]);
         if (d < bestDist) { bestDist = d; best = c; }
      }
      if (labels[i] != best) { labels[i] = best; changed = true; }
   }

   var sums = new double[kUse][];
   var counts = new int[kUse];
   for (int c = 0; c < kUse; c++) sums[c] = new double[2];
   for (int i = 0; i < X.Length; i++)
   {
      counts[labels[i]]++;
      sums[labels[i]][0] += X[i][0];
      sums[labels[i]][1] += X[i][1];
   }
   for (int c = 0; c < kUse; c++)
   {
      if (counts[c] > 0)
      {
         centroids[c][0] = sums[c][0] / counts[c];
         centroids[c][1] = sums[c][1] / counts[c];
      }
   }
```

```
            if (!changed) break;
        }
        var dists = new double[X.Length];
        for (int i = 0; i < X.Length; i++)
            dists[i] = Math.Sqrt(Dist2(X[i], centroids[labels[i]]));
        var sorted = dists.ToArray();
        Array.Sort(sorted);
        double p90 = sorted[(int)Math.Floor(0.90 * (sorted.Length - 1))];
        var clusterCounts = new int[kUse];
        foreach (var lab in labels) clusterCounts[lab]++;
        int minorityCluster = Array.IndexOf(clusterCounts, clusterCounts.Min());
        for (int i = 0; i < txs.Count; i++)
        {
            bool far = dists[i] >= p90;
            bool rare = labels[i] == minorityCluster && clusterCounts[minorityCluster] <= txs.Count *
0.2;
            txs[i].IsAnomalous = far || rare;
            txs[i].AnomalyScore = dists[i];
        }
    }
    private static double Dist2(double[] a, double[] b)
        => (a[0] - b[0]) * (a[0] - b[0]) + (a[1] - b[1]) * (a[1] - b[1]);
}
```

Figure 12. K-Means anomaly detection using embedded unsupervised machine learning

6 Evaluation Considerations and Limitations

6.1 Pilot Evaluation on Synthetic Transaction Data

To complement the architectural discussion, this paper outlines a conservative pilot-evaluation framework based on synthetic transaction data. Such a framework may be used to create regular transactions together with intentionally unusual transactions that emulate anomalous behavior in terms of transaction amount and transaction timing. The purpose of including this framework is methodological: it shows how the embedded detector can be assessed without claiming benchmark results that have not yet been validated on external real-world datasets. [10]

In such a pilot setting, the implementation described in this paper - using k = 2 clusters, Euclidean distance, and the p90 threshold - can be evaluated with

standard detection metrics, including precision, recall, F1-score, false positive rate, and false negative rate. Reporting these values is important because the detector should be assessed not only by whether it can identify unusual points, but also by the frequency with which it may over-flag normal transactions.

Computational overhead should also be evaluated in the same pilot framework by measuring the time required to process complete transaction batches under controlled conditions. Because the detector is intended to operate inside the blockchain environment, runtime measurements are relevant for determining whether the anomaly-screening step remains compatible with acceptable throughput and validator-side processing constraints.

A synthetic pilot study would still remain limited in scope because it would rely on generated data rather than large, naturally occurring financial datasets. For that reason, any pilot results obtained in such a setting should be interpreted as feasibility-oriented indicators rather than as definitive evidence of production-grade fraud-detection performance.

6.2 Detection Quality

A stronger evaluation should report standard detection metrics such as precision, recall, F1-score, false positive rate, and false negative rate. Where real labeled financial data are unavailable, the study could still use a controlled synthetic dataset or scenario-based simulation containing both regular and intentionally suspicious transaction patterns.

6.3 Robustness and Sensitivity

Because the detector depends on configuration choices such as $k = 2$, the p90 threshold, and the selected feature set, a sensitivity analysis would be valuable. Such an analysis could examine how stable the anomaly labels remain when the number of clusters, threshold level, or transaction mix changes. This would make the detector's behavior easier to justify scientifically [10].

6.4 Computational Overhead

Since the anomaly detector is embedded into the blockchain environment, runtime cost is not merely a secondary implementation detail. The paper would be strengthened by reporting the time required for batch processing, approximate memory usage, and the effect of increasing transaction volume on throughput. These measurements would help clarify whether the current method remains practical as transaction frequency grows [11].

