EMV®
Contactless Specifications for Payment Systems

Kernel 8 Guidance

Version 1.0.1
September 2024

Legal Notice

The EMV® Specifications are provided “AS IS” without warranties of any kind, and EMVCo neither assumes nor accepts any liability for any errors or omissions contained in these Specifications. EMVCO DISCLAIMS ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AS TO THESE SPECIFICATIONS.

EMVCo makes no representations or warranties with respect to intellectual property rights of any third parties in or in relation to the Specifications. EMVCo undertakes no responsibility to determine whether any implementation of the EMV Specifications may violate, infringe, or otherwise exercise the patent, copyright, trademark, trade secret, know-how, or other intellectual property rights of third parties, and thus any person who implements any part of the EMV Specifications should consult an intellectual property attorney before any such implementation.

Without limiting the foregoing, the Specifications may provide for the use of public key encryption and other technology, which may be the subject matter of patents in several countries. Any party seeking to implement these Specifications is solely responsible for determining whether its activities require a license to any such technology, including for patents on public key encryption technology. EMVCo shall not be liable under any theory for any party’s infringement of any intellectual property rights in connection with the EMV Specifications.

Revision Log – Version 1.0.1

This section outlines notable updates that have been made to this document since the publication of the EMV Contactless Kernel 8 Guidance, Version 1.0.

This version includes following change:

- Update description of chapter 3.5.5.

Contents

Tables

Figures

1 Using This Manual

1.1 Purpose

This document, *EMV® Contactless Specifications for Payment Systems, Kernel 8 Guidance*, provides guidance on the implementation of contactless transactions using Kernel 8, as defined in the following specifications:

- *EMV Contactless Specifications for Payment Systems, Book A – Architecture and General Requirements*, hereafter referred to as [EMV Book A]
- *EMV Contactless Specifications for Payment Systems, Book B – Entry Point Specification*, hereafter referred to as [EMV Book B]
- *EMV Contactless Specifications for Payment Systems, Book C-8 – Kernel 8 Specification*, hereafter referred to as [EMV Book C-8]
- *EMV Contactless Specifications for Payment Systems, Book E – Security and Key Management*, hereafter referred to as [EMV Book E]

1.2 Audience

This guidance document is intended for use by those involved in the manufacture or deployment of contactless readers and terminals, contactless cards, and by financial institution staff.

1.3 Related Information

The following references are used in this document. The latest version applies unless a publication date is explicitly stated.

Table 1.1—Related Information

Reference	Document Title
[EMV Book A]	EMV Contactless Specifications for Payment Systems, Book A – Architecture and General Requirements
[EMV Book B]	EMV Contactless Specifications for Payment Systems, Book B – Entry Point Specification
[EMV Book C-8]	EMV Contactless Specifications for Payment Systems, Book C-8 – Kernel 8 Specification

Reference	Document Title
[EMV Book E]	EMV Contactless Specifications for Payment Systems, Book E – Security and Key Management
[ISO/IEC 14888-3]	Information technology — Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms

1.4 Terminology

The following terms are used in this document, carrying specialised meanings as indicated.

For abbreviations used in this document, see Table A.1.

Table 1.2—Terminology

Term	Description
Card	Card, as used in these specifications, is a consumer device supporting contactless transactions.
Combination	Combination is the combination of an AID and a Kernel ID.
Configuration Option	A Configuration Option allows activation or deactivation of the Kernel software behind this option. The Configuration Option may change the execution path of the software but it does not change the software itself. A Configuration Option is set in the Kernel database per AID and Transaction Type.
Implementation Option	An Implementation Option allows the vendor to select whether the functionality behind the option will be implemented in a particular installation.
Kernel	The Kernel contains the interface routines, security and control functions, and logic to manage a set of commands and responses to retrieve all the necessary data from the Card to complete a transaction. The Kernel processing covers the interaction with the Card between the selection of the card application (excluded) and the processing of the transaction's outcome (excluded).

Term	Description
POS System	The POS System is the collective term given to the payment infrastructure present at the merchant. It is made up of the Terminal and Reader.
Queue	A Queue is a buffer that stores events to be processed. The events are stored in the order received.
Reader	The Reader is the device that supports the Kernel(s) and provides the contactless interface used by the Card. In this specification, the Reader is considered as a separate logical entity, although it can be an integral part of the POS System.
Signal	A Signal is an asynchronous event that is placed in a Queue. A Signal can convey data as parameters, and the data provided in this way is used in the processing of the Signal.
Terminal	The Terminal is the device that connects to the authorisation and/or clearing network and that together with the Reader makes up the POS System. The Terminal and the Reader may exist in a single integrated device. However, in this specification, they are considered separate logical entities.

1.5 Notations

Table 1.3—Notational Conventions

Notation	Meaning	Example
'0' to '9' and 'A' to 'F'	Hexadecimal notation. Values expressed in hexadecimal form are enclosed in straight single quotes.	27509 decimal is expressed in hexadecimal as '6B75'.
340	Decimal notation. Values expressed in decimal form are not enclosed in single quotes.	'0C' hexadecimal is expressed in decimal as 12.
C-APDU	C-APDUs are written in all caps to distinguish them from the text	GET PROCESSING OPTIONS

2 Transaction Flow

This section summarises the transaction phases and functions that may be performed during a Kernel 8 transaction. Functionality in both the Kernel and the Card is described to give a complete picture of how a Kernel 8 transaction is typically performed.

Note: The exact functionality supported or performed by different payment systems may vary depending on the issuer's chosen configuration for their Cards or depending on the acquirer or merchant configuration in the Terminal.

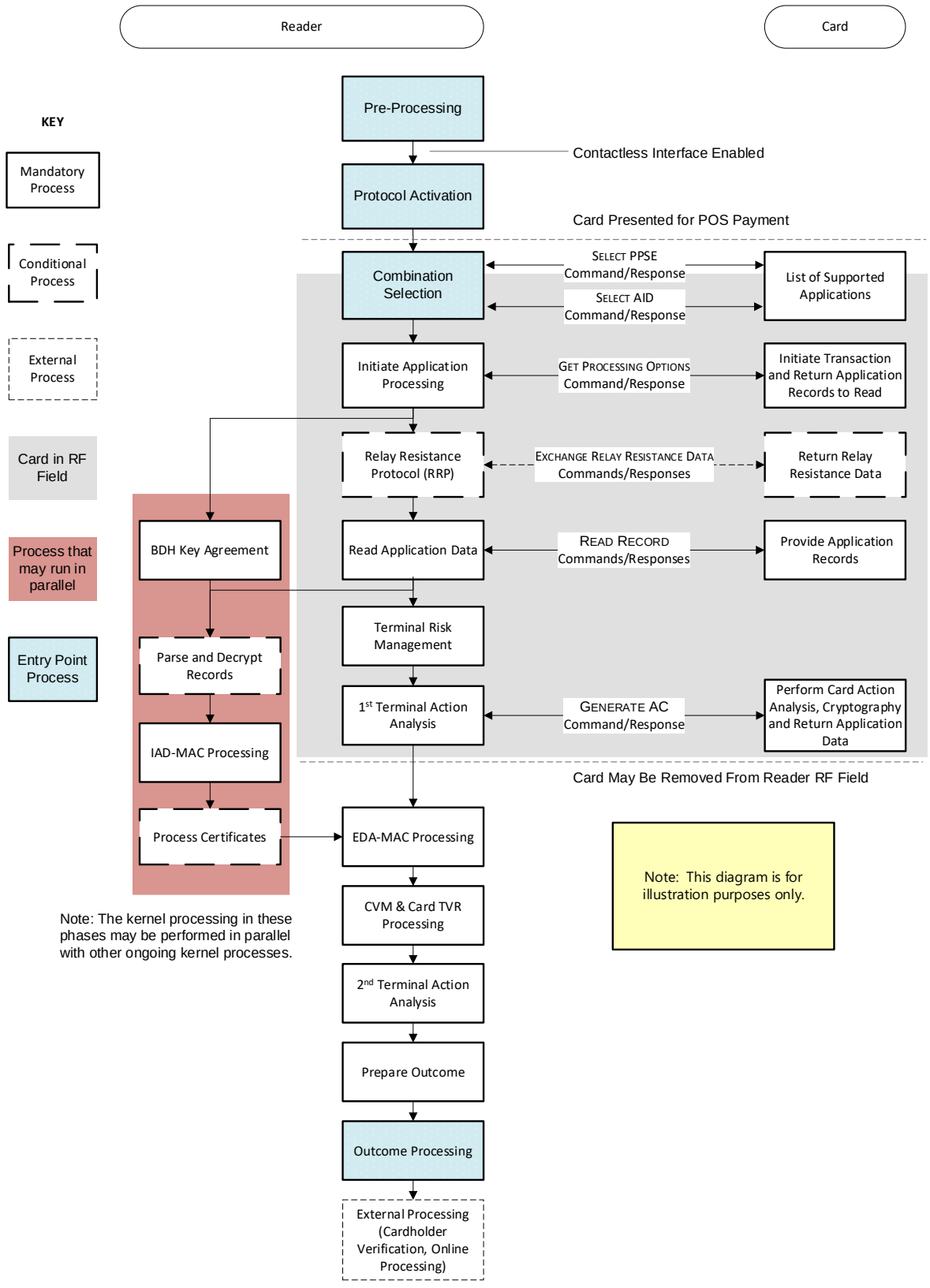
Kernel 8 transactions are fully compliant with the general architecture of a POS System as described in [EMV Book A] and the multi-kernel architecture described in [EMV Book B].

Kernel 8 can operate in terminals alongside other Kernels, including existing payment system-proprietary Kernels. Similarly, Cards can support transactions that may be performed by more than one Kernel – for example, Kernel 8 or the payment system-proprietary Kernel – and the Card selects the relevant functionality for each transaction based on the reader-activated Kernel at the POS System.

Many of the transaction phases defined for an EMV-compliant transaction are retained from existing contactless Kernels. Some phases or components of a Kernel 8 transaction are newly introduced by this Kernel. The following sections provide a high-level overview of both new and existing functionality.

2.1 Diagram of Transaction Flow

Figure 2.1—Sample Transaction Flow



2.2 Pre-Processing

(The processing described in this section is not specific to Kernel 8.)

Pre-Processing occurs prior to the activation of the contactless interface of the reader and before the cardholder is invited to present a contactless card.

Pre-Processing is usually performed for new transactions with a variable amount. In some implementations, for example where the reader processes a fixed transaction amount, Pre-Processing may not be required, and the transaction may start at Protocol Activation.

