

Here is an **in-depth explanation of each key APDU command** involved in an **EMV contact transaction**, covering purpose, structure, flow, and sample dumps:

◆ **1. SELECT (PSE / AID)**

Purpose:

To select either the **Payment System Environment (PSE)** or a specific **Application Identifier (AID)**.

Command Structure:

php-template

CopyEdit

00 A4 04 00 <Lc> <Data> 00

- 00 – CLA (ISO)
- A4 – INS (SELECT)
- 04 – P1 (Select by name)
- 00 – P2 (First or only occurrence)
- <Lc> – Length of AID
- <Data> – AID value
- 00 – Le (expected length)

Example:

text

CopyEdit

00 A4 04 00 07 A0 00 00 00 03 10 10 00

→ Selecting Mastercard AID (A0000000031010)

Response:

6F tag (FCI Template), contains 84 (DF Name), A5 (FCI Proprietary Template)

◆ **2. GET PROCESSING OPTIONS (GPO)**

Purpose:

Initiates the processing of the application after selection.

Command Structure (CDOL1 data):

php-template

CopyEdit

80 A8 00 00 <Lc> 83 <Len> <Data> 00

- 83 tag: Carries PDOL data (constructed using terminal values)

Example:

text

CopyEdit

80 A8 00 00 0B 83 09 00 00 00 00 00 00 00 00 00 00 00

(9 bytes of PDOL input)

Response:

Returns AIP and AFL

text

CopyEdit

77 0E 82 02 38 00 94 08 08 01 01 00 10 01 02 00

- 82 – AIP (Application Interchange Profile)
- 94 – AFL (Application File Locator)

◆ 3. READ RECORD**Purpose:**

Read records defined in AFL to retrieve card data (like PAN, Expiry, etc.)

Command Structure:

php-template

CopyEdit

00 B2 <Record #> <SFI | 04> 00

- SFI is left-shifted by 3 bits

Example:

text

CopyEdit

00 B2 01 14 00

→ Read Record #1, SFI=2 ($2 << 3 = 0x10 | 0x04 = 0x14$)

Response:

Cardholder data in TLV (e.g., 5A – PAN, 5F24 – Expiry, etc.)

◆ **4. GENERATE AC – 1st GENERATION (GAC1)**

Purpose:

Card analyzes terminal risk management, decides if offline approval, decline, or online is needed.

Command Structure:

php-template

CopyEdit

80 AE <CryptogramType> 00 <Lc> <CDOL1 data> 00

- <CryptogramType>:
 - 80 = ARQC (Online)
 - 00 = AAC (Offline Decline)
 - 40 = TC (Offline Approve)

Example:

text

CopyEdit

80 AE 80 00 1A <CDOL1 Data> 00

Response:

text

CopyEdit

77 0E 9F27 01 80 9F36 02 00 01 9F26 08 5A A3...

- 9F27 – Cryptogram Information Data (e.g., 80 for ARQC)
- 9F36 – ATC (Application Transaction Counter)
- 9F26 – Cryptogram (ARQC/TC/AAC)

- 9F10 – IAD (Issuer Application Data)
-

◆ 5. GENERATE AC – 2nd GENERATION (GAC2)

Purpose:

Performed after issuer host response is received for online transactions.

Command Structure:

php-template

CopyEdit

80 AE <CryptogramType> 00 <Lc> <CDOL2 Data> 00

- Usually:
 - 00 – Request AAC (Decline)
 - 40 – Request TC (Approve)

Example:

text

CopyEdit

80 AE 40 00 1B <CDOL2 Data> 00

Response:

Same tags as GAC1 with updated 9F27, cryptogram, ATC, etc.

◆ 6. INTERNAL AUTHENTICATE

Purpose:

Used for offline dynamic data authentication (DDA).

Command Structure:

php-template

CopyEdit

00 88 00 00 <Lc> <Unpredictable Number>

Example:

text

CopyEdit

00 88 00 00 04 12 34 56 78

Response:

Cryptographic data generated by card for the terminal to verify with issuer public key.

◆ **7. EXTERNAL AUTHENTICATE**

Purpose:

Terminal sends response from issuer for verification (e.g., issuer authentication data 91).

Command Structure:

php-template

CopyEdit

00 82 00 00 <Lc> <Issuer Authentication Data>

Example:

text

CopyEdit

00 82 00 00 10 <91 data from issuer>

◆ **8. PUT DATA / GET DATA**

Purpose:

Used for updating or retrieving specific data elements on the card.

GET DATA Command:

php-template

CopyEdit

80 CA <P1> <P2> 00

Example:

Get ATC (Tag 9F36):

text

CopyEdit

80 CA 9F 36 00

Response:

text

CopyEdit

9F36 02 00 5C

ATC = 0x005C

Summary Table

Command	Purpose	Direction	Sample
SELECT	Select AID	Terminal → Card	00 A4 04 00
GPO	Get Processing Options	Terminal → Card	80 A8 00 00
READ RECORD	Read static data	Terminal → Card	00 B2
GAC1	Generate 1st AC	Terminal → Card	80 AE 80 00
GAC2	Generate 2nd AC	Terminal → Card	80 AE 40 00
INT AUTH	Offline DDA	Terminal → Card	00 88
EXT AUTH	Issuer Auth	Terminal → Card	00 82
GET DATA	Get ATC etc.	Terminal → Card	80 CA

If you want:

- Real transaction dump with TVR, TSI, AIP, TTQ
- Sample GAC1/GAC2 data values and how cryptograms look
- Online vs Offline decision flow

Let me know and I can provide next.

You said:

SELECT (PSE / AID) explain in depth what is CLS , INS, P1 and P2 LE, LC

ChatGPT said:

Let's break down the **SELECT APDU command** in depth, especially as it's used during EMV card processing when selecting the **Payment System Environment (PSE)** or **Application Identifier (AID)**.