6.5 Practical Utility

Beyond statistical accuracy, the practical usefulness of the detector depends on whether flagged transactions provide meaningful support for validator awareness and follow-up review. In a fuller study, this could be discussed through case examples, confusion analysis, or an examination of how often alerts correspond to genuinely informative events rather than routine but rare user behavior [3][5].

The current study also has further limitations. It does not provide a formal proof-based security analysis. The network protection layer is described at the architectural level, but not yet evaluated in a production-grade distributed deployment. Therefore, the reported results should be read as an initial feasibility-oriented step rather than as complete production validation.

7 Conclusions

This paper presents the design and implementation of the KGK–UML blockchain, a transactional blockchain architecture that combines integrity protection, validator authentication, confidentiality mechanisms, validator-discipline controls, and embedded unsupervised anomaly detection.

The study shows that it is technically feasible, to construct a blockchain that integrates SHA-256-based chain integrity, Merkle-root transaction verification, ECDSA-authenticated PoA consensus, slashing, timestamp drift control, AES-based block encryption and internal K-Means-based suspicious transaction detection, in a unified framework.

The principal contribution of the work is architectural and implementation-oriented. Rather than proposing a new cryptographic primitive or a new clustering algorithm, the paper demonstrates how multiple established security and governance mechanisms can be combined with an internal unsupervised learning module to support more secure transactional blockchain operation in a closed environment.

At the same time, the present study has limitations. The anomaly detector is described together with a cautious pilot-evaluation framework, but it has not yet been tested on a large labeled real-world dataset.

Future work should include validation on real or institutionally curated transaction datasets, comparative experiments with alternative anomaly detection methods, broader sensitivity analysis, and quantitative reporting of precision, recall, false-positive rate and runtime overhead under realistic deployment conditions.

References

[1] Panda, S. K., Sathya, A. R., & Das, S.: Bitcoin: Beginning of the cryptocurrency era. In Recent advances in blockchain technology: Real-world applications. Cham: Springer International Publishing, 2023, pp 25-58.

[2] Malik, V.: The history and the future of Bitcoin. Praha: Bankovní institut vysoká škola Praha, 2016, pp.1-43.

[3] Van der Cruijsen, C., De Haan, J., & Roerink, R.: Trust in financial institutions: A survey. Journal of economic surveys, 37(4), 2023. pp. 1214-1254.

[4] Csiszárik-Kocsir, Á.: The Purposes and Motivations of Savings Accumulation based on Generational Affiliation Financial Education and Financial Literacy. Acta Polytechnica Hungarica, 20(3), 2023, pp. 195-210.

[5] Han, H., Shiwakoti, R. K., Jarvis, R., Mordi, C., & Botchie, D.: Accounting and auditing with blockchain technology and artificial Intelligence: A literature review. International Journal of Accounting Information Systems, 48, 100598, 2023, pp. 1-16.

[6] Tick, A.: Industry 4.0 narratives through the eyes of SMEs in V4 countries, Serbia and Bulgaria. Acta Polytechnica Hungarica, 20(2), 2023, pp.83-104.

[7] Tick, A., & Szabó-Harka, N.: Analysis of Information Security in a Corporate Environment–a Human Perspective. In 2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics, SAMI pp. 469-474.

[8] Csiszárik-Kocsir, Á., Varga, J., & Fodor, M.: The value based analysis of the financial culture. The journal of macro trends in social science, 2(1), 2016, pp. 89-100.

[9] Molnár, G., Cserkó, J., Nagy, E., & Balogh, Z.: Creating own AI datasets from different language sources efficiently: Advantages, Disadvantages and Best Practices to Make Own Datasets. In 2023 IEEE 6th International Conference and Workshop Obuda on Electrical and Power Engineering, CANDO-EPE, pp. 155-158.

[10] Joshi, S.: Feasibility of proof of authority as a consensus protocol model. arXiv preprint arXiv:2109.02480, Computer Science, Distributed, Parallel, and Cluster Computing 2021, pp. 1-5.

[11] Sheth, H., & Dattani, J.: Overview of blockchain technology. Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146, 2019, pp. 1-3.

[12] Moreira, M. A. R.: ECDSA (Elliptic Curve Digital Signature Algorithm), Specialization in Information Security, Uniminas, Educational Union of Minas Gerais , Uberlândia, MG Brazil, 2006.