ChatGPT

Fraud Surfaces in Multi-Leg Payment Systems

Complex payment flows like AEPS and RuPay split each transaction into multiple steps (authentication, debit leg, credit leg, settlement) [1] [2]. This multi-leg design creates many points of attack. **Authentication:** stolen or spoofed credentials (biometric, PIN) let an attacker impersonate a customer (e.g. "fake Aadhaar" or cloned fingerprints [3]). **Transaction messaging:** attackers may manipulate the sequence of legs – for example triggering a debit without a matching credit or omitting a step (a debit must always precede its credit [1]). **Timeouts and reversals:** Fraudsters can exploit delays (e.g. network lag or ATM glitches) to force a timeout, causing the system to reverse a transaction while still dispensing funds [4] [5]. **Settlement and pooling:** In AEPS Off-us and RuPay Off-us, funds flow through intermediary "pool" accounts. Inconsistent or missing settlement postings (e.g. uncredited pooling accounts) can hide theft or lead to fund "leakage" between banks [6] [7]. **Identity abuse:** Synthetic or stolen IDs (mule accounts) allow coordinated attacks; for example, a ring of accounts colluding to launder money or exhaust overdraft lines. **Sequence manipulation:** Skipping or reordering messages, replaying old messages, or injecting fake confirmations can subvert ledger consistency. **Timeout exploitation:** Deliberately interrupting a leg (e.g. by jamming a POS terminal or ATM) to trigger a reversal on a completed leg [8] [9]. **Reversal abuse:** Repeating the same reversal request or creating false "last status" checks so that funds are never debited while credits are posted. **Settlement discrepancies:** Tampering with the final clearing entries (e.g. falsifying batch totals or delaying uploads) causes mismatches between what NPCI thinks is owed and what banks report. Finally, **coordinated attacks** (e.g. mule networks, social-engineered collusion between Business Correspondents and TSPs) exploit human or systemic trust to scale fraud beyond isolated transactions.

Together, these vulnerabilities span the entire flow – from customer authentication (biometric/PIN) through to post-settlement reconciliation. For example, as in AEPS On-us, the gateway first sends an Aadhaar+biometric request to NPCI/UIDAI, then only posts to the bank if that succeeds [1]. If the biometric auth fails or times out, the process halts [10]. In contrast, in RuPay On-us the gateway splits a request into a PIN validation leg (sent to the ATM switch) and a transaction leg (sent to the bank) [2]. Any failure of the PIN step aborts the transaction (no debit is attempted) [5]. In all cases, such logic must be enforced carefully. In summary, fraud surfaces include the **authentication step**, **messaging sequence**, **credit/debit posting**, **timeout/reversal logic**, and **settlement postings** – each of which can be attacked or subverted in realistic scenarios.

Identity & Authentication Fraud

Fraudsters exploit weak or spoofed authentication. In AEPS systems, attackers may use **fake biometric data or stolen Aadhaar details** to authenticate as a victim [3]. For RuPay and card-based systems, PIN theft or skimming attacks achieve the same end. **Machine-learning defenses:** Supervised classifiers (e.g. gradient-boosted trees or neural nets) can score each authentication attempt for risk based on features like device fingerprint, historical behavior, location, and transaction context. For instance, a model can learn patterns of legitimate biometric scans (pressure, sensor noise) versus known spoofs. **One-class or anomaly models** are also useful: an autoencoder or one-class SVM can learn the normal distribution of a user's authentication characteristics and flag deviations (biometric minutiae, facial features, or typing/voice patterns) as anomalies. Modern CNN-based liveness-detection systems (already used by UIDAI) classify

fingerprint or face images as “live” or “cloned” [11] [12]. These can be trained on image datasets of genuine vs fake prints/faces.

Hard-rule invariants: Enforce multi-factor or step-up checks. For example, require OTP confirmation for large withdrawals or transactions after repeated failures. Enforce that the **gateway only forwards a transaction leg if authentication succeeded** – i.e. reject any financial leg without a preceding successful bio/PIN leg [1] [5]. Maintain a strict count of failed attempts per user/card (lockout after N tries). Use device binding (e.g. only allow known POS devices for a given user) and IP/location restrictions (geofencing). In AEPS, UIDAI’s FMR-FIR liveness check (fingerprint minutiae) and similar AI-driven face recognition are examples of embedding biometric anti-spoof checks [11] [12]. In practice, one would hard-code that **no transaction leg** is sent unless and until the biometric/PIN leg returns OK – any violation is a protocol-level fraud. These invariants can be monitored as simple rules in the gateway or bank switch.