◆ Overview of SELECT Command (APDU)

The SELECT command is used to select:

- **PSE (1PAY.SYS.DDF01)** for contact cards (optional).
 - **PPSE (2PAY.SYS.DDF01)** for contactless cards.
 - Or **AID** directly, if PSE/PPSE is not supported.
-

◆ APDU Command Structure

java

CopyEdit

CLA | INS | P1 | P2 | Lc | Data field (AID or DF name) | Le

✓ Explanation of Each Byte

Field Meaning	Value (Common)	Explanation
CLA Class of instruction	00	Indicates ISO/IEC 7816 command
INS Instruction	A4	A4 = SELECT command
P1 Selection control	04	Select by DF name (most used)
P2 Selection options	00	Return FCI template
Lc Length of Data field	XX	Length of AID/DF Name
Data AID or PSE (e.g. 1PAY.SYS.DDF01)	-	What to select
Le Expected length of response	00	Get full response

◆ Examples

1. Selecting PPSE (for Contactless)

mathematica

CopyEdit

00 A4 04 00 0E 32 50 41 59 2E 53 59 53 2E 44 44 46 30 31 00

Breakdown:

Byte(s)	Meaning
----------------	----------------

00	CLA (ISO 7816)
----	----------------

A4	INS (SELECT)
----	--------------

04	P1 (select by DF name)
----	------------------------

00	P2 (first or only occurrence)
----	-------------------------------

0E	Lc = 14 bytes
----	---------------

32 50 41... DF name = 2PAY.SYS.DDF01

00	Le (expected length = full)
----	-----------------------------

2. Selecting a Specific AID

Example AID: A0 00 00 00 03 10 10 (Visa)

CopyEdit

00 A4 04 00 07 A0 00 00 00 03 10 10 00

Breakdown:

Byte(s)	Meaning
----------------	----------------

00	CLA
----	-----

A4	INS
----	-----

04	P1
----	----

00	P2
----	----

07	Lc (7 bytes)
----	--------------

A0 00 00 00 03 10 10 AID (Visa Credit)

Byte(s)	Meaning
00	Le

◆ P1 and P2 Variants

P1 Meaning

00 Select EF by file ID

04 Select DF by name (AID or PSE)

P2 Meaning

00 Return FCI (File Control Information)

0C Return FCP (File Control Parameters)

02 Return FCI including FCI Proprietary Template

Standard EMV cards generally use P1 = 04, P2 = 00.

◆ Response to SELECT

Successful SELECT command returns 6F tag (FCI Template) with data inside like:

- 84 (DF Name)
- A5 (FCI Proprietary Template)
- 50 (Application Label)
- 87 (Application Priority Indicator)
- 9F38 (PDOL)

Followed by:

 90 00 – success

 Other SW1/SW2 indicate failure

Summary

Element Description

CLA 00 Standard ISO class

INS A4 SELECT command

P1 04 Select by name

P2 00 Return FCI template

Lc Length of AID or PSE

Data AID or PPSE/PSE string

Le 00 to return full response

Let's break down the **SELECT APDU command** in the context of EMV, especially when selecting the **Payment System Environment (PSE)** or **Application Identifier (AID)**.

◆ General Format of an APDU Command

Field Description

CLA Class byte

INS Instruction byte

P1 Parameter 1

P2 Parameter 2

Lc Length of data

Data Command data

Le Expected length of response

✓ SELECT Command Structure

Goal: Select a file (like PSE or application AID) on the card.

▶ APDU Example: SELECT PSE

To select the Payment System Environment:

mathematica

CopyEdit

00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31

Let's decode this:

Field	Value	Description
CLA	00	Inter-industry class
INS	A4	SELECT instruction
P1	04	Select by name (DF name)
P2	00	First or only occurrence
Lc	0E	Length of following data (14 bytes)
Data	31 50 41 59 2E 53 59 53 2E 44 44 46 30 31	ASCII for 1PAY.SYS.DDF01
Le	—	Not specified explicitly (means maximum response)

Explanation of Each Byte

CLA (Class byte)

- 00 — ISO/IEC 7816-4 standard command (most commonly used).
- Non-zero values are used for proprietary or secure messaging.

INS (Instruction byte)

- A4 — Instruction code for **SELECT**.

P1 (Parameter 1)

Specifies **selection options**:

- 00: Select by file ID
- 04: Select by **name** (DF name or AID)
- Common value for PSE/AID is 04.

P2 (Parameter 2)

Specifies **selection control**:

- 00: First or only occurrence
- 0C: Return **FCI template**
- Most POS terminals use 00 for general selection or 0C if they want the full FCI.

Lc (Length of Command Data)

- Hex 0E = 14 bytes, length of AID or DF name.

Data

- DF name or AID to be selected. E.g., 1PAY.SYS.DDF01 (PSE) or a card-specific AID.

Le (Length Expected)

- Optional. If not specified, the terminal expects the full response. Usually omitted when using SELECT.
-

SELECT AID Example

CopyEdit

00 A4 04 00 07 A0 00 00 00 03 10 10

Field	Value	Description
CLA	00	Standard
INS	A4	SELECT
P1	04	Select by AID
P2	00	First or only occurrence
Lc	07	Length of AID

Data A0 00 00 00 03 10 10 Visa credit AID

Le — Full response expected

This command tells the card:

"Select the Visa AID application and return its File Control Information (FCI)."

Typical FCI Response (example for Visa)

markdown

CopyEdit

6F 23

84 07 A0 00 00 00 03 10 10

A5 18

50 0A 56 49 53 41 20 43 52 45 44 49 54

Explanation:

- 6F: FCI Template
 - 84: DF name (AID)
 - A5: Proprietary template
 - 50: Application label (ASCII: "VISA CREDIT")
-

Errors from SELECT

SW1 SW2 Meaning

6A 82 File not found (invalid AID/PSE)

6A 86 Incorrect P1/P2

67 00 Wrong length

6D 00 Instruction not supported

Summary

Field	Meaning	For SELECT
CLA	Class of command	00 (standard EMV)
INS	Instruction	A4 (SELECT)
P1	Select method	04 for DF name (AID/PSE)
P2	Occurrence control	00 (first match), or 0C (FCI return)

Field Meaning	For SELECT
Lc Length of data	Length of AID/DF name
Data Actual data	AID or DF name
Le Expected length of response	Optional

GET PROCESSING OPTIONS (GPO) — In-Depth Explanation

The **GET PROCESSING OPTIONS (GPO)** command is a fundamental APDU command in the EMV transaction flow. It follows the SELECT (AID) step and plays a vital role in telling the card how to proceed with the transaction.

