FIRST PROBLEM STATEMENT (Cryptography Domain)

Create a secure communication system that allows a bank and its customers to exchange messages over an end-to-end encrypted channel. The system must guarantee confidentiality, integrity, and authenticity of messages.

1. Description

Banks and financial institutions handle highly sensitive information. Ensuring secure communication between a bank and its customers is critical to preventing data breaches, identity theft, and financial fraud. This system must establish an end-to-end encrypted (E2EE) communication channel where only the bank and the customer can read the content — no intermediaries, including the server, should have access to the plaintext.

This project involves designing and optionally implementing a secure messaging protocol or system architecture that supports:

- Encryption (for confidentiality)
- Message signing (for authenticity)
- Message integrity verification

2. Objective

Design and/or implement a secure communication system between a bank and its customers that meets the following security requirements:

- **Confidentiality:** Messages are encrypted end-to-end using strong cryptographic algorithms.
- **Integrity:** Messages cannot be modified undetected.
- **Authenticity:** The identity of the sender is verified through cryptographic signatures.
- **Replay Protection:** Defend against replay attacks using nonces or timestamps.

3. Expected Outcome

- A **design document or architecture diagram** explaining the full communication protocol and key management strategy.
- A **working proof-of-concept (PoC)** implementation (CLI or Web-based).

- Explanation of how:
  - Keys are generated and managed
  - Encryption and signing work
  - Message tampering or impersonation is detected

4. Submission Guidelines

You should submit the following:

1. Documentation (PDF or Markdown):
   - System Architecture
   - Cryptographic Protocol Design
   - How security goals are achieved
   - Limitations and future work
2. Source Code:
   - Secure PoC implementation in Python, Java, Go, or any secure language
   - Code must follow secure coding best practices
3. README File:
   - How to run the code
   - Dependencies and tools used
   - Example message exchange walkthrough
4. Submission Format:
   - GitHub repository link

5. Evaluation Metrics

Criteria	**Marks (POINTS)**
Protocol Design & Cryptography Use	20
Technical Design & Quality	20
Security & Privacy	20
Threat Model & Mitigation	10
Innovation & Impact	10
Progress During Hackathon	5
Presentation & Communication	5

SECOND PROBLEM STATEMENT (AI Domain)

Build a Conversational AI System for Handling Dynamic Banking Interactions like Loan Applications, Card Blocking, and Account Queries with Multi-Turn Understanding and Real-Time Task Execution.

1. Description

Modern banking users expect intelligent, real-time conversational assistants that can perform a variety of tasks, understand context across multiple messages, and adapt flexibly to changing user inputs. This project focuses on designing and optionally implementing a conversational AI agent that can handle goal-oriented flows in banking such as:

- Applying for loans
- Blocking a lost or stolen debit/credit card
- Requesting account statements or transaction summaries
- Asking about balances, charges, or interest rates

The assistant must:

- Manage multi-turn conversations where the user may provide information gradually
- Understand evolving user intent as the conversation progresses
- Seamlessly handle context switches (e.g., from “apply for a loan” to “block my card”)
- Deal with ambiguous or incomplete queries and respond with clarifying questions
- Interact with external systems like banking APIs, knowledge bases, and databases.

This solution is expected to mimic the behavior of a human banking assistant — intelligent, helpful, and task-focused — capable of retrieving information, invoking tools, and making decisions throughout the conversation.

2. Objective

Design and/or implement a Conversational AI system that:

- Supports goal-driven, multi-step interactions for core banking workflows
- Maintains context and adapts to user behavior (e.g., topic shifts, clarifications).

- Pulls info from external systems or databases when needed.
- Manages interruptions and ambiguity, guiding users to complete tasks.
- Responds in real time, performing backend actions or fetching personalized data.

The system should be modular, extensible, and support a wide range of use cases with low-latency interaction loops.

3. Expected Outcome

- A design/architecture document illustrating:
    - The conversational flow design
    - Context/state management strategy
    - Decision-making or routing logic for completing banking tasks
- A working PoC that includes:
    - A conversational interface (CLI, Web, or messaging platform)
    - 2–3 end-to-end use cases (e.g., loan application, block card, mini statement)
    - Integration with mock APIs for operations (e.g., submitting forms, fetching data)
- Clear explanation of:
    - How task delegation or external tool access is handled during conversation
    - How the assistant tracks user progress across multiple steps
    - How fallback and clarification logic works in case of uncertainty .

4. Submission Guidelines

You should submit the following:

Documentation (PDF or Markdown):

- System Architecture & Component Breakdown
- Agent Behavior or Flow Design (including fallback and context handling)
- Limitations and Potential Enhancements

Source Code:

- PoC implementation in a modular, extensible framework
- Core components like dynamic responses and API calls.
- Clear separation between dialogue management, tool interaction, and business logic.