Each reader-supported Combination of AID and Kernel ID may have a set of configuration data, which is examined during Pre-Processing. The result is a set of flags and data objects for each supported Combination, one of which will be used to define the attributes of the transaction.

2.3 Protocol Activation

(The processing described in this section is not specific to Kernel 8.)

During Protocol Activation, the reader polls for the presence of contactless cards that may have entered the reader's RF field.

When a Card is detected, the reader moves to Application Selection and Kernel Activation.

2.4 Combination Selection

(The processing described in this section is not specific to Kernel 8.)

Application Selection is performed immediately after activation of the contactless card and is the process of determining which of the applications that are supported by both the Card and reader will be used to conduct the transaction. This process is performed in two steps:

1. The reader constructs a prioritised list of AID and Kernel ID Combinations mutually supported by the contactless card and the reader.

2. Once all supported Combinations have been found and the highest priority Combination has been identified, the reader selects the associated card application by sending a SELECT (AID) command.

For the purposes of the remainder of this overview, it is assumed that a card application that supports Kernel 8 has been selected for the transaction, and the Kernel 8 reader application has been activated.

2.5 Initiate Application Processing (GPO)

During Initiate Application Processing, the Kernel signals to the Card that transaction processing is beginning by sending the GET PROCESSING OPTIONS (GPO) command to the Card. When issuing this command, the Kernel supplies the Card with any data objects requested by the Card in the Processing Options Data Object List (PDOL). Because Kernel 8 supports Elliptic Curve Cryptography (ECC), the PDOL includes the Kernel's ephemeral ECC public key used to establish the secret session keys for the secure channel.

The Card uses its private key, the Kernel's ephemeral ECC public key, and a blinding factor to generate the pair of secret session keys used to establish the secure channel for the transaction. The Card GPO response to the Kernel includes the card blinded public key and the encrypted blinding factor, allowing the Kernel to subsequently generate the same secret session keys for the secure channel.

Following receipt of the GPO response, the Kernel establishes whether the Relay Resistance Protocol (RRP) is supported by both Kernel and Card. If so, the Kernel proceeds to exchange RRP data as described in section 2.7. It may also initiate other parallel processes such as the BDH key agreement described in section 2.6.

2.6 BDH Key Agreement

During BDH key agreement, the Kernel uses its private key and the card blinded public key to generate the secret session keys for the secure channel.

Kernel generation of the session keys may occur as soon as the Kernel receives the GPO response from the Card, but to avoid increasing the time the Card is required to remain in the RF field, this processing may occur after the Card has been removed from the reader field or may be performed in parallel with other kernel processing.

Note: Although the establishment of the session keys by the Kernel may be performed in parallel with other ongoing kernel processing, it must be completed before the Kernel can make use of the keys (for example, to decrypt records (see section 2.10) or to generate the Kernel's IAD-MAC and EDA-MAC (see sections 2.14 and 2.15)).

2.7 RRP (Exchange Relay Resistance Data)

Relay Resistance Protocol (RRP) provides some mitigation against basic relay attacks and makes it more difficult to relay a contactless transaction by introducing additional challenges into the transaction flow.

If the Kernel supports RRP and the Card indicates support for RRP, then the Kernel performs RRP following receipt of the GPO response.

During RRP, the Kernel sends the EXCHANGE RELAY RESISTANCE DATA command to the Card and measures the time it takes the Card to respond to the command. The Kernel compares the time measured against Card and Kernel time limits and thresholds of the expected card time needed to respond to the command. If those are exceeded, the Kernel sets Terminal Verification Results (TVR) bits to indicate that RRP time limits and/or thresholds were exceeded.

The reader is configured with RRP timing values for the protocol, which are established at the level described in section 3.9.3. The Card is personalised with RRP timing values that are defined by the card manufacturers, as described in section 3.9.2.

If the RRP processing time is greater than the maximum permitted time defined by the Card (plus a grace period defined by the reader), then RRP can be retried a maximum of two times.

If the RRP processing time is less than the minimum permitted time defined by the Card (minus a grace period defined by the reader), then the transaction is terminated.

2.8 Data Exchange/Data Storage Read Phase (Read Data)

Data Storage is an extension of the regular transaction flow such that the Card can be used as a scratch pad or mini data store with simple write and read functionality.

Data Storage relies on the Data Exchange mechanism and is only supported if the DE/DS Implementation Option is implemented. Data Storage uses dedicated commands (READ DATA and WRITE DATA) for explicit reading and writing of data.

Following the GPO command (or following RRP if supported), if the Kernel supports DE/DS and either the reader has requested some envelopes to read or there are envelopes in the configuration data, the Kernel sends the READ DATA command. During this phase, the Kernel uses ‘Tags to Read’ (a configuration data object) that may contain one or more pre-defined tags that are read from the Card.

The READ DATA command is repeated until all the data objects in the list have been read by the Kernel and stored for potential use by the Terminal.

When a READ DATA command is used, the data is protected by a MAC computed with one of the session keys so that the Kernel may have confidence that it was received unaltered.

See section 3.4.1 for more detail on Data Storage/Data Exchange.

2.9 Read Records

If the Card indicates that any records are to be read, the Kernel performs the READ RECORD command after the GPO command, or after the RRP command if RRP was performed, or after the Read Phase if DE/DS data was read from the Card (which may also follow performance of RRP). This command is used repeatedly until the Kernel has received all the records indicated by the Application File Locator (AFL) that was returned by the Card in the GPO response.

The Card may return unencrypted records and/or encrypted records containing sensitive data in response to the READ RECORD command. Sensitive data includes the blinding factor returned by the GPO command and any data returned by a READ RECORD command that uniquely identifies the Card. Returning encrypted sensitive card data prevents eavesdropping and tracking by a third party. Issuers identify and personalise the card data that they wish to encrypt in records returned to the Kernel. Records are encrypted by the Card using the Session Key for Confidentiality (SKC) established for the secure channel to provide privacy protection when returning data. The Kernel subsequently decrypts the records for further use in the transaction (see section 2.10).

2.10 Decrypt Records

If the Card returns any encrypted records, the Kernel uses the Session Key for Confidentiality to decrypt the encrypted records returned by the Card.

Kernel decryption of records may occur as soon as the Kernel generates the secret session keys for the transaction, but to avoid increasing the time the Card is required to remain in the RF field, this processing may occur after the Card has been removed from the reader field or may be performed in parallel with other kernel processing.

Note: As stated, the decryption of records may be performed as a parallel activity to other ongoing kernel processing but must be completed before the Kernel can make use of the data objects in the records.

2.11 Certificate Processing (Local Authentication)

Local authentication processing is performed if both the Card and Kernel support local authentication. The Kernel may initiate the local authentication process as soon as all the required data is received from the Card following GPO and READ RECORD processing.

During local authentication processing, the Kernel validates the Issuer Public Key Certificate, the ICC Public Key Certificate, and the blinding factor to perform local authentication of the Card.

The Kernel combines the results of local authentication processing (if performed) with verification of the EDA-MAC (see section 2.15) as part of the enhanced data authentication process introduced in Kernel 8.

See section 3.4.3 for more information on local authentication.

Note: The Kernel may indicate in the TVR if local authentication has failed. However, the Kernel can clear that information from the data record output to the Terminal, in order to simplify application cryptogram validation by the issuer. See section 3.8.2 for more detail.

2.12 Terminal Risk Management and 1st Terminal Action Analysis

Once the Kernel has the data it needs from the previous steps to proceed, it performs Terminal Risk Management to identify reader CVM options, perform floor limit and CVM limit checks, and may initiate local authentication if supported and enabled (see section 2.11).

During 1st Terminal Action Analysis, the Kernel determines its transaction disposition and then sends the GENERATE APPLICATION CRYPTOGRAM (GENERATE AC) command to the Card to retrieve the Application Cryptogram and other application data from the Card.

The Card then performs Card Action Analysis and responds to the Kernel (see section 2.13).

At the conclusion of 1st Terminal Action Analysis, Kernel and Card interaction is complete and the Card may be removed from the reader field.

2.13 Card Action Analysis (Generate AC)

When the Card receives the GENERATE AC command, it performs Card Action Analysis.

During Card Action Analysis of a Kernel 8 transaction, the Card may optionally perform application checks and processing restrictions, and then performs cardholder verification decisioning, determines the card transaction disposition (for example, request online processing, request offline processing, decline offline, or switch interfaces), generates the IAD-MAC, generates the Application Cryptogram, generates the EDA-MAC, and responds to the GENERATE AC command with the outcome of its processing.

The Card's choice of CVM is indicated in the Cardholder Verification Decision data object returned in the GENERATE AC response. Subsequently, the Kernel examines the Card's choice of the CVM to be performed for the transaction. See section 3.3 for more detail on cardholder verification.

The Card generates the IAD-MAC and EDA-MAC using the Session Key for Integrity (SKI) previously established during GPO processing (see section 2.5).

See section 3.4 for more detail on card and data authentication and section 3.7 for more detail on the Card's implementation of processing restrictions in Kernel 8 transactions.

2.14 Kernel IAD-MAC Processing

During GENERATE AC processing (as discussed in section 2.13), the Card generates its IAD-MAC and uses it to generate its EDA-MAC. The Card does not pass the IAD-MAC to the Kernel.

After receiving the GENERATE AC response from the Card, the Kernel:

- Generates its own IAD-MAC
- If requested by the Card, copies the Kernel IAD-MAC into the Issuer Application Data (IAD)

The issuer may subsequently use the Kernel-generated IAD-MAC in remote authentication of the card data, as described in section 3.4.3.

2.15 Kernel EDA-MAC Processing

During GENERATE AC processing (as discussed in section 2.13), the Card generates the EDA-MAC over the Application Cryptogram and the IAD-MAC.

After receiving the GENERATE AC response from the Card, the Kernel:

- Generates its own EDA-MAC
- Compares its EDA-MAC to the EDA-MAC returned by the Card in the GENERATE AC response

If the EDA-MAC generated by the Kernel does not match the EDA-MAC returned by the Card, the Kernel terminates the transaction.

2.16 Data Exchange/Data Storage Write Phase (Write Data)

After the GENERATE AC command, the Kernel may send one or more WRITE DATA commands to the Card if the DE/DS option is supported and the kernel configuration data contains a set of data objects to be written to the Card.

Data Storage protects data being sent to the Card and received from the Card using the same privacy mechanism as the one used to protect the card data returned to the Kernel in response to READ RECORD commands. When data is written by the Kernel using WRITE DATA, the Card returns a MAC computed over the recovered plaintext to provide confidence that it was received by the Card unaltered. See section 2.8 and section 3.4.1 for more information.

