

# How to Use ISO/IEC 24727-3 with Arbitrary Smart Cards

Detlef Hühnlein<sup>1</sup> and Manuel Bach<sup>2</sup>

<sup>1</sup> secunet Security Networks AG,  
Sudetenstraße 16, 96247 Michelau, Germany  
`detlef.huehnlein@secunet.com`

<sup>2</sup> Federal Office for Information Security,  
Godesberger Allee 185-189, D-53175 Bonn, Germany  
`manuel.bach@bsi.bund.de`

**Abstract.** The forthcoming ISO/IEC 24727 series of standards defines application programming interfaces for smart cards and is expected to provide a major contribution to the global interoperability of smart cards and card-applications. However it assumes in part 2 [8] that certain information concerning the capabilities of the card and its (cryptographic) applications is stored *on the card* itself. As already issued smart cards do not provide the required structures, the significance of ISO/IEC 24727 for billions (see [5]) of “legacy cards” seems to be questionable. In order to overcome this problem, the present paper introduces an alternative approach, which does *not require any specific information on the card* but provides the information which is necessary to map generic requests to card-specific APDUs to the middleware in form of XML-based **CardInfo**-files.

## 1 Introduction and Motivation

The forthcoming ISO/IEC 24727 series of standards [7,8,9,10] defines application programming interfaces for smart cards. As this standard – unlike existing cryptographic APIs like PKCS #11 [20] – allows a fine granular access to card-applications and covers aspects of card-application management, it promises to provide a major contribution to the global interoperability of smart cards and card applications. In this architecture (see figure 1) a client-application uses a card-application via two layers (the Service Access Layer defined in [9] and the Generic Card Access Layer defined in [8]). For this purpose the client-application sends some Action Request to the Service Access Layer, which in turn sends a Generic Request to the Generic Card Access Layer. This layer “knows” about the specific capabilities of the card and finally sends a Specific Request to the card-application, which performs some operation and gives back the response through the different layers.

The development of the ISO/IEC 24727 standards was stimulated by the US Government Smartcard Interoperability Specification [17], which defines a virtual card edge interface, which can be supported by cards with a file system according to [12] as well as by Java-cards [16]. In a similar fashion, the

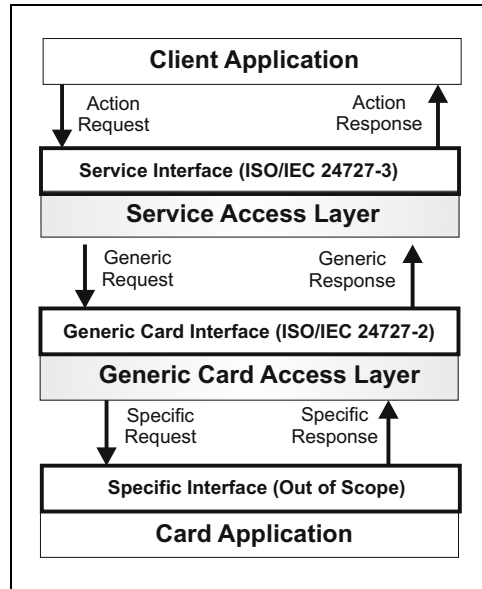


Fig. 1. ISO/IEC 24727 Architecture

Generic Card Interface [8] defines a subset of functions standardized in [12,14] and two files, which may contain further information about the capability of the present card. The Card Capability Description (CCD) tells the Generic Card Access Layer what card applications and predefined Cryptographic Information Application (CIA) profiles according to [11] are present on the card and how some generic request can be mapped to a specific request for the given card. In a similar fashion, the Application Capability Description (ACD) provides such mapping information for application specific requests.

While this approach, just like the virtual card edge from [17], makes it possible that cards with ISO-file system and Java-cards may be accessed using the same interface, it has a major drawback, which seriously limits the applicability of it in (current) practice.

The problem is that the ISO/IEC 24727-2 standard frankly assumes that the *card itself* carries all information (i.e. CCD, ACD and CIA), which is necessary to map some Action Request to the Specific Request for the present card (e.g. consisting of appropriate APDU-sequences referencing a specific file or key on the card).

As the ISO/IEC 24727 series of standards is currently developed it is not surprising that there are no issued cards yet, which comply to this standard and provide the necessary CCD and ACD files. Furthermore, there seem to be very few smart cards in the field, which internal structure is *completely* described by an appropriate CIAInfo-structure according to [11]. This may in part be due to the fact that this structure requires some additional storage on the card and saving storage is still a concern – at least in large volume smart card projects.

