#### **What is PCK60R?**

PCK60R is a service in the Infinity Connect system designed to manage and send different types of customer notifications such as emails, SMS, and letters. These notifications are initiated from the PaymentOne Cards (P1C) system and are processed by Infinity Connect to be sent to clients.

#### **How It Works:**

1. **Message Transmission**:
   * P1C sends a message to Infinity Connect through an MQ (Message Queue).
   * Infinity Connect processes this message and converts it into the required format (XML/JSON) depending on the type of notification (Email, SMS, Letter).
   * The converted message is then sent to the client’s endpoint through a Service API.
2. **Types of Notifications**:
   * **PCK60REM**: Handles email notifications.
   * **PCK60RSM**: Manages SMS notifications.
   * **PCK60RLT**: Deals with letter notifications.
3. **Processing Flow**:
   * Each notification type (Email, SMS, Letter) is associated with a specific service in Infinity Connect.
   * These services convert the MQ message into the appropriate format and ensure that it is sent to the correct client endpoint.

#### **Technical Details:**

1. **Infinity Connect Services**:
   * **PCK60REM Service API**: Processes and sends email notifications.
   * **PCK60RSM Service API**: Handles SMS notifications.
   * **PCK60RLT Service API**: Manages letter notifications.
2. **Java Classes and Packages**:
   * **PCK60RJsonHelper\_v01\_01.java**: Helps prepare the JSON request.
   * **PCK60RXMLHelper\_v01\_01.java**: Assists in preparing XML requests.
   * **PCK60RXMLMessageAdapter.java**: Adapts and translates XML messages.
   * **PCK60RJSONMessageAdapter.java**: Adapts and translates JSON messages.
   * **Authenticator Classes**: These ensure that the notifications (Email, SMS, Letter) are sent securely.
3. **SQL Configuration**:
   * SQL scripts are used to configure backend endpoints, JMS (Java Message Service) endpoints, and service definitions to manage how and where the messages are sent.
   * These scripts ensure that the system is set up correctly to handle and route the notifications.

### **Integration Overview:**

1. **P1C (PaymentOne Cards) System**:
   * **Role**: P1C is the originator of the notifications. Whenever there is a need to notify a customer (like a transaction alert, account update, etc.), P1C generates the notification message.
   * **Integration**: P1C sends these messages to the Infinity Connect system using a messaging protocol known as **MQ** (Message Queue). The message is essentially a request to notify a customer through email, SMS, or letter.
2. **Infinity Connect System**:
   * **Role**: Infinity Connect acts as the middleware or interface that handles these notification requests.
   * **Integration with P1C**: When P1C sends a message via MQ, Infinity Connect receives it, processes it, and converts the message into the appropriate format (XML/JSON) based on the type of notification.
   * **Integration with Clients**: After processing the message, Infinity Connect sends it to the client’s system using a **Service API**. This API ensures that the message is correctly formatted and sent to the right place (e.g., an email service, SMS gateway, or print service for letters).
3. **MPMS (Managed Payment System)**:
   * **Role**: MPMS oversees and manages the entire payment process, including notifications.
   * **Integration with P1C and Infinity Connect**: MPMS is likely responsible for the broader management of the P1C system, ensuring that all transactions and related notifications are handled correctly. It may also monitor the flow of notifications from P1C through Infinity Connect to the end customer.

### **Detailed Flow of Integration:**

1. **Notification Event in P1C**:
   * An event occurs in P1C that triggers a notification (e.g., a customer makes a payment).
   * P1C generates a notification message and sends it via MQ to Infinity Connect.
2. **Processing in Infinity Connect**:
   * **Receiving the Message**: Infinity Connect receives the MQ message from P1C.
   * **Message Conversion**: Infinity Connect then converts this message into a customer-friendly format (XML/JSON).
     + **PCK60REM** handles emails.
     + **PCK60RSM** handles SMS.
     + **PCK60RLT** handles letters.
   * **Sending to Client**: After converting the message, Infinity Connect sends it to the client’s endpoint via the appropriate Service API.
3. **Final Delivery**:
   * The client’s system receives the notification and delivers it to the end customer via email, SMS, or letter.
4. **Monitoring and Management via MPMS**:
   * MPMS may monitor this entire process, ensuring that the notification was sent successfully and managing any issues that might arise in the notification flow.