Purpose of GPO

The **GPO command** is sent after reading the Application File Locator (AFL) or processing the PDOL (Processing Data Object List). It allows the terminal to get:

- The **Application Interchange Profile (AIP)** – tells the terminal what the card supports
 - The **AFL (Application File Locator)** – tells the terminal which records to read
-

Command Format: GPO

Field Length Description

CLA 1 byte Command class (0x80 or 0x00)

INS 1 byte Instruction (0xA8)

P1 1 byte Parameter 1 (0x00)

P2 1 byte Parameter 2 (0x00)

Lc 1 byte Length of Data Field

Data var. PDOL-related data or 83 format tag

Le 1 byte Expected length of response

Example GPO Request (Case 4 Command)

If PDOL is required:

CopyEdit

```
80 A8 00 00 0B 83 09 00 00 00 00 00 00 00 00 00 00 00
```

Breakdown:

- 80 → Proprietary class
 - A8 → GET PROCESSING OPTIONS
 - 00 00 → P1 and P2 (fixed)
 - 0B → Length of data (11 bytes)
 - 83 → Tag (indicates format)
 - 09 → Length of PDOL-related data (9 bytes)
 - Remaining 9 bytes → PDOL data (e.g., Terminal country, currency, transaction amount, etc.)
 - 00 → Le: expected response length (optional)
-

GPO Response Format

There are two response formats defined in EMV Book 3:

Format 1 (Standard)

bash

CopyEdit

```
77 [len]
```

```
|— 82 [2 bytes] – AIP
```

```
└— 94 [variable] – AFL
```

Format 2 (Simpler)

bash

CopyEdit

```
80 [len]
```

|— [2 bytes] – AIP

└— [n bytes] – AFL

Most cards use **Format 2**, which is shorter and preferred.

Example GPO Response (Format 2)

CopyEdit

80 06 00 80 10 01 01 00

Breakdown:

- 80 → Format 2
- 06 → Length of remaining data
- 00 80 → AIP
 - Bit 15 = 0 → No SDA
 - Bit 14 = 0 → No DDA
 - Bit 9 = 1 → Terminal risk management supported
 - Bit 8 = 0 → Offline data authentication not supported
- 10 01 01 00 → AFL
 - SFI: 2 ($10 \gg 3 = 2$)
 - Record: 1 to 1
 - Number of records to authenticate: 1

AIP (Application Interchange Profile) Summary

The **AIP** tells the terminal which capabilities the card supports:

Bit Position Function

- | | |
|---|---|
| 8 | Offline Static Data Authentication (SDA) |
| 7 | Offline Dynamic Data Authentication (DDA) |
| 6 | Cardholder Verification Supported |
| 5 | Terminal Risk Management Required |

Bit Position Function

- 4 Issuer Authentication Supported
 - 3 Combined DDA/AC Generation Supported
-

AFL (Application File Locator) Format

AFL is a sequence of 4-byte blocks:

Byte Description

- 1 SFI (5 MSBs) + bits 1-3 = 0
- 2 Start record number
- 3 End record number
- 4 Offline data auth record (1 = yes, 0 = no)

You will use the AFL to **read records from card memory** in the READ RECORD step.

Summary of GPO Flow

plaintext

CopyEdit

1. SELECT AID
 2. Parse PDOL (if present)
 3. Terminal builds GPO command (with PDOL values)
 4. Send GPO command
 5. Receive AIP and AFL
 6. Use AFL to perform READ RECORD
-

Real Example

GPO Command:

CopyEdit

80 A8 00 00 0B 83 09 00 00 00 00 00 00 00 00 00 00 00

GPO Response:

CopyEdit

80 06 00 80 10 01 01 00

Tag	Meaning
00 80	AIP: Terminal risk management, CVM, but no SDA/DDA
10 01 01 00	AFL: SFI = 2, record 1 to 1, ODA on record 1

Let's go in-depth into the **GET PROCESSING OPTIONS (GPO)** command, which is crucial in the **EMV (Europay, Mastercard, Visa)** transaction flow.

◆ Purpose of GPO

The **GET PROCESSING OPTIONS (GPO)** command is issued **after the card is selected and the AFL (Application File Locator)** is identified. Its purpose is to **get the card's Application Interchange Profile (AIP)** and the **AFL**, which tells the terminal how to read the card's records.

This step helps the terminal understand:

- What EMV features the card supports (e.g., offline data authentication, cardholder verification, etc.)
- Which records to read next

◆ APDU Format: GPO Request

✖ Format 1 (used when PDOL is present)

css

CopyEdit

CLA INS P1 P2 Lc Data (PDOL Response) Le

80 A8 00 00 XX ... 00

Field	Value	Meaning
CLA	0x80	Proprietary class

Field Value	Meaning
INS 0xA8	Instruction = GPO
P1 0x00	Always 00
P2 0x00	Always 00
Lc Length of the PDOL response template	
Data PDOL data (Tag 83 + length + value)	
Le 0x00	Expected length of response

When PDOL (Processing Options Data Object List) is not empty, this data is required. If there is no PDOL, a default template is used.

Format 2 (used when PDOL is absent)

CopyEdit

80 A8 00 00 02 83 00 00

- 83 00 means tag 83 (PDOL-related data) with length 0.
 - Common when the card does not define PDOL.
-

GPO Response Format

Response is typically in **Response Message Template Format 1 or 2**.

Template 1: (Tag 80 — compact format)

php-template

CopyEdit

80 <len> <AIP> <AFL> [optional data] 9000

Field	Length	Description
80	1 byte	Response tag
Len	1 byte	Length of the following data
AIP	2 bytes	Application Interchange Profile

Field	Length	Description
AFL	4N bytes	Application File Locator
SW1 SW2	9000 if success	

◆ **Example:**

CopyEdit

80 06 12 00 10 01 01 00 90 00

Field Value	Meaning
AIP 12 00	Features supported
AFL 10 01 01 00	SFI 2, Record 1 to 1, Offline
SW 9000	Success

◆ **Template 2: (Tag 77 — BER-TLV structure)**

When the card responds using BER-TLV encoding:

yaml

CopyEdit

77 <len> [Tag 82: AIP] [Tag 94: AFL] 9000

Example:

CopyEdit

77 0A 82 02 12 00 94 04 10 01 01 00 90 00

Tag	Meaning
82 02 12 00	AIP: Offline data auth + online capable
94 04 10 01 01 00	AFL for SFI 2, record 1

◆ **Tags Involved**

Tag Name	Meaning
82 AIP	Application Interchange Profile (card's capabilities)

Tag Name	Meaning
94 AFL	Application File Locator (what records to read next)
83 PDOL	PDOL Data Provided by terminal in GPO command

◆ PDOL and Its Impact

PDOL = list of terminal data elements that the card requires for processing. It is found in tag 9F38 in the FCI Template.