2.17 2nd Terminal Action Analysis

Following receipt of the GENERATE AC response from the Card, the Kernel performs the 2nd Terminal Action Analysis phase, during which it evaluates the updated results of processing to determine whether the transaction should be approved, declined offline, or sent online for authorisation.

Following the determination of the final transaction disposition, the Kernel builds the data record (including any discretionary data) that is to be passed back to the Terminal for further processing. The Kernel then passes this data back to the reader and exits. This completes kernel processing.

2.18 Outcome Processing

Kernel processing is complete and the Kernel has provided the Outcome Parameter Set, Data Record (if any), Discretionary Data, and any User Interface Request Data to the Terminal.

Subsequent processing of the transaction outcome and data returned from the Kernel is outside the scope of the Kernel 8 specification.

3 Functional Details

This section provides additional detail about the following aspects of Kernel 8 transaction processing:

3.1 Secure Channel

3.1.1 Introduction

A secure channel serves as a vital security measure, safeguarding contactless transactions against both passive eavesdropping and active Man-in-The-Middle attacks. It prevents unauthorised interception of sensitive data and thwarts attempts to track cardholder activities. Its primary purpose is to ensure the confidentiality and integrity of data during transmission between the Card and the Terminal while also supporting local authentication processes.

Note: The secure channel does not ensure that the Card is communicating with a genuine terminal, and while the secure channel protects against eavesdropping of data it does not provide protection against pickpocketing of data.

3.1.2 Secure Channel Establishment

The establishment of a secure channel between the Kernel and the Card ensures privacy protection and verifies the authenticity of both the Card and the transmitted data. This secure channel relies on the Blinded Diffie-Hellman (BDH) protocol, which uses ECC public key cryptography.

To initiate this secure channel, the Kernel generates an ephemeral ECC key pair, while the Card uses a fixed ECC key pair personalised by the Issuer. The ICC ECC public key, certified by the Issuer through an ICC certificate, remains unchanged. To preserve privacy, the ICC ECC public key undergoes anonymisation in each transaction. This involves multiplying the ICC ECC public key by a randomly generated integer known as the blinding factor, resulting in the ICC ECC blinded public key.

Using both the ICC ECC blinded public key and the Kernel's ephemeral ECC public key, the Card and Kernel compute the BDH shared secret. From this shared secret, two AES session keys are derived, responsible for ensuring the confidentiality and integrity of the transaction. The Kernel performs validation of the blinding factor as part of card authentication.

For an in-depth explanation of the BDH key agreement and the process of deriving session keys, a detailed description is available in [EMV Book E].

3.1.3 Usage

Once a secure channel is established between the Card and the Kernel, it remains intact throughout the entire transaction lifecycle. Its primary purposes include:

- Privacy Protection: This ensures that sensitive information exchanged between the Card and the Kernel remains encrypted, safeguarding it from unauthorised access or interception. For further details on Privacy Protection, refer to section 3.2.
- Card and Data Authentication: The secure channel facilitates the verification of the Card's legitimacy and the integrity of the transmitted data. For more information on Card and Data Authentication, refer to section 3.4.

The establishment and maintenance of this secure channel are essential for maintaining transaction security, protecting sensitive data, and preventing unauthorised access or tampering.

3.2 Privacy Protection

3.2.1 Introduction

Privacy Protection acts as a defence mechanism against eavesdropping during card-to-reader communication, ensuring that the Card's identity used in a transaction remains concealed. This method guarantees that the payment application data does not reveal whether two transactions were conducted by the same Card or payment application, preserving the anonymity of the Card involved in the transactions.

3.2.2 Privacy Protection Flow

Privacy protection employs the Session Key for Confidentiality and uses the AES block cipher to encrypt specific record data that the Card provides and that uniquely identifies the Card. The Card should encrypt privacy sensitive information as described in [EMV Book E]. Privacy sensitive information might include the Application PAN, Track 2 Equivalent Data, ICC Public Key Certificate, ICC RSA Public Key Remainder, ICC ECC Public Key (for RSA Certificates), and/or additional data objects specified by the payment system or issuer.

Moreover, the scope of privacy protection extends to encompass the READ DATA and WRITE DATA commands, which might be used to read from and write to any of the ten Data Envelopes (see section 3.4.1). This ensures that the encryption mechanism covers various data interaction scenarios.

For comprehensive insights into the encryption mechanism employed for privacy protection, detailed information can be found in [EMV Book E].

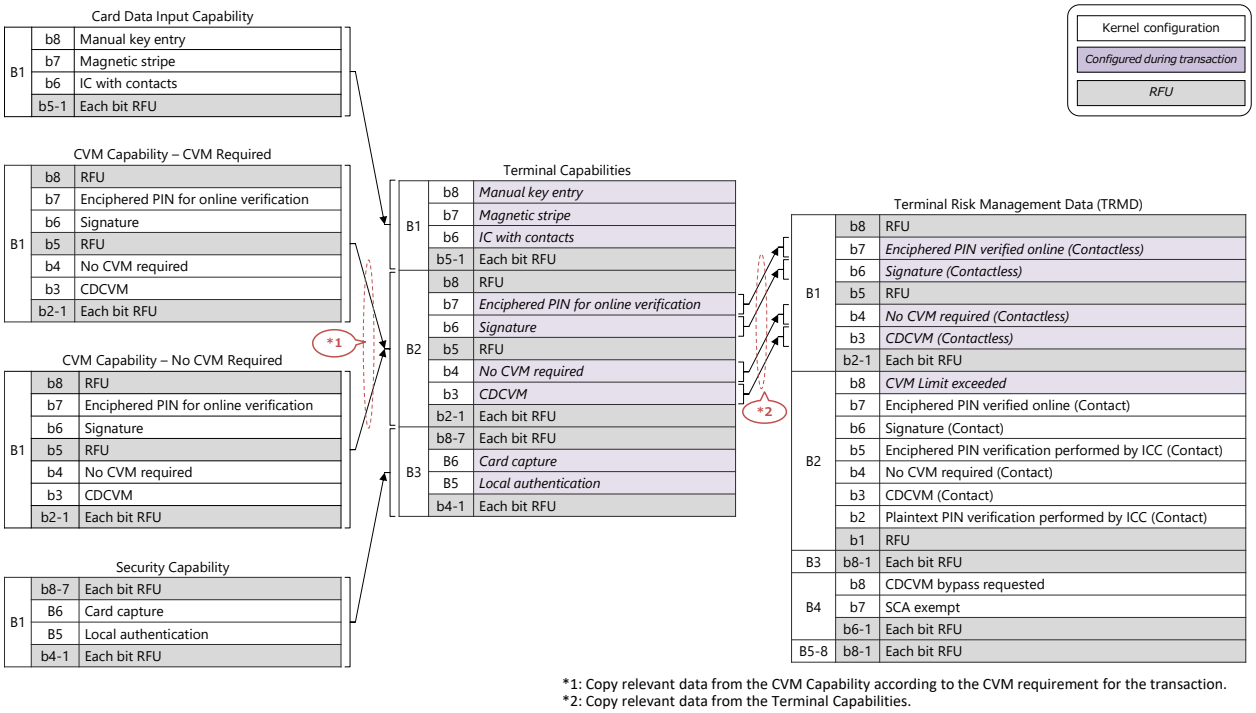
3.3 Cardholder Verification

3.3.1 Introduction

Cardholder Verification Methods (CVMs) are used to verify the cardholder's identity during a transaction. They can vary from requiring a PIN entry to obtaining a signature or even allowing transactions without any verification (No CVM).

Kernel 8 conveys information about available CVMs in Terminal Risk Management Data (TRMD); the Card includes TRMD in its CDOL1 or PDOL in order to receive this information. TRMD is defined in section 'Terminal Risk Management Data' in [EMV Book C-8] Annex A, and illustrated in Figure 3.1.

Figure 3.1—Terminal Risk Management Data (TRMD)



The Kernel sends the TRMD to the Card during the GENERATE AC command. The Card then selects a mutually supported CVM from the provided list and communicates its choice (or communicates that it was unable to find a mutually supported CVM) back to the Kernel in the GENERATE AC response.

CVM limits are used to determine which CVMs are offered to the Card by the Kernel. Terminal Risk Management Data (TRMD) is used by the Kernel to specify which CVMs are available for use in a particular transaction (as defined in the section ‘Terminal Risk Management Data’ in [EMV Book C-8] Annex A). The TRMD contains information about various terminal risk parameters which help determine which CVMs can be offered to the Card during a transaction based on the risk level associated with the transaction and other factors.

The Card, upon receiving the list of available CVMs, selects an appropriate method based on its capabilities and security measures and communicates this choice back to the Kernel to proceed with the transaction using the selected CVM.

The process of personalising the supported CVMs on a payment card and determining the suitable CVM to use, considering the capabilities of both the Kernel and the Card, is unique to each payment system. Therefore, the specifics of this customisation and the decision-making logic behind selecting the appropriate CVM fall outside the scope of the present document.

This process ensures that the most suitable and secure method of cardholder verification is employed based on the transaction circumstances, the Card and the Kernel’s capabilities.

TRMD includes three additional bits related to Cardholder Verification Methods (CVMs), as detailed in the section ‘Terminal Risk Management Data’ in [EMV Book C-8] Annex A.

- ‘CVM Limit exceeded’: This bit explicitly signals to the Card that the transaction amount exceeds the limit set by the merchant, necessitating a CVM for authorisation.
- ‘CDCVM bypass requested’: This bit corresponds to specific business rules, indicating scenarios such as requests from transit terminals to bypass Consumer Device Cardholder Verification Methods (CDCVM) for operational or compliance purposes.
- ‘SCA exempt’: This bit relates to regulatory requirements or business rules, specifying instances where the transaction might be excluded from the Card’s lost and stolen protection or could be exempt from Strong Customer Authentication (SCA) mandates.

These three TRMD bits may not be used by every payment system. When used, these bits conveyed through TRMD provide additional transaction information to the Card, such as surpassing merchant-defined CVM limits, special terminal requests (like CDCVM bypass for transit situations), and considerations for regulatory compliance or exceptions from particular authentication requirements. Understanding these bits aids the Card in making informed decisions about CVM selection and handling transactions based on the provided context and conditions.

Based on the CVMs offered by the Kernel (if any; the Kernel may return ‘N/A’, indicating no preference, as described in [EMV Book A] section 5.5.8) and on the business rules (for both, see section ‘Terminal Risk Management Data’ in [EMV Book C-8] Annex A), the Card then selects a CVM. This chosen CVM is communicated in the Cardholder Verification Decision data object within the GENERATE AC response.