README File:

- How to set up and run the assistant
- Supported flows with example prompts
- Dependency list and usage guide

Submission Format:

- GitHub Repository Link with complete project structure

5. Evaluation Metrics

Criteria	Marks (POINTS)
Problem Understanding & Solution Design	15
Dialogue Flow Design & External Tool/API Interaction Capability	15
Code quality, Modularity & Clarity	15
Context Handling & Realism	15
Innovation & Value-Add	15
User Experience / Usability	15
Progress During Hackathon	5
Presentation & Communication	5

Third PROBLEM STATEMENT (ML & Data Science Domain)

AI-POWERED FRAUD DETECTION SYSTEM

Build an AI-Powered Fraud Detection System capable of identifying fraudulent transactions in real-time, leveraging innovative machine learning approaches to minimize false positives and maximize the detection of actual fraud.

1. Description

Design and implement an AI-powered fraud detection model that operates in real time, leveraging historical transaction data to discover and flag anomalies. Conventional or rule-based methods are not permitted.

The emphasis is on creativity and innovation—solutions that demonstrate novel, effective approaches to fraud detection are highly encouraged.

2. Objective

Successfully design, implement, and deploy a real-time fraud detection model that accurately identifies fraudulent transactions while minimizing false positives. The solution should demonstrate originality and effectiveness, prioritizing innovative approaches over conventional methodologies.

3. Expected Outcome

QUANTITATIVE (70 POINTS):

- **Precision (30 POINTS):**
  Measures how many of the transactions flagged as fraud are actually fraudulent. It's crucial to minimize false positives.

- **Recall (Sensitivity) (30 POINTS):**
  Indicates how well the model detects actual fraudulent transactions. This metric ensures that most fraud cases are caught.

- **F1 Score (5 POINTS):**
  Balances precision and recall, providing a single metric to evaluate the model's performance.

- **AUC-ROC (5 POINTS):**
  Measures the trade-off between true positive rate and false positive rate. A higher AUC indicates better model performance.

QUALITATIVE (30 POINTS):

- **Feature Importance (5 POINTS):**
  Teams should analyze which features contribute most to the model's predictions.

- **Insights from Data (10 POINTS):**
  Participants should provide a detailed analysis of the dataset, highlighting interesting patterns or insights that informed their model development.

- **Progress During Hackathon (5 POINTS):**
  Evaluate the progress made by the team throughout the hackathon.

- **Innovation & Documentation (10 POINTS):**
  Assess the quality and effectiveness of the presentation.

BONUS POINTS (10 POINTS):

- **Adversarial Robustness:**
  How well does the model hold up against deliberately manipulated data?

- **Model Size / Efficiency:**
  Is the model lightweight enough for real-time deployment?

- **Explainability Score:**
  How interpretable are predictions for real-world operators?

4. Submission Guidelines

Jupyter notebook with the code and the prediction csv file.

5. Evaluation Metrics

Criteria	Marks (POINTS)
Precision	30
Recall (Sensitivity)	30
F1 Score	5
AUC-ROC	5
Feature Importance	5
Insights from Data	10
Progress During Hackathon	5
Documentation, Presentation & Communication	10

Fourth PROBLEM STATEMENT (Java / Web / API Engineering Domain)

API-DRIVEN FINANCIAL DATA AGGREGATOR

Develop a robust, API-driven financial data aggregator with a real-time, role-based dashboard for accessing and analyzing diverse financial data, incorporating advanced engineering features for security, performance, and automated deployment.

1. Description

Develop an API that aggregates and normalizes financial data from various sources, offering a unified interface (dashboard) for accessing, analyzing, and managing this information. The system should incorporate advanced engineering features for performance, security, and scalability

2. Objective

Create a robust API that simplifies access to diverse financial data, enabling developers to build comprehensive financial applications. The solution should include:

- A real-time configurable dashboard that visualizes the data through multiple graphs.
- Role-based filtering using JWT (admin, client, and user segregation).
- Proper data segregation and security protocols.
- Additional backend layers for data performance analysis and scalability.

3. Expected Outcome

- An API or set of APIs capable of fetching and normalizing financial data from provided or simulated datasets.

- A secure, user-role-based dashboard utilizing these APIs to display dynamic graphs with features like pagination, rate limiting, and performance tracking.

4. Submission Guidelines

-GitHub repository link with complete source code
-Screenshots/videos of the dashboard or a live deployment link (e.g., Vercel or Netlify)
-API documentation (Swagger/Postman)
-Unit testing reports
- RDBMS design schema and ER diagrams

5. Evaluation Metrics

Criteria	Marks (POINTS)
User-Friendliness	40
Performance (Latency, DB Query Efficiency)	10
Authentication and Secure API	10
Database	20
Deployment	5
Innovation & Documentation	5
Progress During Hackathon	5
Presentation & Communication	5