Example PDOL:

CopyEdit

9F66 04 9F02 06 9F37 04 9F1A 02

Terminal needs to construct values for each of these tags and concatenate them into the 83 value in the GPO request.

◆ Typical Flow

1. Terminal selects application (SELECT AID).
 2. Terminal reads PDOL (Tag 9F38) from FCI.
 3. Terminal builds GPO with tag 83 data based on PDOL.
 4. Terminal sends GPO request to card.
 5. Card returns AIP and AFL.
 6. Terminal uses AFL to read records.
-

✓ Summary Table

Step	Terminal Action	Card Response
SELECT	Select AID	FCI with PDOL
Build GPO	Construct PDOL response —	
GPO Request	Send GPO with PDOL data AIP + AFL	
Read Records	Use AFL	Records for CVM, risk mgmt

Let's dive **in depth** into the **READ RECORD** command in EMV (Europay, MasterCard, and Visa) and smart card environments.

Purpose of READ RECORD Command

The **READ RECORD** command is used by the terminal (POS device or ATM) to retrieve **data records** from the card after GPO (Get Processing Options) is completed. These records typically include:

- Track 2 Equivalent Data
 - Cardholder Name
 - PAN (Primary Account Number)
 - Expiry Date
 - CVV / CVV2
 - Application File Locator (AFL)-defined elements
-

READ RECORD APDU Format

Here's the **standard APDU format** of READ RECORD:

mathematica

CopyEdit

CLA | INS | P1 | P2 | Le

---- | ---- | ---- | ---- | --

0x00 | 0xB2 | SFI | 04+Record Number | Le

Breakdown:

Field Meaning	Example
CLA Class byte – always 0x00 for EMV	0x00
INS Instruction – 0xB2 indicates READ RECORD	0xB2
P1 Record number to read (e.g., 1, 2, 3...)	0x01

Field Meaning	Example
P2 (SFI << 3)	0x14 means SFI = 0x02 → 0x02 << 3 = 0x10
Le Length of expected data	0x00 (get full response)

SFI (Short File Identifier): A value from 1 to 30 indicating the file group you're targeting.

How Terminal Knows What to Read?

After the **GPO**, the card returns the **AFL (Application File Locator)** – a TLV field (tag 94). It tells the terminal:

- Which SFIs to read
- How many records to read
- Starting record number

AFL structure (4 bytes per entry):

Byte Meaning

- 1 SFI (upper 5 bits)
 - 2 First record to read
 - 3 Last record to read
 - 4 Number of records involved in offline data authentication (DDA/CDA)
-

Example of AFL:

Let's say card responds with AFL:

CopyEdit

94 08 10 01 03 01 14 01 02 00

Breakdown:

- Entry 1 → SFI = 0x10 >> 3 = 2; Records 1–3; 1 record used for offline authentication
- Entry 2 → SFI = 0x14 >> 3 = 2; Records 1–2; 0 used for offline auth

So the terminal must send:

- 00 B2 01 0C 00 → Read Record 1 from SFI 2
 - 00 B2 02 0C 00 → Record 2 from SFI 2
 - and so on.
-

Sample READ RECORD APDU + Response

◆ Command:

mathematica

CopyEdit

00 B2 01 0C 00

- Record 1, SFI 2 ($0x02 << 3 = 0x10 \rightarrow 0x10 + 0x01 = 0x0C$)

◆ Response:

mathematica

CopyEdit

70 1A

5A 08 5413330000000000 ; PAN

5F24 03 240630 ; Expiry Date (YYMMDD)

5F34 01 01 ; PAN Sequence Number

9F10 07 06011203A00000 ; IAD

- 70 = Template for record data
 - Each tag in the 70-template represents card data elements
-

! Status Words (SW1 SW2)

SW1 SW2 Meaning

9000 Success

6A83 Record not found

6A82 File not found (invalid SFI)

Terminal Loop for READ RECORD

The terminal reads all records based on AFL using a **loop**:

kotlin

CopyEdit

```
for each AFL entry{
```

```
    sfi = getSFI(entry[0])
```

```
    first = entry[1]
```

```
    last = entry[2]
```

```
    for (i = first; i <= last; i++) {
```

```
        send READ RECORD (record i, sfi)
```

```
        if success, store data
```

```
}
```

```
}
```

Key Use Cases

- Getting PAN and expiry for transaction
- Fetching IAD for issuer authentication
- Fetching records required for DDA/CDA

Security Consideration

- If records contain CDA-related data, terminal will verify the digital signature.
- Sensitive tags like Track 2 Equivalent (57), IAD (9F10) should be validated or passed securely to issuer host.

Let's go in-depth on the **READ RECORD** APDU command, which plays a crucial role in the EMV transaction process by retrieving data from the card's file structure.

◆ Purpose of READ RECORD

The **READ RECORD** command retrieves data from the EMV card's Elementary Files (EFs), typically defined in the AFL (Application File Locator) returned in the **GPO** response.

◆ Syntax of READ RECORD APDU

Field Length Description

CLA 1 byte Class (usually 00 or 80)

INS 1 byte Instruction (B2 for READ RECORD)

P1 1 byte Record number

P2 1 byte SFI (Short File Identifier, left-shifted by 3 bits) + 04

Lc - Not used

Le 1 byte Max number of bytes expected in response (usually 00 = get all)

◆ P2 Byte Breakdown

$$P2 = (SFI \ll 3) | 0x04$$

- The **SFI** is 5 bits and must be shifted left 3 bits.
 - The lower 3 bits should be 100 (0x04), which is required by EMV to indicate a READ RECORD command.
-

📌 Example Command

Suppose you want to read **record 1 from SFI 2**:

- CLA: 00
- INS: B2
- P1: 01 → Record number
- P2: $(2 \ll 3) | 4 = 0x10 | 0x04 = 0x14$
- Le: 00

APDU:

CopyEdit

00 B2 01 14 00

Expected Response

The card responds with:

- Data in **BER-TLV format** (usually the record's content: Track 2 equivalent data, cardholder name, etc.)
- Status Word: 9000 (Success)

Example:

yaml

CopyEdit

70 1A 5A 08 1234567890123456 5F24 03 240731 5F34 01 01 9000

This contains:

- Tag 5A (PAN)
 - Tag 5F24 (Application Expiry)
 - Tag 5F34 (PAN Sequence Number)
-

When READ RECORD is Used

After the **GPO** response, the terminal checks the **AFL** (Application File Locator). The AFL indicates which records to read and from which SFIs.