Transaction Flow & Sequence Manipulation

Transactions in AEPS/RuPay follow strict sequences (e.g. authenticate→debit→credit). An attacker might try to “skip” or replay parts of this sequence. **Surface:** for instance, sending a duplicate debit request without matching the previous credit, or manipulating the “last transaction status” check (LTS) messages between POS and gateway [4] [5]. **ML models:** Sequence or graph models can learn the normal ordering of steps. Recurrent neural nets (LSTM/GRU) or Hidden Markov Models can ingest streams of message events (AUTH, DEBIT, CREDIT, RESPONSE) and output an anomaly score if the observed sequence deviates. Alternatively, anomaly detectors (Isolation Forest, LOF) can flag unusual transitions (e.g. a credit request with no prior debit in logs). **Graph-based learning** (e.g. a knowledge graph of transaction states per session) can also be applied.

Rule invariants: Enforce state-machine logic: e.g. the system should reject any **credit posting that does not follow a successful debit**. Every transaction must carry a unique session ID or reference; maintain a log table where the debit and credit legs share this ID. Hard rules include “no second leg if first leg failed” and “no outgoing funds unless the account has first been debited”. In practice, the core banking or switch system should check that each inbound credit corresponds to a previous debit reference – violations indicate tampering. Similarly, “end-to-end confirmations” (like signed tokens or sequence numbers) can ensure message integrity and ordering.

Timeout Exploitation & Reversal Abuse

Attackers exploit timeouts to get funds without debit. For example, in ATM reversal fraud (PIN reversal), a criminal causes an ATM to deliver cash, then forces a timeout so the system issues a reversal [8] [9]. In payment networks, if the transaction leg times out, the gateway may auto-generate a reversal (credit back) while the cash has already left the system [4] [5].

Detection models: Monitor timeout/reversal patterns. Time series or anomaly detection on the frequency of timeouts per account, per terminal, or per timeframe can catch abnormal clusters of reversals. For instance, if a customer’s transactions see an unusual spike in “91 Timeout” responses, an anomaly score should rise. An unsupervised autoencoder trained on transaction metadata (amounts, locations, response codes) can flag transactions with an anomalously high reversal probability. Supervised classifiers (GBDT, neural nets) can also be trained on historical data to predict whether a given transaction is at high risk of

“timeout fraud” (features might include network latency, terminal health, transaction amount vs typical size, etc.).

Hard rules: Impose strict timeout and retry limits. For example, **timeout thresholds** should be set conservatively; if exceeded, the system may require an alternate authentication rather than auto-reversing. ATM operators explicitly recommend strict time limits on transactions [13]. Limit the number of consecutive or rapid reversals per account or POS. Implement a real-time check that if a transaction reversal is issued, flag the customer’s account for review (this can be a business rule). The “Last Transaction Status” check (LTS) must be invoked immediately; if no response, the system should abort rather than presuming success. In summary, require an explicit “credit success” signal before releasing cash, and allow only one active transaction per card at a time.

Settlement and Pooling Anomalies

In Off-us flows, banks credit/debit intermediary “pooling” accounts during settlement. For example, after a successful AEPS Off-us withdrawal, the system posts a debit to the issuer customer and credit to the issuer pool account [6]. Corresponding entries are then made to the acquirer’s pool accounts. **Fraud surfaces:** Fraudsters may intercept or alter these settlement postings. For instance, a malicious agent could fail to pass the pooling credit to NPCI, causing NPCI to under-credit the acquirer bank’s account. Over time, this creates a settlement discrepancy. Additionally, **round-trip laundering** could happen if an attacker engineers settlements that return money to their own accounts via pooling exploits.

Models: Compare settlements to transaction logs via anomaly detection. For each day, the sum of all debit legs (transactions) should equal the sum of credits into the pooling account (minus fees). A change-detection algorithm (e.g. control charts or CUSUM) can flag days when these totals diverge beyond a small tolerance. Unsupervised clustering on the ratio of settlement to transaction volumes across accounts can find outliers (e.g. one bank consistently settling “short”). Graph models (GNNs) can represent the network of accounts and flows; anomalies in flows between nodes (e.g. missing edge weight) can signal tampering.