If the Card cannot or does not want to use any available CVM method, it may indicate ‘CV FAILED’ in the Cardholder Verification Decision data object. This implies that the cardholder verification wasn’t successful or couldn’t be performed for this transaction. Upon encountering ‘CV FAILED’, the Terminal responds by setting the TVR bit ‘Cardholder verification was not successful’. This action allows the Kernel to potentially decline the transaction as per the configured rules defined by the Terminal Action Code – Denial data object (see section ‘Terminal Action Code – Denial’ in [EMV Book C-8] Annex A) during the subsequent Terminal Action Analysis.

3.4 Card and Data Authentication

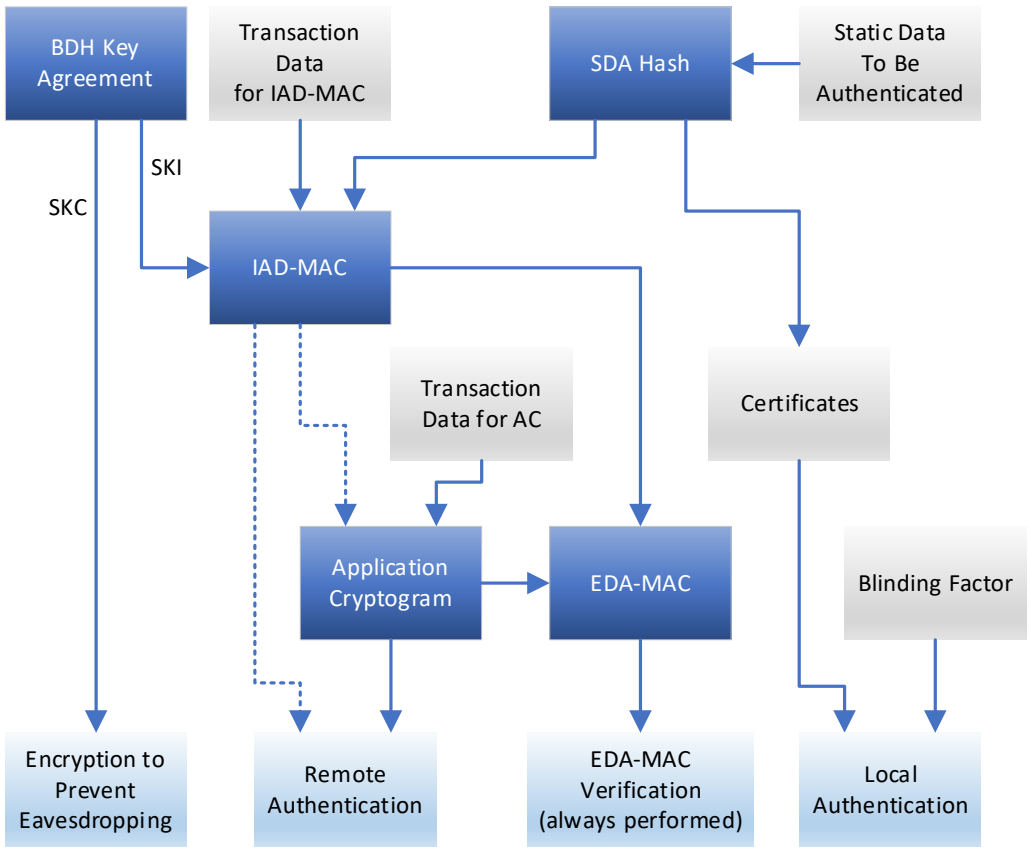
3.4.1 Introduction

Figure 3.2 provides a high-level look at card and data authentication in Kernel 8.

Several data objects created for use during authentication are shown in the figure and described in section 3.4.2.

Section 3.4.3 discusses the differences between the local and remote authentication methods.

Figure 3.2—Card and Data Authentication



BDH:	Blinded Diffie-Hellman
SKC:	AES Session Key for Confidentiality
SKI:	AES Session Key for Integrity
SDA Hash:	Hash over static data
MAC:	Message Authentication Code
IAD-MAC:	Issuer Application Data MAC
EDA-MAC:	Enhanced Data Authentication MAC

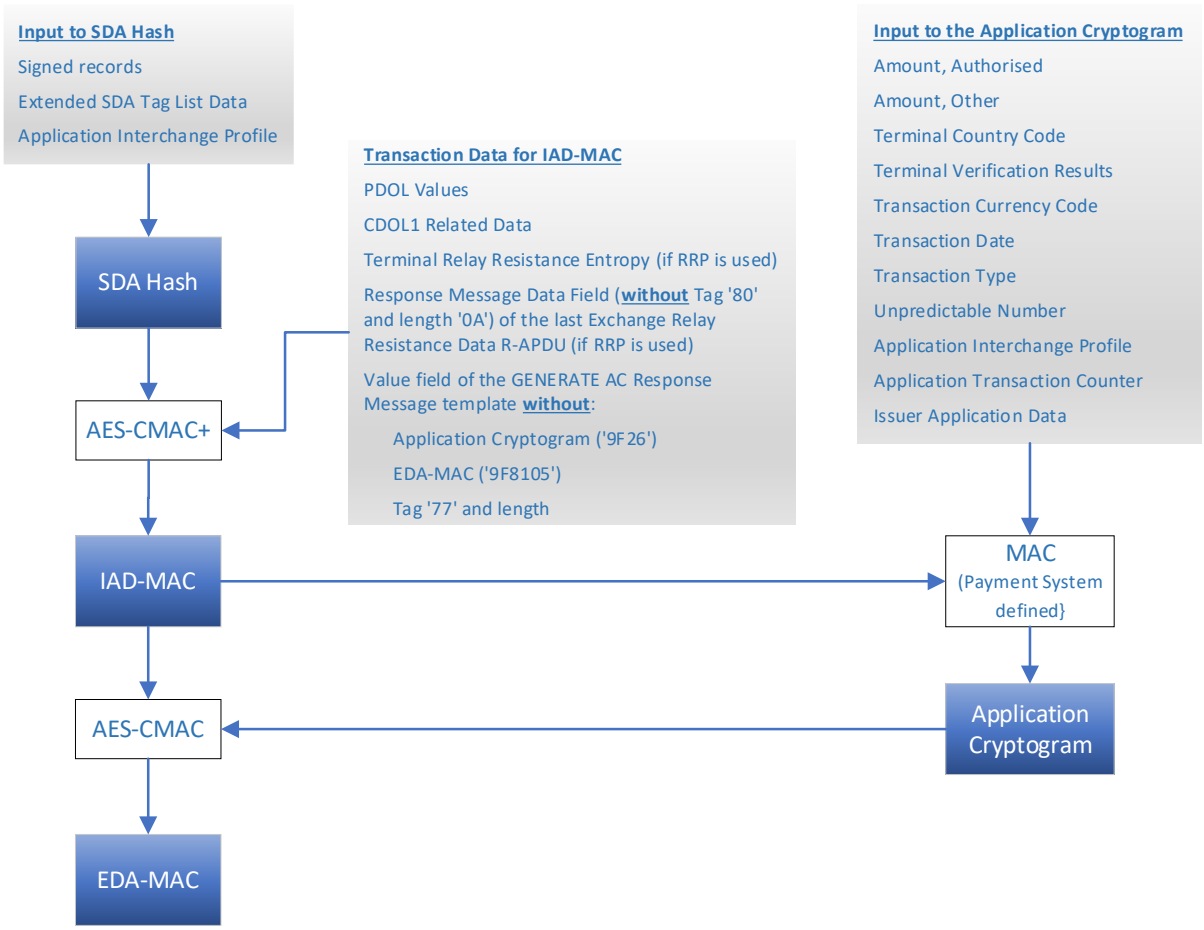
3.4.2 Authentication-related Data Objects

The Card and the Kernel separately create each of the following data objects:

- SDA Hash
- Issuer Application Data MAC (IAD-MAC)
- Enhanced Data Authentication MAC (EDA-MAC)

The Card-generated EDA-MAC and SDA Hash are compared to the Kernel-generated EDA-MAC and SDA Hash, to confirm the integrity of data exchanged between the Card and the Kernel.

Figure 3.3—Creation of IAD-MAC and EDA-MAC



3.4.2.1 SDA Hash

The SDA Hash is used to authenticate the static data provided by the Card. Generated separately by the Card and the Kernel using the SHA-256 hashing algorithm, the SDA Hash is a hash over the concatenation of:

- the signed records
- the Extended SDA Tag List Data

  Data objects identified by the Extended SDA Tag List; these data objects are not in records; examples include the AID and PDOL.
- the Application Interchange Profile

The SDA Hash is used as input for the Issuer Application Data MAC (IAD-MAC; see section 3.4.2.2).

3.4.2.2 Issuer Application Data MAC (IAD-MAC)

The IAD-MAC is an 8-byte AES-CMAC+-based authentication code generated separately by the Card and the Kernel over selected transaction data, using the Session Key for Integrity (SKI), as detailed in [EMV Book E].

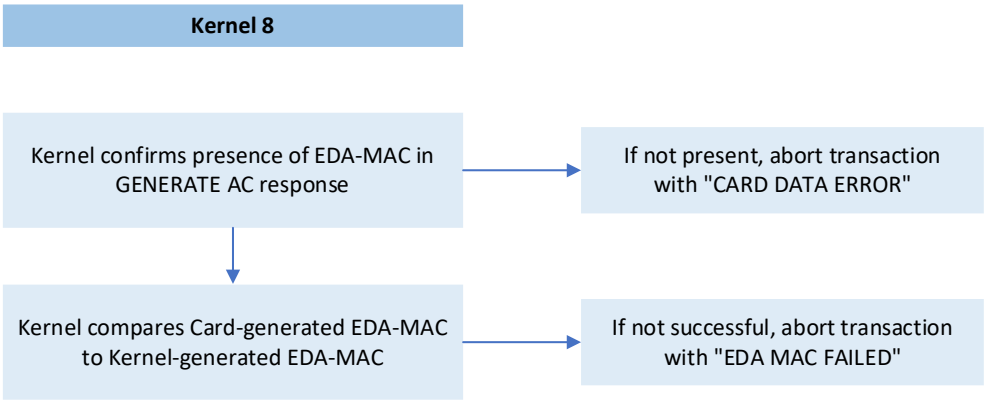
The IAD-MAC is used as input for the Enhanced Data Authentication MAC (EDA-MAC; see section 3.4.2.3), may be used as input for the Application Cryptogram, and may be provided to the issuer host during Remote Authentication.