Sample AFL:

CopyEdit

08 01 01 00 10 01 02 00

Means:

- SFI 1, record 1
- SFI 2, record 1

The terminal issues two READ RECORD commands using this structure.

Notes

- If the record/SFI combo is invalid or not present, card returns 6A83 (Record not found).
 - This command is mandatory for the terminal to retrieve key EMV data like:
 - Track 2 Equivalent Data
 - PAN and Expiry
 - CVM List
 - Application Interchange Profile (AIP)
 - Cardholder Name (if present)
-

Possible Response Codes

Status Word Meaning

9000	Success
6A83	Record not found
6A82	File not found (bad SFI)
6985	Conditions of use not satisfied (e.g., CDA required before access)

Real Example

Suppose AFL tells us to read:

mathematica

CopyEdit

SFI 2, Records 1–2

SFI 3, Record 1

You would send:

mathematica

CopyEdit

00 B2 01 14 00 // Record 1, SFI 2

00 B2 02 14 00 // Record 2, SFI 2

00 B2 01 1C 00 // Record 1, SFI 3 (3 << 3 = 0x18, 0x18 | 0x04 = 0x1C)

Let's dive into the **4. GENERATE AC – First Generation (GAC1)** command in the EMV transaction flow, including **what it is, when it occurs, what it contains, and how it's used.**

WHAT IS GAC1?

GAC1 (Generate Application Cryptogram - 1st Generation) is an APDU command sent **after risk management and terminal action analysis**. Its purpose is to:

- Ask the card to generate a cryptogram based on terminal decision
- Include transaction data for the card to evaluate
- Let the card decide:
 - Approve Offline (AAC – Application Authentication Cryptogram)
 - Request Online Authorization (ARQC – Authorization Request Cryptogram)
 - Approve Offline with Transaction Certificate (TC)

WHEN IS GAC1 CALLED?

It is triggered **after**:

1. Terminal reads AIP/AFL (via GPO)
2. Terminal reads relevant records (READ RECORD)
3. Terminal performs:
 - Offline data authentication
 - Risk management (e.g., velocity checking, floor limit)
 - Terminal action analysis
4. Based on terminal's analysis (TVR + IAC), it generates a **Cryptogram Information Data (CID)** request.

Then → Send GAC1 to card.

GAC1 – COMMAND FORMAT

APDU Format (Command to Card):

Field Value (Hex) Description

CLA	80	Proprietary class
INS	AE	GENERATE AC instruction
P1	00	Cryptogram Type Requested (e.g., ARQC)
P2	00	Must be 00
Lc	XX	Length of data field

Data Template 80 Contains transaction data (TVR, amount, date, etc.)

Le	00	Response expected
----	----	-------------------

GAC1 DATA FIELD (Template 80)

This field contains the **Transaction Data Hash Input**, known as **CDOL1 (Card Risk Management Data Object List 1)**.

Sample Data Elements (CDOL1):

Tag Description

9F02 Amount, Authorized

9F03 Amount, Other

9F1A Terminal Country Code

95 Terminal Verification Results (TVR)

5F2A Transaction Currency Code

9A Transaction Date

9C Transaction Type

9F37 Unpredictable Number

Tag Description

... (Can vary depending on CDOL1)

 The terminal must construct the data field using CDOL1 order and send to card.

GAC1 – RESPONSE FROM CARD

Format (Template 77 or 80):

Tag Description

80 Response Message Template Format 1 (if short)

77 Response Message Template Format 2 (detailed)

9F27 Cryptogram Information Data (CID)

9F36 Application Transaction Counter (ATC)

9F26 Application Cryptogram (TC/ARQC/AAC)

9F10 Issuer Application Data (IAD)

9F37 Unpredictable Number

9F1A Terminal Country Code (echoed)

82 AIP (echoed from GPO)

... Other tags may be included

CID (Cryptogram Information Data)

This byte tells the terminal what the card decided:

CID (Binary) Meaning

0x00 AAC – Declined offline

0x40 TC – Approved offline

0x80 ARQC – Request online

SAMPLE DUMP

Command:

```
yaml  
CopyEdit  
80 AE 00 00 2B
```

```
80  
9F02 000000010000  
9F03 000000000000  
9F1A 0840  
95 8000048000  
5F2A 0840  
9A 250716  
9C 00  
9F37 6D453A91  
[... up to CDOL1 length]
```

```
00
```

Response:

```
yaml  
CopyEdit  
77  
9F27 80  
9F36 0023  
9F26 7A4B3E1A2934E711  
9F10 0610A503E00000  
9F37 6D453A91  
82 1800
```

INTERPRETATION:

- 9F27 80 → ARQC (Request Online)

- 9F26 → Generated cryptogram (will be verified by issuer)
 - 9F10 → Issuer Application Data
 - 9F36 → ATC = 35th transaction
 - 9F37 → Echoed unpredictable number
 - 82 → AIP
-

DECISION PROCESS (Simplified)

Terminal uses:

- TVR + IAC (Terminal)
- TSI, AIP (Card)
- CDOL1 data

Then sends GAC1 → Card responds with CID.

If card gives:

- ARQC → Terminal sends it to host
- TC → Offline approved
- AAC → Offline declined

Let's go in-depth on **Generate Application Cryptogram – 1st Generation (GAC1)**.

What is GENERATE AC (GAC1)?

The GENERATE AC command is one of the most critical APDU commands in the EMV transaction flow. It is used to request the card to generate a cryptogram, which will be used to determine how the transaction proceeds (offline approval, online authorization, or decline).

There are two types:

1. **GAC1 (1st Generation)** – Initiated after Terminal Risk Management and CVM process.
2. **GAC2 (2nd Generation)** – Initiated **after** the online response is received (optional, but recommended in online transactions).