Hard rules: Mandate **daily reconciliation** invariants. For example: *TotalDebits* + *IncomingTransfers* = *TotalCredits* + *OutgoingTransfers* for each participant and NPCI pool. Automate liquidity thresholds on pooling accounts – if an acquirer’s pool account lacks expected inflows, trigger an investigation. Ensure that every settlement message carries a unique reference; if a settlement response is delayed, use alerting rather than retry indefinitely. Some networks use *dual-control* settlement: two independent systems must confirm the same settlement amount before posting. In any case, pooling interactions should be audited by invariant checks (e.g., sum of all intra-clearing credits equals sum of debits) – any discrepancy beyond rounding errors is a red flag.

Coordinated & Novel Attack Patterns

Fraudsters often act in collusion. For instance, a ring of accounts (mules) may funnel money through multiple transactions so that individual transactions appear normal. **Surface:** patterns like many small transactions between the same set of accounts, or sudden changes in money flow topology.

Graph and sequence models: Use network analysis to detect collusion. Construct a graph where nodes are accounts or terminals and edges are transactions. Graph neural networks or community detection

algorithms can find anomalous clusters of tightly-interacting accounts (mule rings) [14]. Temporal models (LSTM/CNN on time series) can detect sudden spikes in transaction velocity for a user or terminal. Even supervised "social network" features (number of unique counterparties, average transaction peer risk score) can feed into an ML classifier for coordinated fraud.

Hard rules: Limit new accounts or IBANs until KYC is satisfied (e.g. a rule preventing transfers from a new beneficiary until 24h after initial KYC verification). Flag money flows that form cycles (e.g. A→B→C→A) within a short period. Enforce *velocity rules* – e.g. a user cannot debit >X times within a given window or above a certain sum without extra checks. Implement "mule account" screening: if an account receives and then quickly redisburses funds frequently, it should trigger an investigation.

Layered Fraud Detection Architecture

A robust system combines **fast rule checks** with **ML models** in real-time, plus deeper offline analytics. **Real-time layer:** As transactions arrive at the gateway, a rule engine should immediately apply known invariants (e.g. validate sequence order, check blacklists/IP/device) and a pre-trained ML model can score risk in milliseconds. For example, a streaming anomaly detector (e.g. a lightweight one-class or tree model served via a low-latency engine like TensorFlow Serving) can flag each transaction before it posts. Gateway rules might include "reject if biometric failed" or "time-of-day check". **Machine learning in real-time:** use fast models like gradient boosting (XGBoost/LightGBM) or pre-compiled neural nets that can run in a few milliseconds. These models take transaction features (amount, terminal ID, prior history features) and output a fraud score. Combining rules and ML in an ensemble is common – rules catch known threats with certainty, ML catches subtler patterns [15] [16].

Offline/Batch layer: In parallel, all transaction data feeds a data warehouse or lake. Here more computationally intensive analytics run on longer time-scales: for example, computing behavioral profiles, training sequence/graph models, mining for new patterns of fraud, and reconciling settlement data. Periodic batch jobs (hourly/daily) can re-score accounts for risky behavior (aggregate spending changes, drift in device usage) and retrain models on the latest labeled data. This offline system also monitors concept drift: if model performance degrades, it triggers retraining or alerting.

Layered design example: As recommended by modern fraud architectures [16] [17], use three layers – (1) a fast **rule-based layer** (blacklists, velocity limits, IP/device checks), (2) an **anomaly-detection layer** (e.g. unsupervised statistical models on transaction features) and (3) an advanced **predictive ML layer** (e.g. XGBoost) trained on historical fraud labels. Each layer filters out a portion of traffic. In practice, a transaction might first be weeded out by simple rules, then scored by an anomaly detector, then given a final score by a trained classifier [16] [17]. This multi-layer approach reduces false positives and ensures different fraud types are caught (known vs novel) [16] [15].