3.4.2.3 Enhanced Data Authentication MAC (EDA-MAC)

The EDA-MAC is an 8-byte AES-CMAC generated separately by the Card and the Kernel over transaction data including the Issuer Application Data, the card static data, and the Application Cryptogram, using the Session Key for Integrity (SKI), as detailed in [EMV Book E]. The legitimacy of the session key is ensured through the validation of the Issuer Public Key Certificate, the ICC Public Key Certificate, and the blinding factor.

The Card returns its EDA-MAC in the GENERATE AC response. If EDA-MAC (Card) is absent, the Kernel aborts the transaction with Error Indication: CARD DATA ERROR. The Kernel generates its own EDA-MAC and compares it to the EDA-MAC returned by the Card. If EDA-MAC verification fails, the Kernel aborts the transaction with Error Indication: EDA-MAC FAILED.

Figure 3.4—Kernel Verification of EDA-MAC



Note that the Kernel verifies the EDA-MAC regardless of whether local authentication is performed.

3.4.3 Local vs. Remote Authentication

There are two types of card and data authentication:

- Local Authentication

  During local authentication, the Kernel verifies the Issuer Public Key Certificate, the ICC Public Key Certificate, and the blinding factor.

  Local authentication is performed only if supported by both the Kernel and the Card; i.e. if ‘Local authentication’ in Terminal Capabilities is set and if ‘Local authentication supported’ in Application Interchange Profile is set.

- Remote Authentication

  For remote authentication, the Card generates an Application Cryptogram (AC) using data objects specified by the Card’s payment system, possibly including the IAD-MAC. The Kernel generates its own IAD-MAC and, if requested, sends it to the issuer. The issuer does not receive the Card’s IAD-MAC, so if the IAD-MAC was used to generate the AC, the issuer uses the Kernel’s IAD-MAC to validate the AC.

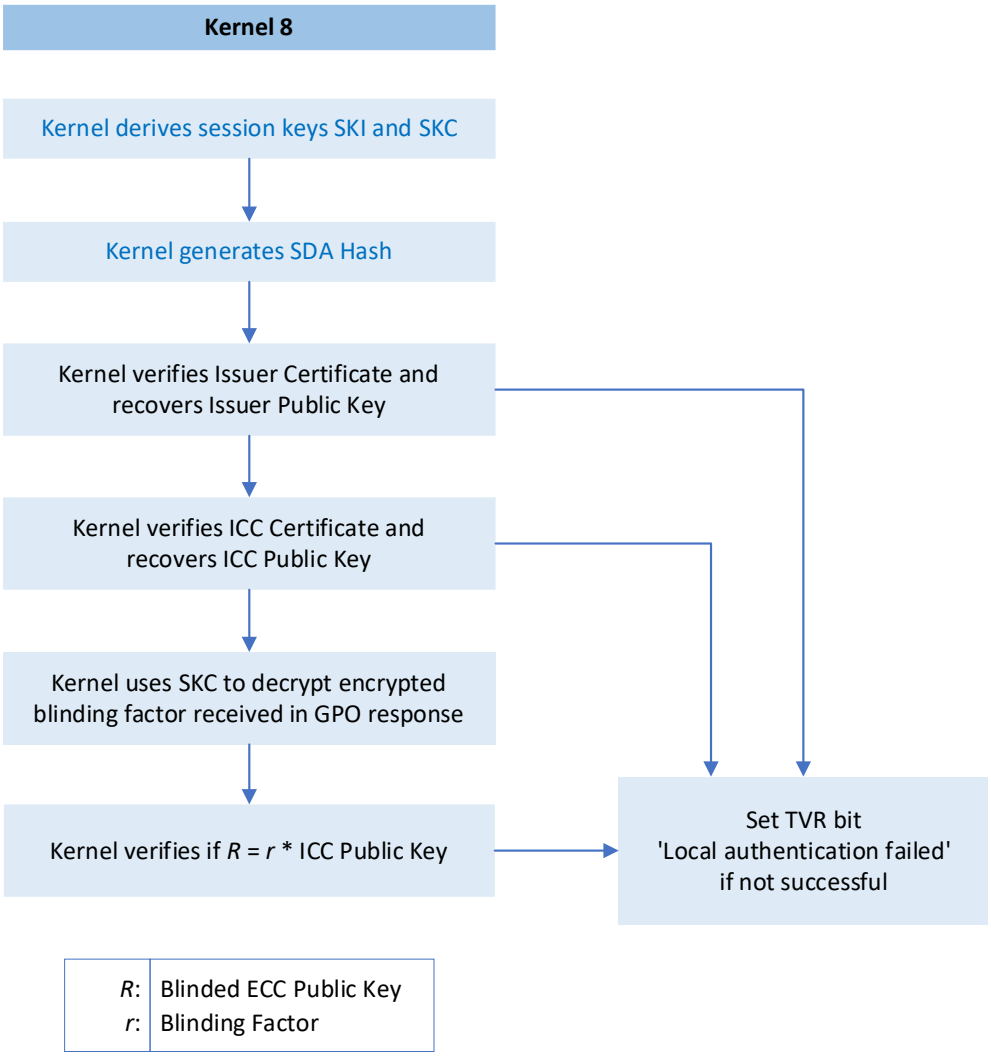
  Remote authentication is performed for every transaction.

Note that the Kernel verifies the EDA-MAC as described in section 3.4.2.3 regardless of whether local authentication is performed.

3.4.3.1 Local Authentication

During local authentication, the Kernel verifies the Issuer Public Key Certificate, the ICC Public Key Certificate, and the blinding factor.

Figure 3.5—Local Authentication



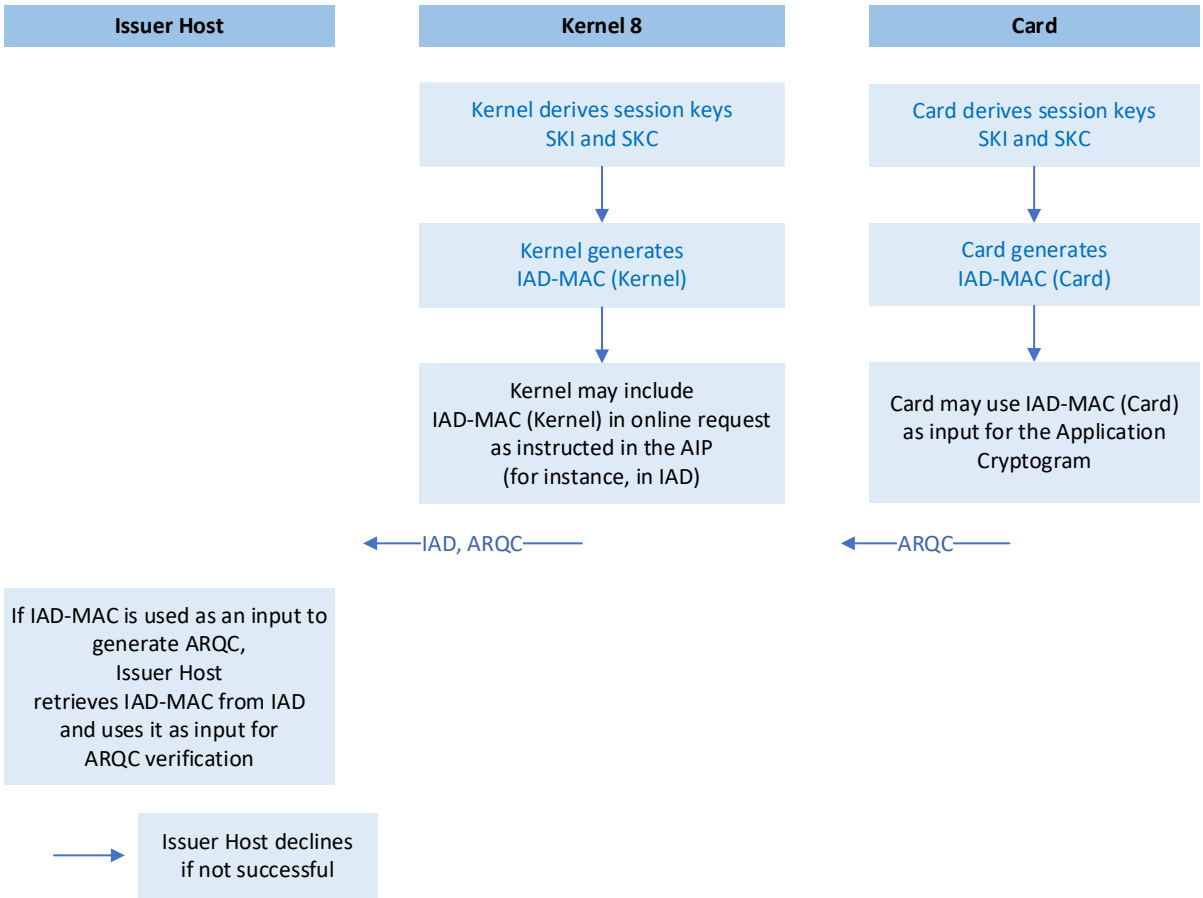
3.4.3.2 Remote Authentication

For remote authentication, the Card generates an Application Cryptogram (AC) using data objects specified by the Card's payment system; one of the specified data objects may be the IAD-MAC.

While processing the transaction, the Kernel generates its own IAD-MAC. If indicated in the Application Interchange Profile (AIP) provided by the Card, the Kernel sends its IAD-MAC to the issuer along with the AC and the authorisation message, as a part of Issuer Application Data (IAD).

If the IAD-MAC was used when generating the AC, the issuer will need the IAD-MAC to validate the AC. Using the Kernel-generated IAD-MAC (rather than the Card-generated one) to validate the AC indirectly confirms the integrity of the Card-generated IAD-MAC, adding an extra layer of security to the transaction.

Figure 3.6—Remote Authentication



3.5 Data Storage

3.5.1 Introduction

The Implementation Option called Data Storage in Kernel 8 is designed to provide a mechanism for storing data on a Card and reading data from the Card using dedicated commands, namely WRITE DATA and READ DATA. Unlike traditional payment system tags, Data Storage introduces a set of payment system tags ranging from '9F8111' to '9F811A' specifically for handling non-payment data. These tags correspond to Data Envelope 1 through 10, allowing explicit reading and writing of this data within the payment system.

