**Introduction**

**Problem Statement**:

* Our new POS terminals lack built-in receipt printing capability.
* This absence could lead to complaints from both merchants and customers.

**Proposed Solution**:

* Develop a **receipt delivery service** to securely share receipts from POS terminals to customers' phones via a **QR code**.
* accessible in real time.

**System Architecture**

* **Android App**:
  + Installed on the POS- devices
  + Captures receipt and securely pushes it to the server.
  + Then retrieves a QR link from the server for customer access.
* **APK Server**:
  + Manages API endpoints and handles internal communications.
  + Receives and securely stores receipt data; generates QR link for the Android app.
  + Administers and organizes receipt data for internal processing.
* **Web Server (DMZ)**:
  + Hosted within the DMZ.
  + Available to the public
  + Manages customer requests for QR code links.

All the nodes are communicated via API

**Security Layers and Controls**

1. **Resource Isolation**:
   * Public and Private Endpoints are **fully isolated**
   * Hosted on **separate servers**, DMZ for public access and APK server for private handling.
   * This separation minimizes exposure by ensuring no public access to sensitive internal data.
2. **Directory Traversal Prevention**:
   * **Only single endpoint** is available to the public **invoice.bankofabyssina.com/<receipt\_id:UUID4>**
   * **UUID4 for Receipt IDs**: Non-sequential, unique identifiers prevent unauthorized access or enumeration.
3. **Network Layer Security**:
   * **IP Whitelisting**:
   * A dedicated IP range is reserved exclusively for POS terminals, **no other devices** can obtain an address within this range.
   * This setup restricts the APK server to communicate only with requests originating from these authorized IPs, enhancing security and minimizing unauthorized access.
4. **Application Layer Security**:
   * **API Key Validation**: only requests with valid API keys are processed,
   * reducing the risk of unauthorized access.
5. **Data Encryption**:
   * All data transmitted through every API endpoint, including both **requests and responses** (POST, GET), is **encrypted** alongside an API key for added security.
   * Used encryption algorithm: AES(**Advanced Encryption Standard**)-128bit

**Why AES-128 over AES-256**:

* **Server Performance**: AES-256 requires more processing power, which would slow down decryption/encryption for the high volume of requests.
* **Device Limitations**: POS terminals have limited resources (1GB RAM), and AES-256 would consume significant resources, impacting performance.
* **Battery Efficiency**: Encrypting with AES-256 would drain POS device batteries faster, posing issues for merchants who may not consistently recharge devices.

1. **Web Application Firewall (WAF) Integration**

* All public-facing resources are secured through a **WAF**.
* Monitors traffic in real time to detect and block malicious requests,
* enhancing system resilience against attacks.