### **Summary:**

* **P1C** generates the notification.
* **Infinity Connect** processes and converts the notification and sends it to the client.
* **MPMS** oversees and manages the entire notification process, ensuring smooth operation between P1C and Infinity Connect.

#### **Overview of RSA Integration**

* **RSA (Rivest-Shamir-Adleman)**: RSA is a cryptosystem used for secure data transmission. In this context, RSA is employed as part of a multi-factor authentication (MFA) system. It enhances security by ensuring that users provide multiple forms of verification before accessing a service, particularly for financial transactions.

#### **2. Purpose of RSA in This System**

* **Enhanced Security**: The RSA system adds an extra layer of security by requiring users to verify their identity through different methods, reducing the likelihood of unauthorized access during sensitive operations like financial transactions.

#### **3. Workflow of RSA Integration with FIS and P1C**

* **Step 1: Transaction Initiation**
  + When a customer initiates a transaction, such as making a purchase on an eCommerce site, RSA determines whether the transaction requires additional identity verification. This decision is based on pre-set security policies or transaction characteristics (like the amount or type of purchase).
* **Step 2: Fetching Verification Methods**
  + RSA sends a request to FIS using the FetchAvailableAliases API. This API call retrieves a list of possible verification methods (aliases) available for the customer. These aliases might include options like SMS (where a one-time password is sent to the customer’s phone) or Out-of-Band (OOB) methods (like a push notification to a mobile app).
* **Step 3: Customer Chooses Verification Method**
  + The customer selects their preferred verification method:
    - **SMS Verification**: If SMS is selected, RSA sends an OTP (one-time password) directly to the customer’s mobile number. The customer enters this OTP to verify their identity.
    - **Out-of-Band (OOB) Verification**: If OOB is chosen, the verification involves a mobile application. Here, RSA and FIS engage in a more complex interaction.
* **Step 4: Out-of-Band (OOB) Process**
  + **Initiating OOB**: RSA sends the InitiateOOB API request to FIS to start the OOB process. FIS then sets the transaction status to “Pending” in its internal database.
  + **Client Verification**: FIS communicates with the client application (the mobile app) to initiate the customer verification process. This involves sending an InitiateClientOOB API request to the client.
  + **Status Checks**: RSA continuously checks the status of the OOB verification by sending CheckOOBStatus API requests to FIS. These checks happen every few seconds until the verification process is completed.
  + **Updating Status**: Once the client’s mobile app completes the verification (e.g., the customer approves the transaction via the app), the client sends an UpdateOOBStatus API call to FIS. FIS updates the status in its database.
  + **Final Status**: FIS responds to RSA’s status checks with the updated status. If the status is no longer “Pending” (e.g., “Approved” or “Rejected”), the transaction can proceed accordingly.

#### **4. Challenges and Improvements in the Process**

* **Initial Challenges**: The original solution involved FIS calling the client for every status update, which led to potential timeouts and failed authorizations. This was problematic, especially when the response times exceeded the Service Level Agreement (SLA) of 3 seconds.
* **Upgraded Solution**: The process was improved by reducing the number of direct calls between FIS and the client. Instead of repeatedly querying the client, FIS now updates the status in its internal database, which RSA accesses. This reduces the risk of timeouts and ensures smoother, more reliable transaction processing.

#### **5. Connection to P1C**

* **Integration with P1C**: P1C (which appears to be a backend or database system) now plays a crucial role in handling these secure authentication requests. New services and APIs were created within P1C to:
  + Handle inbound OOB requests.
  + Manage status updates for these requests.
  + Provide a user interface for browsing and maintaining transaction statuses.
* **Internal Database**: P1C uses an internal database to store the statuses of these OOB requests. This database is continually updated and queried to ensure that the most recent status is available to both RSA and the customer.

#### **6. Benefits of This System**

* **Improved Security**: By leveraging RSA and multi-factor authentication, the system significantly reduces the chances of unauthorized transactions.
* **Efficiency**: The upgraded process minimizes delays and errors, ensuring that legitimate transactions are processed quickly while maintaining high security.
* **Scalability**: The integration with P1C ensures that the system can handle a large number of transactions without performance degradation.

### **Conclusion**

The RSA integration within this system is designed to provide a robust, secure method for verifying customer identities during transactions. By interacting with FIS and P1C through a series of API calls, the system ensures that only authorized transactions are processed, thus enhancing the overall security and reliability of the service.