The purpose of these dedicated commands and tags is to facilitate the secure storage and retrieval of non-payment-related information on the Card, enhancing the functionality and versatility of the payment system beyond standard transaction-related data. This feature enables the storage and management of additional information or specific data sets that may not be directly associated with payment transactions but are still relevant to the Card's usage or functionality.

Note: Within the Kernel's operations, problems with Data Storage, particularly during READ DATA or WRITE DATA processes, are distinct from financial transactions. Therefore, should the Kernel encounter any challenges related to Data Storage during these procedures, it won't disrupt the ongoing financial transaction.

To clarify, issues pertaining to Data Storage won't trigger the Kernel to halt or cancel the financial transaction; rather, the transaction proceeds independently, disregarding any issues that occur within the Data Storage phase.

The payment systems may decide not to implement Data Storage in their Cards. The absence of support for Data Storage in Cards does not create any financial transaction processing issues.

3.5.2 Data Envelopes

Data envelopes are tags used within a payment system that contain proprietary information from various entities involved, such as the acquirer, payment system, or third parties. In Kernel 8, there is support for ten payment system tags identified by the tags '9F8111' to '9F811A'.

These data envelopes can be retrieved using the READ DATA command and updated using the WRITE DATA command. The length of the data within these envelopes is variable, with a maximum size of 243 bytes.

3.5.3 Read Data

The READ DATA command serves to retrieve a Data Envelope containing specific information from a Card in an EMV transaction. ‘Tags To Read’ is an optional configuration data object containing a list of tags indicating the data envelopes the Terminal has requested to be read. ‘Tags To Read’ can also be populated by the Terminal using the ACT Signal. Before executing the GENERATE AC command, depending upon the list of data envelopes in ‘Tags to Read’, the Kernel may issue several READ DATA commands, each with distinct Enhanced Data Storage-related payment system tags ('9F8111' to '9F811A'). These tags represent various data elements stored on the Card.

The purpose of these READ DATA commands is to extract necessary information from the Card. If the Data Envelope being retrieved lacks data for a particular payment system tag, the Card will respond with that tag and a zero-length data field, indicating the absence of the requested information.

3.5.4 Write Data

The WRITE DATA command is employed by the Kernel to save information on the Card following the issuance of the GENERATE AC command during a transaction. ‘Data Envelopes To Write’ is an optional configuration data object containing Terminal data writing requests to be sent to the Card. The value of ‘Data Envelopes To Write’ is composed of a series of TLVs. ‘Data Envelopes to Write’ can also be populated by the Terminal using the ACT Signal. After the GENERATE AC command, depending upon the value of ‘Data Envelopes to Write’, the Kernel may direct one or more WRITE DATA commands to the Card, each containing a Data Envelope.

However, if local authentication fails, the Kernel will bypass WRITE DATA processing as a precautionary measure.

3.5.5 Data Integrity

The communication protocol between the Card and the Kernel for Data Storage prioritises data security by employing strict security measures.

Information transmitted to and from the Card is safeguarded using a dedicated privacy mechanism. This ensures the protection of sensitive card data and extends the same level of security to information sent back to the Kernel through READ RECORD commands.

Using a method that generates a MAC with the Session Key for Integrity, data fetched from the Card using a READ DATA command is shielded. The Card generates the MAC over the encrypted data envelope and appends the MAC at the end of the encrypted data envelope. When the Kernel receives the data, it first verifies that it is received unchanged by generating the MAC over the received encrypted data envelope and comparing it with the received MAC. This ensures that the received data is not altered during transfer between the Card and the Terminal.

When the Kernel writes data to the Card via the WRITE DATA command, the Card responds with a MAC computed using the Session Key for Integrity over the recovered plaintext. The Kernel verifies the MAC received from the Card with the MAC generated by the Kernel. This process assures the Kernel that the data was received and stored on the Card without unauthorised alterations.

In summary, this protocol emphasizes the security of data privacy and integrity through specific measures that protect the confidentiality of information and ensure its authenticity during transmission and storage between the Card and the Kernel.

3.6 Cloud Optimisation

The Kernel 8 specification offers the opportunity to build implementations that support cloud-based kernel architectures, in which the kernel processing is distributed between a thin client in a local reader and a cloud-based server application.

Such an implementation places a minimum amount of functionality in the client and minimises or eliminates any interaction between the client and cloud server until the Card has been removed from the reader field. In particular, the Kernel 8 specification allows the resource-intensive cryptographic functions to be cloud-based, along with all the functions that are performed after the Card has been removed.

All APDU commands between the reader and Card must operate as quickly as possible, thus requiring that they are kept in the reader client. That is, the commands and processing steps for Application Selection, GPO Processing, Relay Resistance Protocol, Read Records, and Generate AC remain on the client side.

A few other simple and quick functions that are required before the Card can be removed from the reader field must be performed in the client in order to avoid client-server interactions that could impact the 500ms card-in-field processing requirement. These functions are Algorithm Suite Negotiation, Terminal Risk Management, and the 1st Terminal Action Analysis.

All other functions can be performed on the cloud server. This includes all the cryptographic functions such as Ephemeral ECC Key generation, BDH Key Agreement, Record Decryption, IAD-MAC and EDA-MAC Processing, and Process Certificates. By allowing these functions to be performed after the Card has left the reader field, they can be concentrated together rather than being strictly performed in the logical order of the end-to-end transaction flow.

Other functions performed after the Card has been removed from the reader field can also be moved to the server side. That is, CVM & Card TVR Processing, the 2nd Terminal Action Analysis, and the preparation of the final transaction outcome.

The Kernel 8 specification allows for the above regrouping and technical separation of processing steps by nature of the availability of the data necessary to perform the Kernel-side functions after the Card has been removed. For example, the BDH Key Agreement could theoretically be performed as soon as the Card returns the card blinded public key and the encrypted blinding factor in the GPO response, but because the Kernel doesn't immediately require the AES Session Keys for Confidentiality and Integrity, these data objects can be passed to the cloud server to be used further downstream in order to generate the Kernel-side Session Keys at the point they are needed, and with no impact on the 500ms card-in-field processing requirement.

3.7 Processing Restrictions

3.7.1 Introduction

Kernel 8, unlike earlier Kernels, does not perform Processing Restrictions. Optionally, the Card may provide this functionality during GENERATE AC command processing.

This phase involves the transmission of essential terminal data objects from the Kernel to the Card as requested by the Card using the CDOL1 in the data field of the GENERATE AC command.

Once the Card receives this data, it undertakes various internal processes, including the assessment of any Processing Restrictions set by the card issuer. These restrictions might include limitations or conditions on transaction types, currencies, or other predefined criteria.

Following the evaluation, the Card provides feedback on the Processing Restrictions it encountered through the Card TVR. This Card TVR is conveyed within the response sent back to the Terminal following the execution of the GENERATE AC command. It contains specific bits reflecting the outcome of the transaction, including those directly impacted by Processing Restrictions. Sections 3.7.2 through 3.7.5 discuss selected Card TVR bits.

Note that the exact nature of Processing Restrictions is dependent on the payment system and is out of scope of this document.

3.7.2 Application Version Number Checking

The TVR bit 'ICC and terminal have different application versions' highlights a discrepancy between the application versions used by the Card and the Terminal.

To check Application Version Number:

- The Card must have Application Version Number (Card) in its internal data store.
- The Card must include the tag of Application Version Number (Reader) in CDOL1 so that the Kernel will send the tag of Application Version Number (Reader) to the Card in the GENERATE AC command.

3.7.3 Application Effective Date Checking

The TVR bit ‘Application not yet effective’ indicates that the Card’s application hasn’t been activated or authorised for use.

To check Application Effective Date:

- The Card must have Application Effective Date (Tag '5F25') in its internal data store.
- The Card must include the tag of Transaction Date (Tag '9A') in CDOL1.

This verification process ensures that, if the Card supports Application Effective Date checking, it actively assesses whether the application is authorised for use based on the transaction date compared against the internally stored effective date. If the application hasn’t become effective by the transaction date, it might influence authorisation decisions or necessitate specific procedures as per the Card’s settings or issuer’s directives.

If the Transaction Date tag is missing from CDOL1 or if the Application Effective Date is not available or has not yet occurred, the Card will manage the TVR bit as per the processing required by the respective payment system.

3.7.4 Application Expiration Date Checking

When the Card supports Application Expiration Date checking, this check ensures that the application on the Card has not expired.

To perform this check, the following conditions must typically be met:

- The Card must have Application Expiration Date (Tag '5F24') in its internal data store.
- The Card must include the Transaction Date (Tag '9A') in CDOL1.

The Application Expiration Date is checked against the Transaction Date during the transaction to verify whether the application is still valid or has expired.

If the Transaction Date tag is missing from CDOL1 or if the Application Expiration Date is not available or has passed, the Card will manage the TVR bit ‘Expired application’ as per the processing required by the respective payment system.

3.7.5 Application Usage Control Checking

Application Usage Control checking involves verifying whether a requested service or transaction type is permitted based on the Terminal and the Card’s configuration.

If the Card supports Application Usage Control checking, then to perform this check:

- The Card must include specific tags in CDOL1, such as Terminal Type, Additional Terminal Capabilities, Transaction Type, Terminal Country Code, and Amount, Other (Numeric).

- The Card must have Application Usage Control in its internal data store. This data object defines which services or transaction types the Card is configured to accept or decline.
- The Card must have Issuer Country Code in its internal data store.

If any necessary tag is missing from CDOL1 or if Application Usage Control and/or Issuer Country Code is unavailable or is configured to disallow the requested service or transaction type, then the TVR bit 'Requested service not allowed for card product' may be set.

Note that specific implementations of these checks can differ among payment systems.

3.8 Updating TVR after GENERATE AC Command

3.8.1 Card TVR

As a result of Processing Restrictions and CVM processing, the Card may change the value of the Terminal Verification Results. For example:

- If the Card chooses Online PIN, the TVR bit 'Online CVM captured' needs to be set.
- If the Card is expired, then the TVR bit 'Expired application' may be set.

This means that the TVR value received by the Card during the GENERATE AC command might need adjustments due to these card conditions. Consequently, the Card may include a Card TVR in its GENERATE AC response, reflecting the TVR used to generate the Application Cryptogram.

3.8.2 TVR Bit 'Local Authentication Failed'

If local authentication is supported by the Card and the Kernel, then following the GENERATE AC response from the Card, the Kernel undertakes critical tasks, which include:

- Validation of the Issuer Public Key Certificate
- Validation of the ICC Public Key Certificate
- Validation of the blinding factor

Should any of these validations fail following the GENERATE AC command, the Kernel sets the TVR bit 'Local authentication failed'. This bit is crucial in determining the Kernel's decision.