Detecting Novel and Evolving Patterns

Static ML models cannot see truly new fraud tactics. To catch **novel/mimic frauds**, use unsupervised and self-supervised methods. **Behavior modeling:** Build a profile of each user's "normal" behavior (e.g. typical transaction amounts, geolocations, device IDs, timing patterns). Autoencoders or sequence models trained on historical normal transactions can spot deviations: if a transaction does not fit the autoencoder's reconstruction, it's flagged. For example, [41] describes an ensemble of an autoencoder and variational

autoencoder that learns the normal transaction manifold, raising alarms on subtle anomalies [14] [18]. **Deviation detection:** Clustering algorithms can group similar transactions; any point far from clusters is suspect. **One-class classification:** Train one-class SVMs or isolation forests on bona fide data to detect outliers. **Graph-based anomalies:** New fraud rings often show as unusual subgraphs; algorithms like Deep Graph Infomax or subgraph autoencoders can highlight these.

Self-supervised learning: Pre-train representations on all transactions (e.g. contrastive learning on graphs of transactions or users) so that the model understands normal patterns. Recent research suggests combining *predictive* and *anomaly* learning – e.g. predict the next transaction and flag when the prediction error is high. Additionally, using generative models (GANs) to simulate fraudulent examples can augment detection of new fraud variants.

When a truly unseen fraud occurs, the system should fall back on general signals: unusually high transaction velocity, new device/IP for an account, or contradictions across data sources (e.g. an Aadhaar auth succeeded but the face recognition failed). A carefully-tuned unsupervised detector (e.g. Random Cut Forest on feature embeddings [19]) is invaluable for flagging these patterns that supervised models have never seen.

Training Strategies for Imbalance and Drift

Class imbalance: Fraud is rare (often <1% of transactions), so training must address this. Common approaches include data-level and algorithm-level methods. At the data level, techniques like **SMOTE/ADASYN** (oversampling fraud cases) or **undersampling** the majority class can rebalance the training set. For example, experiments have shown that small, well-chosen undersamples can yield high F1 scores, and tree models like LightGBM perform well on such imbalanced banking data [20]. One can also use ensemble sampling (e.g. bagging on different balanced folds) to stabilize results [21]. At the algorithm level, use **class-weighting** or specialized losses (e.g. focal loss) so the model penalizes missing a fraud more than a normal case. Often a combination of slight oversampling plus cost-sensitive learning works best. In many cases, a hybrid approach (e.g. SMOTE + boosting) improves recall at the expense of precision [21], so thresholds must be tuned to business needs.

Open-source libraries: **imbalanced-learn** (Python) provides SMOTE, ADASYN, cluster-based resamplers; **scikit-learn** and **XGBoost/LightGBM** support class weights and efficient training. The **PyOD** library contains many anomaly-detection models for one-class tasks. For streaming models that adapt to new data, libraries like **River** (for online learning with drift detection) can be used. Always evaluate with precision-recall metrics, since accuracy is misleading in imbalanced settings.

Fraud drift: Patterns of fraud change as attackers adapt. Monitor model performance (e.g. fraud detection rate) over time, and use drift-detection algorithms (like ADWIN) on feature distributions. Periodically retrain models on the latest weeks or months of labeled data. Use rolling windows or ensemble time decay (give more weight to recent examples). Consider online learning: lightweight models can be updated continuously with new labels. Also, use human-in-the-loop feedback: when fraud is found by investigators, feed it back for supervised retraining. Open-source options include **scikit-multiflow** or **River** for concept drift, and platforms like **TensorFlow Extended (TFX)** or **PyTorch Lightning** for streamlined retraining pipelines.

Trade-offs: Performance, Explainability, Complexity, Latency

Designing fraud detection involves balancing accuracy vs speed vs transparency. **Latency:** Real-time detection (in the gateway) demands extremely fast inference. Complex deep models or large graph algorithms may be too slow for per-transaction scoring; such models may be confined to offline analysis. In real-time, lightweight models (e.g. shallow trees, logistic regression) or precomputed rule checks are safer. It is common to use a fast ML model for an initial risk score, and reserve deeper checks for flagged cases or offline review.

Explainability: Regulators and operations teams require reasons for flags. Simple models or rule-based decisions are fully explainable ("Triggered rule X because IP not seen before"). Gradient boosting trees can offer moderate explainability via feature importance or SHAP values (as noted by practitioners in payments) [22]. Deep learning (RNNs, GNNs) often work as black boxes; if used, they should be augmented with explainability tools (LIME/SHAP) and human review. The EU AI Act and other regulations push for transparency. In practice, a high-risk model might use a black-box under the hood but always output an interpretable reason code for the customer/regulator. For example, "blocked because transaction amount is 5× normal and location is unusual," possibly derived via SHAP-annotated model outputs [22].