Because the card does not provide the required information, it is necessary that the smart card middleware (i.e. some software between the client-application and the card-application) is able to “recognize” a given card type and “knows” how to map a generic call on the Service Interface to card-specific APDUs. The naive but common way to realize this in practice is that the specific features and personalization of some card type are directly coded into the smart card middleware. As this implies that the executable code of the middleware needs to be changed if there are new card types which need to be supported, this clearly renders smart card interoperability more difficult and the maintainance of the smart card middleware turns out to be a major cost factor, especially if the system is to be evaluated according to Common Criteria [6]. Furthermore it is very hard to successfully implement card-application management systems as the middleware would need to be changed if there is a change in some card-application on a supported card. While it is possible to choose a modular middleware design as in [19,1,15] in which only a certain part of the middleware – the “card-provider” – needs to be updated, the problem is not entirely solved as there are still changes to the executable code, which would require some re-evaluation according to [6].

In order to overcome these problems, the present paper introduces an alternative to ISO/IEC 24727-2, in which the necessary information to map a generic Action Request to a Specific Request for the present card (e.g. card-specific APDUs) is provided to the middleware in form of an XML<sup>1</sup>-based configuration file and hence it is *not* necessary to change the executable code of the middleware but it is sufficient to provide a new `CardInfo`-file. This configuration file may be viewed as an off-card variant of the CCD-, ACD- and CIA-files which otherwise would need to be present on the card itself.

The rest of the paper is structured as follows: Section 2 introduces an alternative to ISO/IEC 24727-2 and explains the major content of the `CardInfo`-files as well as its use to recognize a card type and to map generic requests to card-specific APDUs. We conclude this work in section 3 and sketch how our approach may be embedded in a comprehensive framework for electronic identity cards [3] as it is used for the implementation of the eCard-strategy of the German government.

## 2 A Generic Alternative to ISO/IEC 24727-2

In this section we present an alternative to ISO/IEC 24727-2, which allows to use the Service Interface according to [9] with cards which do *not* provide CCD-, ACD- and CIA-files and hence are not compliant to [8].

This section is structured as follows: In section 2.1 we will sketch the main ideas of our approach in which the middleware is fed with so called `CardInfo`-files, which describe how to recognize the card type and allow to translate generic

---

<sup>1</sup> It would also be possible to use ASN.1-based `CardInfo`-files. As the files are not supposed to be stored on the card, tools for handling XML-based data tend to be more widespread than similar tools for ASN.1 and XML serves as basis for the definition of web service interfaces, it seems to be the canonical choice here.

requests to card-specific APDUs. How the middleware may recognize the card type is explained in section 2.2. Section 2.3 sketches how the middleware is able to perform the mapping from a generic request to card-specific APDUs. Section 2.4 explains the structure of the `CardInfo`-files, which are at the heart of our approach, in more detail.

## 2.1 Outline of the Approach

As our aim is to support *arbitrary* cards via the generic Service Interface defined in ISO/IEC 24727-3 [9], it is necessary that the middleware “knows” how to perform the mapping from a generic request to card-specific APDUs. In order to achieve this goal the middleware is fed with `CardInfo`-files which allow to perform the following steps:

1. *Recognition of the card type*

As soon as the card is captured by an interface device, the middleware must be able to “recognize” the type of the card in order to identify the appropriate `CardInfo`-file which allows to perform the mapping of generic requests to card-specific APDUs.

2. *Mapping the generic requests to card-specific APDUs*

When the client-application sends a generic Action Request to the Service Interface the middleware must look into the appropriate `CardInfo`-file in order to translate the generic request to card-specific APDUs.

These two steps are explained in the following subsections.

## 2.2 Recognition of the Card Type

In this step the middleware must be able to “recognize” the type of the presented card in order to determine the applicable `CardInfo`-file. For every card type there is a unique `CardInfo`-file, which contains a set of `CharacteristicFeature`-elements which are used to recognize the type of a given card. A characteristic feature is described by a pointer to a (part of a) file on the card and a reference value, which is compared to the answer provided by the card. As depicted in figure 2 the set of `CardInfo`-files accessible by the middleware is used to build at runtime a “decision tree”, which is traversed upon reset of the card.