Note that if the Kernel Configuration bit 'Report local authentication failed in TVR' is not set, then the TVR bit 'Local authentication failed' will be cleared before storing Terminal Verification Results in the Data Record. The value of the Kernel Configuration bit 'Report local authentication failed in TVR' is defined by individual payment systems, allowing them to determine this setting's behaviour.

This decision surrounding the TVR bit ‘Local authentication failed’ and its handling post-validation directly influences the decision-making process within the payment system, potentially affecting subsequent transaction authorisation steps based on AC verification.

3.8.3 Cardholder Verification-related Bits in TVR

During the GENERATE AC command processing, the Card may update the TVR bits ‘Cardholder verification was not successful’ or ‘Online CVM captured’. The Card also returns the data object Cardholder Verification Decision (Tag '9F8102') in the GENERATE AC response.

To make sure that the Cardholder Verification-related bits in the TVR are set correctly, the Kernel again sets the TVR bits ‘Cardholder verification was not successful’ or ‘Online CVM captured’ based upon the value of Cardholder Verification Decision returned by the Card in the GENERATE AC command. This ensures that the TVR accurately represents the outcome of cardholder verification for subsequent analysis by the Kernel during 2nd Terminal Action Analysis.

This correction process could lead to a situation where the TVR used by the Card to generate the Application Cryptogram differs from the TVR received by the Issuer.

3.8.4 Kernel Reserved TVR Mask

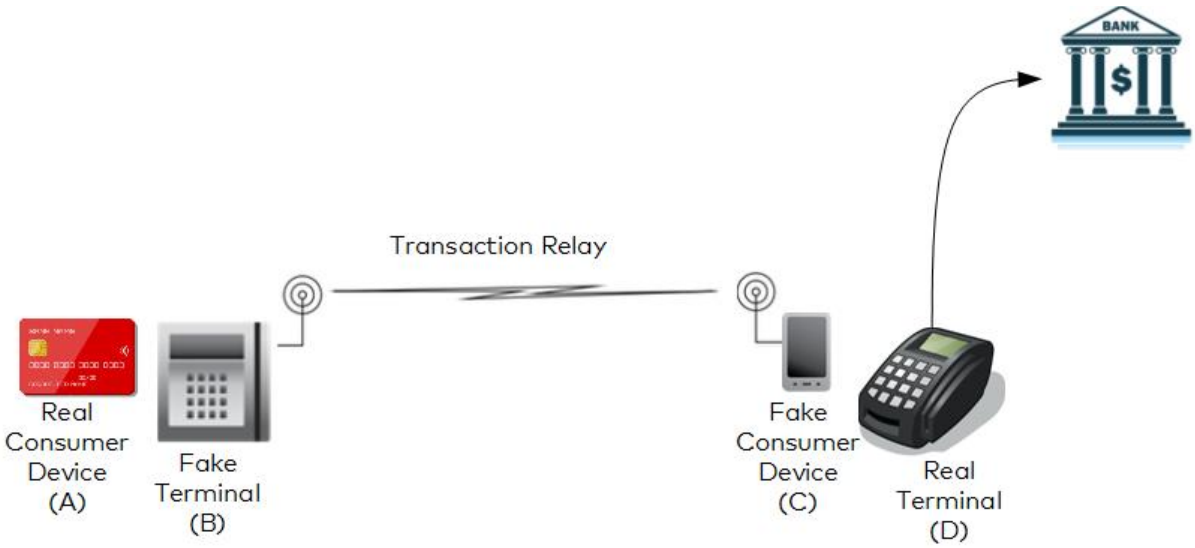
To safeguard against unauthorised modifications of the TVR, a Kernel Reserved TVR Mask is applied. This mask prevents the Card from altering specific TVR bits that reflect the Kernel’s decision-making. While updating the TVR with bits set in the Card TVR, the Kernel does not update bits that are marked as 1 in the Kernel Reserved TVR Mask. This ensures that crucial TVR bits, especially those representing the Kernel’s decisions or specific card conditions, remain unchanged by the Card and are accurately transmitted to the acquirer during transaction authorisation.

3.9 Relay Resistance Protocol (RRP)

3.9.1 Introduction

A relay attack occurs when a deceitful terminal deceives an unsuspecting cardholder into making a transaction, where the actual transaction is sent through a fraudulent card or simulator to the genuine terminal of an unsuspecting merchant. It can also happen when a fraudulent reader is used without the cardholder's knowledge of the transaction.

Figure 3.7—Relay Attack



To detect this type of attack, a specific command called EXCHANGE RELAY RESISTANCE DATA (ERRD) and a set of associated data elements are used. These measures enable terminals to identify potential relay fraud and report the findings to the Card and to the issuer, aiding in card risk management.

Relay Resistance Protocol (RRP) identifies delays introduced during attempts at relay fraud. This capability allows suspicious transactions to be examined in real-time or declined offline, based on the observed delays.

Note: While RRP provides some mitigation against basic relay attacks and makes it more difficult to relay a contactless transaction by introducing additional challenges, RRP cannot completely prevent a relay attack from occurring. Different factors may cause the time measured by the Kernel to vary, causing the Card and Kernel time limits and thresholds to be exceeded, even when a genuine Card is presented without relay. Issuers should balance the mitigation of relay attacks provided by RRP against the potential for 'false positives' and unnecessary declines.

3.9.2 Card RRP Timing Values

Cards are personalised with a minimum processing time (Min Time For Processing Relay Resistance APDU), a maximum processing time (Max Time For Processing Relay Resistance APDU), and a transmission time (Device Estimated Transmission Time For Relay Resistance R-APDU).

Table 3.1 provides guidelines for setting RRP-related personalisation values in the Card.

Table 3.1—Card RRP Personalisation Values

Parameter	Value
Min Time For Processing Relay Resistance APDU	This 2-byte binary value represents, in units of hundreds of microseconds, the minimum time for processing the ERRD command by the Card. The value must be set such that it is not possible to obtain a response from the Card faster than this minimum time under any circumstances. A typical value might be 50 ('0032') representing 5ms.
Max Time For Processing Relay Resistance APDU	This 2-byte binary value represents, in units of hundreds of microseconds, the maximum time for processing the ERRD command by the Card. A typical value might be 100 ('0064') representing 10ms.
Device Estimated Transmission Time For Relay Resistance R-APDU	This 2-byte binary value represents, in units of hundreds of microseconds, the time taken to send the R-APDU. The transmission time should be set to the slowest time expected under normal operating conditions so that any faster transmission (e.g. using Type A rather than Type B) does not extend the maximum processing time that will be measured. A typical value might be 15 ('000F') representing 1.5ms.

The actual values for the data objects specified in Table 3.1 will be provided by the card manufacturers and may vary for different card products.

3.9.3 Terminal RRP Timing Values

Table 3.2 and Table 3.3 provide guidelines for setting RRP-related configuration values in the Terminal.

Table 3.2—Payment System-related RRP Configuration Values

Parameter	Value
Maximum Relay Resistance Grace Period	Represents, in units of hundred of microseconds, how far outside the window defined by the Card the measured time may be and still be considered acceptable.
Minimum Relay Resistance Grace Period	Represents, in units of hundreds of microseconds, how far outside the window defined by the Card the measured time may be and still be considered acceptable.
Relay Resistance Accuracy Threshold	Represents, in units of hundreds of microseconds, the threshold above which the Kernel considers the variation between Measured Relay Resistance Processing Time and Min Time For Processing Relay Resistance APDU unacceptable. A typical value might be 200 ('00C8') representing 20ms.
Relay Resistance Transmission Time Mismatch Threshold	A percentage, expressed as an integer, that represents the threshold above which the Kernel considers the variation between Device Estimated Transmission Time For Relay Resistance R-APDU and Terminal Expected Transmission Time For Relay Resistance R-APDU unacceptable. A typical value might be 50, representing 50%.

The data objects outlined in Table 3.2 are subject to variation and the values will be furnished by the payment system.

Table 3.3—Terminal RRP Configuration Values

Parameter	Value
Terminal Expected Transmission Time For Relay Resistance C-APDU	Represents the time, in units of hundreds of microseconds, that the Kernel expects to need for transmitting the EXCHANGE RELAY RESISTANCE DATA command to the Card.
Terminal Expected Transmission Time For Relay Resistance R-APDU	Represents the time, in units of hundreds of microseconds, that the Kernel expects that the Card will need for transmitting the EXCHANGE RELAY RESISTANCE DATA R-APDU.

The information presented in Table 3.3 will be the same across all AIDs and transaction types. However, the values may be different based upon the terminal manufacturers. The actual values will be configured under the guidance of the terminal manufacturers.

3.9.4 RRP Risk Management

3.9.4.1 Kernel Processing

The Terminal sets selected TVR bits according to the rules described below:

- If the Measured Relay Resistance Processing Time computed by the Terminal is greater than the Max Time For Processing Relay Resistance APDU provided by the Card plus the Maximum Relay Resistance Grace Period set in the Terminal, then the TVR bit ‘Relay Resistance Time Limits Exceeded’ is set.
- If the Measured Relay Resistance Processing Time computed by the Terminal is less than the Min Time For Processing Relay Resistance APDU provided by the Card minus the Minimum Relay Resistance Grace Period set in the Terminal, then the Terminal terminates the transaction.
- If any of the following are true:
  - (Device Estimated Transmission Time For Relay Resistance R-APDU divided by Terminal Expected Transmission Time For Relay Resistance R-APDU) * 100 < Relay Resistance Transmission Time Mismatch Threshold, or
  - (Device Estimated Terminal Expected Transmission Time For Relay Resistance R-APDU divided by Transmission Time For Relay Resistance R-APDU) * 100 < Relay Resistance Transmission Time Mismatch Threshold, or
  - (Measured Relay Resistance Processing time minus Min Time for Processing Relay Resistance APDU) > Relay Resistance Accuracy Threshold

  then the TVR bit ‘Relay Resistance Threshold Exceeded’ is set.

- If RRP is performed, then it will typically be indicated by setting TVR bit ‘RRP PERFORMED’; otherwise, typically TVR bit ‘RRP NOT PERFORMED’ is set.

3.9.4.2 Card Processing

As part of the GENERATE AC command processing and if an ERRD command has been received (RRP Counter > 0), the Card compares the Terminal Relay Resistance Entropy value received in the ERRD command with the Unpredictable Number value received in the GENERATE AC command. If they do not match, the card application will decline the transaction with an AAC.