GAC1 - Command Format (Request)

This is a **command APDU** from the terminal to the card:

css

CopyEdit

CLA INS P1 P2 Lc Data Le
80 AE 00 00 xx CDOL1 Data elements (TLV list) 00

Byte Description

CLA Command class – 0x80 (proprietary)

INS Instruction byte – 0xAE for GAC

P1 Cryptogram Type Indicator

P2 Usually 0x00

Lc Length of Data field

Data CDOL1 (Card Data Object List 1)

Le Expected length of response (00)

P1 – Cryptogram Type Indicator

Value Meaning

0x80 Request AAC (Application Authentication Cryptogram – Offline Decline)

0x00 Request ARQC (Authorization Request Cryptogram – Online Authorization)

0x40 Request TC (Transaction Certificate – Offline Approval)

GAC1 - Response Format

scss

CopyEdit

Tag Value

80 Cryptogram Data (includes cryptogram, ATC, CID, IAD, etc.)

77 Response Message Template Format 2 (optional)

9F27 Cryptogram Information Data (CID)

9F36 Application Transaction Counter (ATC)

9F26 Cryptogram (ARQC / TC / AAC)

9F10 Issuer Application Data (IAD)



Example CDOL1

A typical **CDOL1** contains:

- 9F02 – Amount, Authorised
- 9F03 – Amount, Other
- 9F1A – Terminal Country Code
- 95 – TVR
- 5F2A – Transaction Currency Code
- 9A – Transaction Date
- 9C – Transaction Type
- 9F37 – Unpredictable Number

The **terminal** collects all these and sends them in the GAC1 command.



Sample GAC1 Command

SCSS

CopyEdit

80 AE 00 00 20

000000000100 (9F02)

000000000000 (9F03)

0256 (9F1A)

8000008000 (95)

0978 (5F2A)

230717 (9A)

00 (9C)

B5C9F497 (9F37)

00

- Lc = 0x20 (32 bytes of data)
 - Last 00 = Le
-

GAC1 – Sample Response

SCSS

CopyEdit

77 2A

9F27 01 80 (CID = 0x80 = AAC)

9F36 02 0025 (ATC = 0x0025)

9F26 08 6B32D1431B65420A (Cryptogram)

9F10 07 06011203A00000 (Issuer Application Data)

CID – Cryptogram Type Returned

CID Value Meaning

0x00 AAC – Decline

0x40 TC – Approve Offline

0x80 ARQC – Online Auth

The response helps the terminal determine the next step in the flow.

When is GAC1 Called?

After these steps:

1. Application Selection
2. GPO (Get Processing Options)
3. Read Records

4. Offline Data Authentication (SDA/DDA/CDA)
 5. Processing Restrictions (AIP, AFL)
 6. CVM Process
 7. Terminal Risk Management
- ➡ Then **GAC1** is invoked.
-

Use Cases

Scenario	Terminal Request Card May Respond with	
Offline approval allowed	TC (P1=0x40)	TC / AAC
Terminal wants to go online ARQC (P1=0x00)	ARQC	/ AAC
Terminal forces decline	AAC (P1=0x80)	AAC

Security Role

- Ensures that the card is not cloned (via dynamic cryptograms)
- Proves card's presence (via ARQC)
- Ties card data to transaction data (Unpredictable Number, TVR)

Yes, we've already covered **GENERATE AC – 2nd Generation (GAC2)** in depth, including:

- **Why it's optional**
- **When it's triggered**
- **What data is involved**
- **Response structure**
- **Realistic example dumps**
- **Cryptographic operations**
- **How it differs from GAC1**

Here's a quick recap in case you need it in one place:

GAC2 – 2nd Generation Generate Application Cryptogram

◆ Purpose

Triggered by the terminal to request the final cryptogram after online authorization is completed.

◆ When It's Used

- After the issuer has responded to an **ARQC** (Authorization Request Cryptogram) sent via GAC1.
- The issuer provides a **response with Issuer Authentication Data (IAD)** and possibly **Issuer Scripts**.
- GAC2 is then used to produce a **TC (Transaction Certificate)** or **AAC (Application Authentication Cryptogram)** based on issuer approval or denial.

◆ Terminal Conditions for Triggering GAC2

- Issuer responded with success (or denial).
- Issuer script present or authentication data needs to be processed.
- Terminal requests the final cryptogram for receipt/logging or security assurance.

GAC2 Command Structure

CLA: 80

INS: AE (Generate AC)

P1: Cryptogram Information Data (CID) — usually 40 or 80

P2: 00

Lc: Length of CDOL2 data

Data: CDOL2 constructed with updated fields (e.g., issuer auth data, updated amounts, time, etc.)

Le: 00

CID in P1 (for GAC2):

- 40 = request TC
- 80 = request AAC

GAC2 Response Structure

Returned data objects in **BER-TLV** format, typically includes:

- **Cryptogram (ARPC / TC / AAC)** — 9F26
 - **CID** — 9F27
 - **ATC** — 9F36
 - **Issuer Authentication Data (IAD)** — 9F10
 - **Application File Locator (AFL)** — sometimes
 - **Script Results (if scripts were executed)**

Example

GAC2 Request (hex):

CopyEdit

80 AE 40 00 1B

00

GAC2 Response (hex):

pgsql

CopyEdit

77 27

9F27 01 40 ; CID – TC requested and generated

9F36 02 004A ; ATC – Application Transaction Counter

9F26 08 2D34A1B247781A90 ; TC

9F10 12 910A0600000000000000000000000000000000FF

! Why GAC2 is Optional

- If the transaction is **declined** by the issuer, and no further cryptogram is needed (e.g., offline AAC was already generated), then **GAC2 isn't triggered**.
 - Some transactions, especially **fully offline**, may **end with GAC1**.
 - If **issuer scripts are not present**, and nothing else needs to be authenticated post-ARQC, GAC2 can be **skipped**.

Great. Let's dive **in-depth** into APDU Command 6: **INTERNAL AUTHENTICATE**, including its purpose, structure, tags involved, and cryptographic role in EMV.

6. INTERNAL AUTHENTICATE (INS = 88)

Purpose

INTERNAL AUTHENTICATE is used for **Static Data Authentication (SDA)** or **Dynamic Data Authentication (DDA)**.

Its **primary purpose** is to:

- Let the terminal **authenticate the card** using a **digital signature**.
 - Confirm the card has not been **cloned** or **tampered** with.
 - Prove that the card has access to its **private key**, especially during **DDA** or **CDA**.
-

When Is It Used?