After the reset of the card the middleware reads the “application directory file” (EF.DIR) at address '2F00' and checks whether there is a match with one or more `CharacteristicFeature`-elements given by the set of `CardInfo`-files. As we assume in our simple example (which is currently the case in practice) that the presence of certain card applications on a card uniquely determine the card type (e.g. AID='A0 00 00 03 08 00 00 10 00 01 00' would make clear that the card is a Personal Identity Verification (PIV) card [18] and AID='4F 06 D2 76 00 00 01 02' would make clear that the card is a German electronic Health

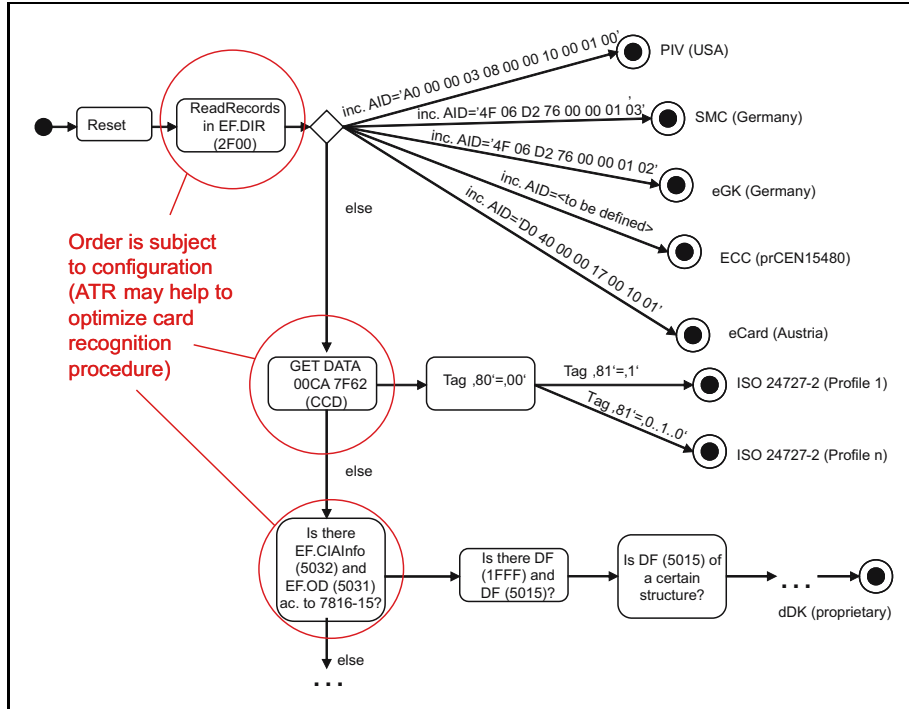


Fig. 2. An example for a decision tree to recognize the card type

Card (elektronische Gesundheitskarte, eGK) [4]) the recognition process would already stop in an acceptable state in which the card type is uniquely determined. If the analysis of EF.DIR in our example would not lead to a match, the next request to the card would check whether the Card Capability Description (CCD) is present at address '7F62' and so on. This process will finally determine the card type or end up with an error message which states that the presented card is not among the supported card types determined by the CardInfo-files. Note that the order of the calls to the card determines the efficiency of the recognition step and hence should be optimized for a certain user environment such that card types which are more likely to occur are tested first.

### 2.3 Mapping the Generic Requests to Card-Specific APDUs

As soon as middleware has determined the type of a card, it can use the information provided in the CardInfo-file in order to map a generic Action Request at the Service Interface according to [9] to a Specific Request (e.g. consisting of card-specific APDUs) for the present card.

This step will be explained by a simple example as depicted in figure 3.

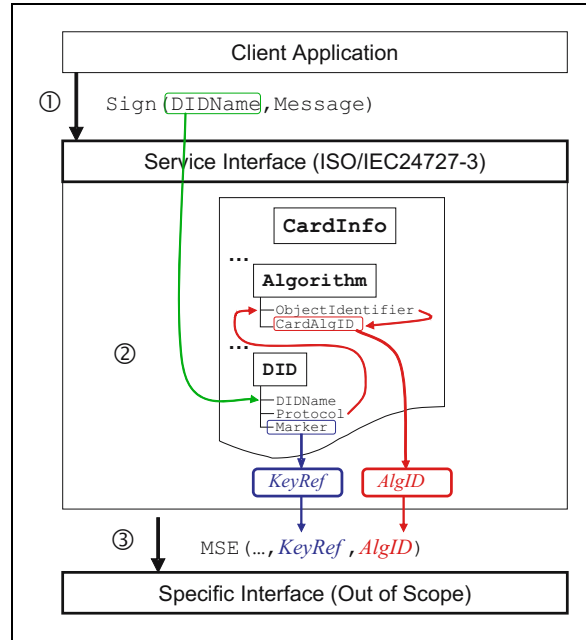


Fig. 3. Mapping of a generic request to card-specific APDUs

Suppose that a client-application wants to sign some message with a key stored on the card. Then it would roughly<sup>2</sup> invoke the `Sign`-function at the Service Interface with the two parameters `Message` and `DIDName`. The parameter `DIDName` is the logical name of a key-structure (called "Differential-Identity" (DID) in [9]), which is used for authentication and other cryptographic purposes. A DID comprises at least the following information:

- `DIDName` – is the logical name of the DID.
- `Protocol`<sup>3</sup> – specifies the cryptographic protocol, for which the DID can be used in form of an object-identifier OID. This OID must be among the algorithms supported by the card (see figure 3).
- `Marker` – may be
  - a PIN / password
  - a symmetric key
  - an asymmetric key, which may be used to generate digital signatures
  - a card-verifiable certificate

<sup>2</sup> As [9] is card-application oriented it may be necessary to connect to a specific card-application first.

<sup>3</sup> In the current draft of [9] this parameter is called `Authentication Protocol`. As a DID can also be used in other cryptographic primitives (e.g. for decryption or signature generation) it would be advisable to change the name of this parameter to `Protocol`.

- a biometric image or template
- a pair of symmetric keys (one for encryption and one for message authentication)

Typically a marker will be a reference to a key on a card (see figure 3).

In order to compute a digital signature the following steps are necessary:

1. *Manage Security Environment (MSE)*

In the first step the middleware will use the MSE command (see [12, Section 7.5.11] and [14, Section 10]) to prepare the card for the computation of a digital signature with a certain key (identified by a card-specific key reference, *KeyRef*) and a certain algorithm (identified by a card-specific algorithm identifier, *AlgID*).

As depicted in figure 3 the *KeyRef* is found in the **Marker**-element of the DID referenced by **DIDName**. The *AlgID* is found by looking into the description of the algorithm with the same object-identifier as the **Protocol** element in the DID referenced by **DIDName**.

2. *PSO: Compute Digital Signature (PSO:Compute DS)*

In the second step the middleware will call the Perform Security Operation: Compute Digital Signature command (see [14, Section 11.7]) and send the data to be signed (DTBS) to the card.

In this way the middleware is able to map all generic requests defined in [9] to card-specific APDUs.

## 2.4 The Structure of the CardInfo-Files

At the heart of our approach are the **CardInfo**-files, which allow to recognize the type of a given card (cf. section 2.2) and map generic requests to card-specific APDUs (cf. section 2.3).

A **CardInfo**-file consists of the following four elements:

- **CardType**
- **CardIdentification**
- **CardCapabilities**
- **ApplicationCapabilities**

The main content of these elements is explained in the following. Full details may be found in [3].

**CardType.** This element contains a unique identifier for the card-type and optionally further useful information, like the name of the specification body or issuer (e.g. “CEN” in the case of a European Citizen Card according to [2]), the name of the card-type (e.g. “European Citizen Card”), the version and date of the specification and further references to specification documents (e.g. a URL where the specification documents [2] of the European Citizen Card may be downloaded).

**CardIdentification.** This element is used to identify the card-type and an individual card of this type. It consists of the following elements:

- **ATR** – may be used to specify a boolean mask of the ATR/ATS which is specific for the card-type. This information may be used to determine an appropriate starting point in the decision tree (cf. figure 2) which is traversed within the card-recognition procedure (cf. section 2.2).
- **CharacteristicFeature** – contains a sequence of characteristic features which are checked in order to recognize the card-type. A **CharacteristicFeature**-element consists of a reference to a (part of a) file (**FileRef**) on the card and a **Value**-element, against which the answers from the card are compared. Note that the set of **CharacteristicFeature**-elements in all **CardInfo**-files available to the middleware and their order determines the structure of the decision tree (cf. figure 2).
- **ICCSN** – may contain a reference to a (part of a) file, which contains a unique serial number of the card (e.g. an Integrated Chip Card Serial Number (ICCSN) or a Primary Account Number (PAN)), which allows to distinguish individual cards of a given type.

**CardCapabilities.** This element contains information about the general capabilities of the given card. It contains the following elements:

- **ISO7816-4-CardCapabilities** – contains information about the minimum requirements concerning the basic capabilities of the card according to [12, Section 8.1.1.2.7]. If the specification of the card-type does not define such minimum requirements, this element may be omitted.
- **ExtendedLength** – possibly contains a pointer to a (part of a) file on the card, which specifies the extended length supported by the card.
- **CryptoCapabilities** – contains information about the cryptographic capabilities of the card. If there is a **CIAInfo**-file according to [11] on the card, which *completely* describes the cryptographic capabilities and keys of the card, it is sufficient to set the boolean element **ISO7816-15-CompliantCard** to **TRUE**. If not this element contains the equivalent information. This means that it contains information about the profiles and card flags according to [11] and the supported algorithms of the card. This includes the object-identifier of the algorithm and the card-specific algorithm-identifier, which are necessary to map the generic requests to card-specific APDUs (cf. figure 3).
- **BiometricCapabilities** – may contain information about the biometric capabilities of the card.