3.9.5 RRP Counter

The card application will allow the Terminal to send the EXCHANGE RELAY RESISTANCE DATA C-APDU a maximum of three times.

The card application sets the RRP Counter to zero while processing the GET PROCESSING OPTIONS command, then uses the RRP Counter to count the number of subsequent ERRD commands sent by the Terminal. The RRP Counter is incremented after receiving the EXCHANGE RELAY RESISTANCE DATA command and before selecting the Device Relay Resistance Entropy.

3.10 Tag Mapping

3.10.1 Introduction

Payment systems may define their own tags that differ from the Kernel 8 definitions. Therefore, Kernel 8 introduces a feature that allows the conversion of data element tags in the outcome, based on the payment system configurations. This can also be used to send 2-byte tag data elements to payment systems where 3-byte tags are not defined or used, by mapping a 2-byte tag to a corresponding 3-byte tag data element.

3.10.2 Tag Mapping List

Tag mapping is performed on data elements in the Data Record or Discretionary Data that are passed from the Kernel to the reader. The tags in the Tag Mapping List are ordered in pairs of two tags: { Tag1 MappedTag1 Tag2 MappedTag2 ... Tag*n* MappedTag*n* }.

For example, define {'9F8106', '9F60'} in the Tag Mapping List. Then the Kernel can send the Card’s Authenticated Application Data '9F8106' to the reader as Authenticated Application Data '9F60'.

4 Configuration

4.1 Databases

The Terminal must have a TLV Database (as discussed in [EMV Book C-8] section A.2) and an Internal Data Store (as discussed in [EMV Book C-8] section 7.1).

4.2 Proprietary Tags

Proprietary tags refer to data elements within a Card or Terminal that are introduced by the payment systems for their specific purposes. These tags are not part of the standardised EMV specifications but are included as extensions by the payment systems to cater to additional functionalities, features, or unique data requirements within their payment systems.

Any data object not listed in [EMV Book C-8] section A.2 is considered a proprietary tag within the context of this document. Therefore, these tags vary in their definition and usage, as they are specific to individual Application Identifiers (AIDs) and transaction types.

For each proprietary tag, certain details need to be defined, including:

- Access Conditions: Describing under what circumstances or conditions the data within the proprietary tag can be accessed or modified
- Format: Specifying the structure or format in which the data within the proprietary tag is stored or transmitted. This might include details about data types, encoding, or specific arrangements.
- Length Range: Defining the acceptable range of lengths for the data within the proprietary tag. This ensures that the data conforms to a predetermined size or range.

While [EMV Book C-8] outlines the necessity of defining access conditions, format, and length range for proprietary tags, the actual implementation and definition of these tags are left to the discretion of the implementer. Therefore, how these proprietary tags and their associated properties are precisely defined and used can vary between implementations.

4.3 Configuration Data Objects

[EMV Book C-8] section A.3 lists all configuration data objects. Configuration data objects are classified into the following types:

- Mandatory Configuration Data Objects: These essential data objects are required for a transaction to proceed. If a mandatory configuration data object is missing from the Configuration Data, the transaction will be aborted, indicating that the necessary data is not available to proceed.
- Optional Configuration Data Objects: These data objects are not essential for the transaction to proceed but might be used or expected under certain conditions. If an optional configuration data object is not present in the Configuration Data, it will also not be present in the Kernel's TLV database. In other words, a check for its presence will return FALSE.

Configuration data objects may have different values based on factors such as Application Identifier and Transaction Type. Thus, specific values associated with these configuration data objects depend on the context of the transaction.

Some configuration data objects will have update conditions that define which source is permitted to update the data object. These objects might not be initially present in Configuration Data but can be sent to the Kernel (part of the system processing transactions) either at the start of the Kernel or during transaction processing.

4.4 Certificates

[EMV Book E] describes the Public Key Infrastructure (PKI) that supports the local authentication of an EMV transaction. Every Card is personalised by the Issuer with a unique private/public key pair where the public key is certified by the Issuer.

A three-layer authentication scheme allows the Reader to validate the Local Cryptogram (EDA-MAC) generated by the Card:

- The Issuer public key is authenticated by the Certification Authority (CA) public key stored in the Reader.
- The ICC public key is authenticated by the Issuer public key which is signed by the Certification Authority in the Issuer certificate. The Card authentication includes the authentication of the blinding factor which is sent encrypted by the Card.
- The EDA-MAC is authenticated by the Reader using the Session Key for Integrity obtained during the BDH key agreement (see section 3.1.2).

If the validation of the EDA-MAC and the Issuer and Card certificates (including the blinding factor validation) is successful, the local authentication is successful, and the Card is considered as genuine.

Issuer and Card certificates are personalised in the Card by the Issuer and transmitted by the Card to the Reader if local authentication is supported. The Card typically holds a set of ECC certificates used to authenticate the Card ECC public key.

For migration purposes, it is possible to use instead a set of RSA certificates where the Card RSA certificate is used to authenticate the Card ECC public key. In that case the Card ECC public key must be added to the Card static data and should be signed by the Issuer.

Note that if ECC is used for Issuer certificates, the Card certificates must also use ECC. Similarly, if RSA is used for Issuer certificates, the Card certificates must also use RSA and the ICC ECC Public Key is stored as a separate data object.

4.4.1 RSA Certificates

In this document the term *RSA certificate* consists of the recoverable certificate plaintext, certificate-related data, and certificate signature.

Note that support for RSA certificates is a kernel Configuration Option and up to each payment system.

In an RSA signature, the length of the certificate signature is equal to the length of the public key modulus of the signer. In order to sign a certificate of an arbitrary length, the certificate plaintext and certificate-related data are hashed, then the certificate plaintext and the resulting hash are signed with the signer's private key.

The certificate signature is verified with the signer's public key so that the certificate plaintext and the hash value are recovered from the certificate signature. The certificate-related data is required to verify the hash value and thereby the certificate signature.

The formats of the Issuer RSA Public Key Certificate and ICC RSA Public Key Certificate, and the process to validate these certificates are explained in [EMV Book E].

CA public keys can be shared between AIDs that have the same RID. For RSA CA Public Keys, the Terminal should be able to store the information for at least six RSA CA Public Keys per RID.

Table 4.1 specifies the fields that may be used to store RSA CA Public Key in the Terminal's database. However, the exact details to store RSA CA Public Keys in the Terminal's database are left to the implementer of Kernel 8.

Table 4.1—RSA CA Public Key Data

Field Name
CA Public Key Index
CA Hash Algorithm Indicator
CA Public Key Algorithm Indicator
CA Public Key Modulus
CA Public Key Exponent
CA Public Key Checksum

4.4.2 ECC Certificates

An ECC certificate consists of the certificate plaintext and the certificate signature.

In an ECSDSA signature[1] (as described in [EMV Book E]), the certificate plaintext of an arbitrary length is hashed, then the resulting hash is signed with the signer's private key. The length of the certificate signature (N_{SIG}) is equal to the length of the hash value (N_{HASH}) plus the length of an element of the curve field (N_{FIELD}).

The ECSDSA signature is verified with the signer's public key. The entire certificate plaintext is required to verify the hash value and thereby the certificate signature.

The formats of the Issuer ECC Public Key Certificate and ICC ECC Public Key Certificate, and the process to validate these certificates are explained in [EMV Book E].

CA public keys can be shared between AIDs that have the same RID. For ECC CA Public Keys, the Terminal needs to support ten ECC CA Public Keys per RID.

Table 4.2 specifies the fields that may be used to store ECC CA Public Keys in the Terminal's database. However, the exact details to store ECC CA Public Keys in the Terminal's database are left to the implementer of Kernel 8.

Table 4.2—ECC CA Public Key Data

Field Name
CA Public Key Index
CA Public Key Algorithm Suite Indicator
CA Public Key
CA Public Key Checksum

[1] The ECSDSA signature is an ECC version of the Schnorr digital signature scheme with appendix using a hash algorithm according to [ISO/IEC 14888-3]. This is the optimised version where only the *x*-coordinate is hashed rather than the (*x, y*) coordinates.

4.4.3 Certification Revocation List

A CA Public Key Certificate Revocation List (CRL) is a list of digital certificates that have been revoked or are no longer considered valid before their expiration date. CRLs are issued and maintained by Certificate Authorities to inform users and relying parties about certificates that should not be trusted.

Table 4.3 specifies the fields that may be used to store Certification Revocation List in the Terminal's database. However, the exact details to store Certification Revocation List in the Terminal's database are left to the implementer of Kernel 8.

Table 4.3—Certification Revocation List

Field Name
CA Public Key Index
Certificate Serial Number
Additional Data (optional)

Annex A Abbreviations

Table A.1 lists abbreviations that are used in this document. For information on terms used in this document, see section 1.4.

Table A.1—Abbreviations

Abbreviation	Description
AAC	Application Authentication Cryptogram
AC	Application Cryptogram
AES	Advanced Encryption Standard
AFL	Application File Locator
AID	Application Identifier
AIP	Application Interchange Profile
APDU	Application Protocol Data Unit
ARQC	Authorisation Request Cryptogram
BDH	Blinded Diffie-Hellman
CA	Certification Authority
C-APDU	Command APDU
CDCVM	Consumer Device Cardholder Verification Method
CDOL	Card Risk Management Data Object List
CLA	Class byte of command message
CMAC	Cipher-based Message Authentication Code
CRL	Certification Revocation List
CVM	Cardholder Verification Method
DE/DS	Data Exchange & Data Storage Implementation Option
ECC	Elliptic Curve Cryptography
ECSDSA	Elliptic Curve Schnorr Digital Signature Algorithm
EDA-MAC	Enhanced Data Authentication MAC
ERRD	Exchange Relay Resistance Data
GPO	GET PROCESSING OPTIONS
IAD	Issuer Application Data

Abbreviation	Description
IAD-MAC	Issuer Application Data MAC
ICC	Integrated Circuit Card
ID	Identifier
ISO	International Organization for Standardization
MAC	Message Authentication Code
N/A	Not applicable
PAN	Primary Account Number
PDOL	Processing Options Data Object List
PIN	Personal Identification Number
POS	Point of Sale
PPSE	Proximity Payment System Environment
R-APDU	Response APDU
RID	Registered Application Provider Identifier
RRP	Relay Resistance Protocol
SCA	Strong Customer Authentication
SDA	Static Data Authentication
SHA	Secure Hash Algorithm
SKC	AES Session Key for Confidentiality
SKI	AES Session Key for Integrity
TLV	Tag Length Value
TRMD	Terminal Risk Management Data
TVR	Terminal Verification Results