**1. EMV evidence: please attach (or redact) a snippet of an EMV Level-3 test log you produced (field 55, DE 39, cryptogram) from any past project.**

Field 55 is BER-TLV encoded. Here are a few of the relevant EMV tags and values from a past test case (with a few values redacted)

BER-TLV-encoded string:

9F1A0208409F3901059F360200029F370441C2386E5F3401014F07A00000015230109F3403
5E03009F3501229F10080105000000000009F3303E0F8C89F0E0500000000005F2A020840
9F0F05000000000050084449534F5645529F0D05000000000095050080008800571[redacted]
[redacted]9F4C08E21037F0B5750EEF9F2701809A032504229
F0607A00000015230109B02E8009F03060000000000009C01015F24034912319F2608828AC
0BF9346404C5A08[redacted]9F0702FFC09F080200029F40050300C000009F02060
000000195009F2103122436820238009F1E083030303030393035

Here is that TLV string parsed into individual tags and values:

| | |
|---|---|
| 9F1A Terminal Country Code | 0840 |
| 9F39 POS Entry Mode | 05 |
| 9F36 Application Transaction Counter (ATC) | 0002 |
| 9F37 Unpredictable Number (UN) | 41C2386E |
| 5F34 PAN Sequence Number (PSN) | 01 |
| 4F Application Identifier (ADF Name) | A0000001523010 |
| 9F34 CVM Results | 5E0300 |
| 9F35 Terminal Type | 22 |
| 9F10 Issuer Application Data (IAD) | 0105000000000000 |
| 9F33 Terminal Capabilities | E0F8C8 |
| 9F0E Issuer Action Code - Denial | 0000000000 |
| 5F2A Transaction Currency Code | 0840 |

| | |
|---|---|
| 9F0F Issuer Action Code - Online | 0000000000 |
| 50 Application Label | 444953434F564552 (DISCOVER) |
| 9F0D Issuer Action Code - Default | 0000000000 |
| 95 Terminal Verification Results (TVR) | 0080008800 |
| 57 Track 2 Equivalent Data | [redacted] |
| 9F4C ICC Dynamic Number | E21037F0B5750EEF |
| 9F27 Cryptogram Information Data (CID) | 80 |
| 9A Transaction Date (YYMMDD) | 250422 |
| 9F06 AID (Terminal) | A0000001523010 |
| 9B Transaction Status Information (TSI) | E800 |
| 9F03 Amount, Other | 000000000000 |
| 9C Transaction Type | 01 |
| 5F24 Application Expiration Date (YYMMDD) | 491231 |
| 9F26 Application Cryptogram (AC) | 828AC0BF9346404C |
| 5A Primary Account Number (PAN) | [redacted] |
| 9F07 Application Usage Control (AUC) | FFC0 |
| 9F08 Application Version Number | 0002 |
| 9F40 Additional Terminal Capabilities | 0300C00000 |
| 9F02 Amount, Authorised (Numeric) | 000000019500 |
| 9F21 Transaction Time (HHMMSS) | 122436 |
| 82 Application Interchange Profile (AIP) | 3800 |
| 9F1E IFD Serial Number | 3030303030393035 (00000905) |

DE 39 (Response Code):  00 (Approved)

**2. ISO 8583 library: will you use an existing Kotlin/JVM encoder/decoder or build custom? Which one and why?**

Given that we likely want to get an MVP up and running as fast as possible, I would choose to use an existing library instead of implementing ISO 8583 from scratch. Though I am familiar with the ISO 8583 standard, building a custom version will lengthen the development timeline. I would recommend using the jreactive-8583 library. It's an open source solution that provides a lot of important features out of the box:

- Encoding and decoding ISO 8583 messages
- Secure logging (sensitive fields are masked automatically)
- Uses Netty for TCP/IP

The Apache License 2.0 allows for modifications to the source code so if we needed to add other features that would be an option, but writing a custom implementation seems like overkill, at least initially.

**3. MDB timing: on your last vending project how did you handle (a) price broadcast, (b) Vend Approved, (c) Vend Complete, and (d) timeout recovery?**

I haven't worked with MDB directly in the past, but the communication between the application and the VMC is all based on the state of the current vending transaction. At a high level, this is how I would handle each of those operations:

(a) Price Broadcast - The application listens on the bus for price broadcast commands. Once it receives a price broadcast command, the application initializes a transaction with the given price and activates the EMV kernel listening on the contactless interface for a card to be presented.

(b) Vend Approved - Once the application receives an authorization response from the cloud, it sends a Vend Approved message to the VMC to trigger the delivery of the product.

(c) Vend Complete - The application listens on the MDB for a Vend Complete. Once received, the application will be ready to initiate a new transaction (reset screen to initial state, begin listening again for a price broadcast)

(d) Timeout recovery - there are a few timeout situations I can think of

(i) The application doesn't respond to an MDB command within the allotted time window

(ii) A customer doesn't tap their card within an acceptable timeframe

(iii) A network call (authorization request) times out.

Timeout recovery logic is some of the most complex to write. At a high level for each of those 3 scenarios I would create a state machine representing all of the possible timeout/error states, ensuring that the operations that can be recovered are recovered, and those that don't trigger a reset state (restarting the transaction, notifying the customer, etc.)

**4. Timezone & overlap: we're UTC-4 (St Kitts). Which 2-3 h window can you guarantee overlap for stand-ups?**
I'm based in Utah (UTC-6). I can guarantee 12:00 PM - 3:00 PM (UTC-4, St Kitts), which is 10:00 AM - 1:00 PM (UTC-6, Utah) my time. I'm very flexible so I can adjust earlier or later as needed.

**5. Hardware access: comfortable starting with ADB-over-IP? After the trial we'll DHL an IM30 dev kit confirm shipping address.**
Yes, ADB-over-IP will work great initially and if everything goes well, having some hardware in-person would be awesome.

**6. Security controls: outline the steps you'll take to ensure no PAN is ever written to disk or log during development.**
PCI DSS compliance is extremely important for our application, never logging PANs is an important part of that. Here are a few steps I would take:

(a) Use the jreactive-8583 for ISO 8583 messages, as it automatically doesn't log pans
(b) Configure the more general logging framework to never log full PANs or other sensitive fields (cryptogram, etc.). If logging a PAN is required for troubleshooting, I will only log a masked version of the PAN compliant with PCI DSS
(c) Run log messages through a cleaning process which would include using Luhn's algorithm to check numeric fields and ensure they aren't valid PANS.

**7. MVP timeline: once the trial is passed and hardware arrives, how many calendar weeks to a working MVP that (a) completes an online EMV auth against our cloud and (b) triggers MDB Vend Approved?**
Based on my current understanding of the project, I estimate that it would take around 5 weeks to build the MVP.

- 1 week to develop the UI (unknown what that entails as of now)
- 2 weeks to integrate the application with the PAX EMV SDK and correctly configure it
- 1 week to build out th MDB state machine
- 1 week for testing

That estimate may fluctuate some as I get a better grasp of exactly what the project looks like, but I feel based on my past experience that 5 weeks should be sufficient.