Complexity vs coverage: A highly complex system (many layers of ML and rules) can catch more patterns but is harder to maintain and slower. A simpler rule engine is fast and transparent but can miss novel attacks. Often a **tiered approach** is used: simple rules fire first to block blatant fraud, while sophisticated ML models (possibly in batch) hunt subtler cases. This multi-tier design (rule → ML → human review) is a practical compromise [16] [15].

Performance vs false positives: Aggressive models catch more fraud but also flag more legit transactions, harming customer experience. This trade-off must be tuned via threshold selection and feedback loops. Some systems use risk scores, only automatically blocking extremely high-risk events and sending medium-risk cases to analyst review. Others dynamically adjust sensitivity by user profile (e.g. known high-value customers get "looser" screening unless very high risk).

Deployment needs: Depending on the scale and risk appetite, options range from lightweight solutions (rule-based with occasional ML scoring) to full enterprise ML pipelines. For maximum throughput (millions of transactions/day), distributed stream processing frameworks (Apache Kafka + Flink/Spark Streaming) and model-serving layers (Seldon, TensorFlow Serving) may be used. For smaller deployments, batch predictions (e.g. nightly scorecards) may suffice. Importantly, any model requiring seconds of computation (like a GNN) will likely run offline, not in the live path.

In summary, we recommend a **hybrid architecture**: simple rules and fast models in the hot path (maximizing throughput and explainability) combined with advanced anomaly detection and profiling offline. This layered defense (as in recent industry blogs and case studies [16] [14]) offers the best balance of real-time protection and strategic learning of new fraud trends. All high-risk decisions should include audit trails and explainable features (SHAP/LIME) to satisfy compliance demands [22].

Sources: The above strategy draws on technical literature and industry best practices. NPCI's own process flows (e.g. for AEPS and RuPay) outline critical checks and reversal logic [1] [4] [5]. Recent analyses of payment fraud emphasize ML's role in adapting to evolving attacks [15] [16] and the need for layered

defenses combining rules and models [16] [14]. Practical experiments on imbalanced financial data highlight the effectiveness of boosting and sampling strategies [20] [21]. Finally, studies of explainable AI stress the importance of transparency (using SHAP/LIME) and governance when deploying complex models in financial systems [22] [11].

---

[1] [2] [4] [5] [6] [7] [10] Gateway_transaction_processflow_new.pdf
file://file_0000000080e07209ae16b4c9071e786e

[3] microsave.net
https://www.microsave.net/wp-content/uploads/2025/01/241212_Frauds_the-Achilles-heel-of-AePS-transactions.pdf

[8] [9] [13] Unraveling ATM Transaction Reversal Fraud - INETCO
https://www.inetco.com/blog/atm-transaction-reversal-fraud/

[11] [12] How India Is Using Cutting-Edge AI to Tackle Payment Frauds
https://www.analyticsvidhya.com/blog/2023/08/how-india-is-using-cutting-edge-ai-to-tackle-payment-frauds/

[14] [18] Building a Three-Model, Real-Time Fraud Detection System with FastAPI | by Nafisa Lawal Idris | Medium
https://medium.com/@nafisaidris413/building-a-three-model-real-time-fraud-detection-system-with-fastapi-d5b5c2de5544

[15] Fraud Detection with ML: Practical Guide for 2025 | Designed to Scale
https://medium.com/designed-to-scale/fraud-detection-machine-learning-practical-playbook-d6b74a3ac8fd

[16] [17] [19] Fraud Detection at Scale with CockroachDB & AWS AI
https://www.cockroachlabs.com/blog/fraud-detection-at-scale/

[20] [21] GitHub - fraperez/fraud-detection-imbalance-strategies: Comparing imbalance strategies for fraud detection
https://github.com/fraperez/fraud-detection-imbalance-strategies

[22] Opening the black box: How explainable AI is transforming fraud detection in payments | The Payments Association
https://thepaymentsassociation.org/article/opening-the-black-box-how-explainable-ai-is-transforming-fraud-detection-in-payments/