It depends on what kind of authentication is supported by the card:

Authentication Type	Uses INTERNAL AUTHENTICATE?	Notes
SDA (Static)	 No	Uses static digital signature verified by terminal
DDA (Dynamic)	 Yes	Terminal challenges card
CDA (Combined)	 Internally invoked during GENERATE AC	Real-time dynamic + transaction signing

So:

- **In DDA, INTERNAL AUTHENTICATE is explicitly used.**
 - **In CDA, it's built into GENERATE AC**, not a separate step.
-

APDU Format

Field Description

CLA 0x00 or 0x80

INS 0x88 (INTERNAL AUTHENTICATE)

P1 0x00

P2 0x00

Lc Length of challenge data (e.g., 8 bytes)

Data Dynamic authentication data (usually a **terminal challenge**, 8–16 bytes)

Le 0x00 (expecting full response)



Typical Command Example

Let's say terminal sends an 8-byte random challenge: 1122334455667788

Command:

CopyEdit

00 88 00 00 08 1122334455667788 00



Response Format

The response is typically a **digital signature** over the challenge and other card data.

The signature is returned in the format:

kotlin

CopyEdit

(EMV format) 0x9F4B followed by length and signature data

But **in raw APDU**, the card returns:

- The **Signed Dynamic Application Data (SDAD)**: signature using the card's private key.
- Status Word: 9000 (success)

Example Response:

java

CopyEdit

77 1A

9F4B 18 <Signature> ; 24-byte digital signature (RSA-1024)

The **terminal must verify** this signature using the **card's public key**, which is found from:

- **ICC Public Key Certificate** (from card, tag 9F46)
 - **Issuer Public Key Certificate** (from AID config)
 - **CA Public Key** (from EMV kernel)
-

What's in the Signature (SDAD)?

The SDAD (Signed Dynamic Application Data) is RSA-encrypted and includes:

Field	Description
Header (6A)	Signature Header
Format ID	Usually 05
Hash Algorithm Indicator e.g., 01 for SHA-1	
ICC Dynamic Data	Includes challenge
Padding	Optional
Hash Result	Hash of signed data
Trailer (BC)	Signature trailer byte

Terminal uses card's public key to **decrypt** and validate:

- The **challenge matches**
 - The **hash is correct**
 - The **padding and structure are valid**
-

Key Points to Remember

Feature	Notes
Card's Role	Proves it holds private key matching certified public key
Terminal's Role	Validates signature with public key chain (ICC → Issuer → CA)
Challenge Size	Typically 8–16 bytes
Signature Type	RSA, often 1024 or 2048-bit
Tag for Signature	9F4B (Signed Dynamic Application Data)
Decline if Failed	If signature is invalid, card is likely a clone → transaction rejected

Real-Life Example

Command:

CopyEdit

00 88 00 00 08 0123456789ABCDEF 00

Response:

css

CopyEdit

9F4B 18 6A0501123456789ABCDE...(rest of signature)...BC

9000

What if Internal Authenticate Fails?

- If the **signature fails**, it indicates **card authenticity failure**.
- Terminal **declines the transaction** with an appropriate reason code (e.g., authentication failed).
- In some implementations, may retry another method (e.g., fallback to SDA if supported).

Summary

Component	Value
Command	INTERNAL AUTHENTICATE (88)
Used In	DDA (explicit), CDA (internal), not SDA
Terminal Provides	Challenge (8–16 bytes)
Card Returns	Digital Signature (SDAD, 9F4B)
Signature Validated Using	ICC Public Key (→ Issuer → CA)
Security Purpose	Ensure card is genuine and holds valid private key

Let's explore **EXTERNAL AUTHENTICATE** in-depth — including its purpose, structure, usage in the EMV transaction flow, and real-world example.

◆ What is EXTERNAL AUTHENTICATE?

The **EXTERNAL AUTHENTICATE** command is part of **Static Data Authentication (SDA)** or **Dynamic Data Authentication (DDA)** in the EMV flow and is used during **offline data authentication**.

It allows the terminal (specifically the **issuer** via the terminal) to **authenticate the card**, i.e., verify the card has not been tampered with and is genuine.

📌 **Purpose:** To allow the **terminal to verify** the authenticity of the card by sending a cryptographic challenge that only a genuine card can decrypt using its private key.

◆ Where in the EMV Flow?

EXTERNAL AUTHENTICATE is used:

- **Only in offline transactions**
 - Specifically **after the terminal receives Signed Dynamic Application Data (SDAD)** from the card (from INTERNAL AUTHENTICATE or earlier processing)
 - Part of **Dynamic Data Authentication (DDA)** or **Combined DDA/AC Generation (CDA)**
-

◆ APDU Format – EXTERNAL AUTHENTICATE

Field Description

CLA Class byte (0x00 usually)

INS Instruction byte (0x82 for EXTERNAL AUTHENTICATE)

P1 Parameter 1 (0x00)

P2 Parameter 2 (0x00)

Lc Length of data sent

Data Authentication data (cryptographic data from issuer)

Le Length of expected response (often omitted or 00)

Example APDU

ruby

CopyEdit

```
=> 00 82 00 00 10 112233445566778899AABBCCDDEEFF00
```

- 00 → CLA
- 82 → INS (EXTERNAL AUTHENTICATE)
- 00 00 → P1, P2
- 10 → Lc = 16 bytes
- 112233445566778899AABBCCDDEEFF00 → Challenge data (issuer-created cryptogram)

◆ What Happens During Execution?

1. Issuer creates a cryptogram using:

- The card's public key (obtained via certificate chain)
- A random challenge
- Terminal-specific transaction data

2. Terminal sends EXTERNAL AUTHENTICATE command with the cryptogram

3. Card decrypts it using its private key (stored securely)

4. If valid:

- The card confirms the authenticity of the issuer/terminal
 - No explicit response data (only 0x9000 success SW)
-

◆ Response Format

If successful:

yaml

CopyEdit

<= 9000

If failure (e.g., authentication failed):

kotlin

CopyEdit

<= 6985 (Conditions of use not satisfied)

<= 6A80 (Incorrect data)

◆ Real Use Case Scenario

Suppose a terminal is performing **Combined DDA/AC**. It must:

- First do **INTERNAL AUTHENTICATE** to verify card authenticity
- Then do **EXTERNAL AUTHENTICATE** to prove the **issuer/terminal authenticity** to the card

The **mutual authentication** ensures:

- The **terminal/card is genuine**
 - The **issuer is trusted**
-

◆ EMV Books Reference

- **Book 2 – Security and Key Management**
 - **Book 3 – Application Specification**, especially for APDU flow
-

✓ Summary

Aspect	Description
Command	EXTERNAL AUTHENTICATE (INS = 0x82)
Purpose	Card authenticates the issuer/terminal using issuer-provided challenge
Used In	Offline Authentication (DDA, CDA)
Success Response	9000
When Used	After INTERNAL AUTHENTICATE or SDAD verification
Main Requirement	Card must support public key decryption with its private key

Great! Here's an **in-depth explanation of the EXTERNAL AUTHENTICATE command** in the EMV transaction flow:

7. EXTERNAL AUTHENTICATE (INS = 82)

Purpose

The EXTERNAL AUTHENTICATE command is used **by the terminal to authenticate itself to the card** using a cryptographic challenge-response mechanism. It is part of **Static or Dynamic Data Authentication**, but **this command is rarely used in modern EMV implementations** because most cards use **mutual authentication via certificates** (such as in **CDA** or **DAA**), not terminal authentication.

Where it occurs in the flow

- It occurs **after card authentication** (Static or Dynamic Data Authentication), **before transaction authorization** (Generate AC).
 - Used **only if terminal supports and requests terminal authentication**, and **card supports it** (checked via AIP).
-

How it works

1. The card sends a **random number (Unpredictable Number)**.
2. The **terminal encrypts this number using its private key**, forming the **cryptogram**.

3. The terminal sends this cryptogram in the EXTERNAL AUTHENTICATE command.
 4. The **card uses the terminal's public key** (from the certificate chain) to **verify** the cryptogram.
 5. If verification is successful, the terminal is authenticated.
-

Command Format (APDU)

Field Value Description

CLA 0x00 Command class

INS 0x82 Instruction code (EXTERNAL AUTHENTICATE)

P1 0x00 No meaning – usually 00

P2 0x00 No meaning – usually 00

Lc Length Length of authentication data

Data X bytes Authentication data (cryptogram)

Le – Not usually present

Example:

mathematica

CopyEdit

00 82 00 00 10 8A 4F 1C 78 6F 3D 72 3E 5F 49 25 3A 6D 22 8D

This 10-byte field is the cryptogram (dynamic signature) computed by the terminal using its private key.

Response Format

Field Description

SW1 SW2 9000 for success

Optional Data None

Example:

CopyEdit

90 00

If SW1 SW2 is not 9000, the card failed to verify the terminal.

When is EXTERNAL AUTHENTICATE used?

Used **only if the card and terminal both support terminal authentication**, which is rare now. Commonly seen in **offline-capable environments** like:

- Government services
- Offline-capable ATMs
- Secure access systems

Most **online EMV transactions skip this step** and rely on **issuer authentication** instead.

Summary Table

Component Detail

Command EXTERNAL AUTHENTICATE (82)

Purpose Terminal proves its authenticity to card

Trigger If terminal and card both support terminal authentication

Input Encrypted cryptogram using terminal's private key

Output 90 00 if authentication is successful

Dependency Requires AIP bit set for terminal authentication

Here's an **in-depth explanation of the EMV PUT DATA and GET DATA commands**, their purpose, structure, and real-world usage:

8. PUT DATA / GET DATA – In-Depth

These commands are used to **exchange individual data elements (tags)** between the terminal and the card during an EMV transaction. Unlike READ RECORD, which reads predefined files, GET DATA and PUT DATA work on **single data objects**.

GET DATA (INS = CA)

Purpose

To **retrieve a single tag's value** from the card.

When is it used?

- To obtain **card risk management data** like:
 - 9F36 – Application Transaction Counter (ATC)
 - 9F13 – Last Online ATC Register
 - 9F17 – PIN Try Counter
 - 9F10 – Issuer Application Data (in some cases)
 - 9F4F – Log Format
- Used **after GENERATE AC**, to read post-authorization status.

Format

Command APDU:

Field	Value	Description
--------------	--------------	--------------------

CLA	80	Proprietary
-----	----	-------------

INS	CA	GET DATA
-----	----	----------

P1	First byte of tag
----	-------------------

P2	Second byte of tag
----	--------------------

Lc	–	Not used
----	---	----------

Data	–	None
------	---	------

Le	00	Get all data for tag
----	----	----------------------

Example:

To get tag 9F36 (ATC):

CopyEdit

80 CA 9F 36 00

Response:

CopyEdit

9F36 02 00 14 90 00

Here, 9F36 02 00 14 → Tag 9F36, length 2 bytes, value 0014 (decimal 20)

 **Tags Commonly Retrieved Using GET DATA**

Tag Meaning

9F36 Application Transaction Counter (ATC)

9F17 PIN Try Counter

9F13 Last Online ATC Register

9F10 Issuer Application Data

9F4F Log Format

 **PUT DATA (INS = DA)**

 **Purpose**

To **store data elements** (usually keys or settings) **into the card**.

 **When is it used?**

- Used in **card personalisation** or **issuer script processing**.
- In normal consumer EMV flow, **not typically used**.
- Can be used by the issuer to send updates to the card (e.g., new keys, offline limits) during **Issuer Scripts**.

 **Format**

Command APDU:

Field Value	Description
CLA 80	Proprietary
INS DA	PUT DATA
P1	First byte of tag
P2	Second byte of tag
Lc	Length of data
Data	Tag value
Le	– Usually omitted

Example:

To set tag 9F1A (Terminal Country Code) to 0250 (France):

r

CopyEdit

80 DA 9F 1A 02 02 50

Response:

CopyEdit

90 00

This confirms that the card accepted the data.

 **Summary Table**

Feature	GET DATA	PUT DATA
Instruction	CA	DA
Purpose	Read specific tag from the card	Write specific tag to the card
Usage	Transaction monitoring/logging	Issuer scripts or personalisation
Availability	Most cards support reading a few tags	Rare in normal terminals
Example Tag	9F36 – ATC	9F1A – Terminal Country Code

Security Notes

- PUT DATA may require **secure messaging** or **issuer authentication**.
 - Misuse of PUT DATA can lead to transaction failure or security breach.
-

Typical Transaction Usage

Stage	Command	Purpose
Before GAC	GET DATA 9F36	Read current ATC
After GAC	GET DATA 9F17	Check PIN Try Counter
After AC	PUT DATA (via issuer script)	Update offline limits, keys, etc.