**ApplicationCapabilities.** This element contains information about the card-applications available on the card. For every card-application this contains the following information:

- **ApplicationID** – is a unique identifier of the card-application. This identifier may be a registered card-application according to [13] or a unique value which is defined by the creator of the **CardInfo**-file.

- **ApplicationName** – may contain a user-friendly name of the card-application which only serves for informational purposes.
- **DocumentationReferences** – may contain references to the specification of the card-application.
- **CardApplicationServiceSet** – contains information about the services supported by the card-application, the respective access control information (cf. [9, Section 5.4.3]) and optionally (a reference to) code to be executed if the card is a Java Card [16].
- **DIDInfo** – contains for every Differential-Identity (DID) on the card the information which is necessary to map the generic requests to card-specific APDUs (cf. section 2.3) together with the related access control information. Given this access control information the middleware knows what kind of authentication steps (using other Differential-Identities) are necessary to access a particular DID.
- **DataSetInfo** – contains information about the data sets present on the card and the related access control information. For a card with file system according to [12] a data set is a directory file. A data set consists of a sequence of information about data structures for interoperability (DSI) (cf. [9, Section 8]) and associated access control information. A DSI is referenced by a logical **DSIName** and contains a reference to a (part of a) file on the card and optionally further information which describes the MIME-type and the encoding of the stored data. This **DSIDescription** may be used by a generic client-application (card browser) to visualize arbitrary data stored on the card.

### 3 Conclusion

In this paper we introduced a generic alternative for ISO/IEC 24727-2 [8] which allows to use the Service Interface defined in ISO/IEC 24727-3 [9] with arbitrary smart cards. While the Service Interface provides comprehensive functionality for accessing card-applications, this interface alone is often not sufficient. In particular the experiences gathered in [3] suggest that it is beneficial to have a related interface underneath these layers to access card terminals and another interface above the ISO/IEC 24727-3-interface which supports services for identity management and advanced electronic signatures. While there have been first steps towards standardizing an Interface Device API in [10] it remains to be seen, whether the interfaces in the “Identity Layer” will be standardized within the scope of ISO/IEC 24727 and/or CEN TS 15480.

### References

1. German Signature Alliance. SigAll-API - Specification of the Application Programming Interface to the Signature Card. Version 1.0 (2004)
2. Comité Européen de Normalisation (CEN). Identification card systems – European Citizen Card. CEN proposed Standard prCEN15480 (Working Drafts) (2006)

3. Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eCard-API-Framework (Part 1-6). Technical Directive (BSI-TR-03112), Draft, A copy of the documents may be obtained from the authors (2007)
4. Gesellschaft für Telematikanwendungen der Gesundheitskarte (gematik). The Specification of the German electronic Health Card eHC – Part 2: Applications and application related structures. Version 1.1.1, 2006-03-23, (2006), [http://www.gematik.de/upload/gematik\\_eGK\\_Specification\\_Part2\\_e.V1.1.1\\_516.pdf](http://www.gematik.de/upload/gematik_eGK_Specification_Part2_e.V1.1.1_516.pdf)
5. IMS Research Group. The Worldwide Market for Smart Cards and Semiconductors in Smart Cards–2006 edn. Research Report # IMS9654 (May 2006), [http://www.electronics.ca/reports/ic/smart\\_cards.html](http://www.electronics.ca/reports/ic/smart_cards.html)
6. ISO/IEC 15408: Information technology – security techniques – evaluation criteria for it security (part 1-3). International Standard (2005)
7. ISO/IEC 24727-1: Identification cards – Integrated circuit cards programming interfaces – Part 1: Architecture. Final Draft International Standard (2006-08-25) (2006)
8. ISO/IEC 24727-2: Identification cards – Integrated circuit cards programming interfaces – Part 2: Generic Card Interface. Final Committee Draft (2006-07-30) (2006)
9. ISO/IEC 24727-3: Identification cards – Integrated circuit cards programming interfaces – Part 3: Application programming interface. Committee Draft (2006-09-07) (2006)
10. ISO/IEC 24727-4: Identification cards – Integrated circuit cards programming interfaces – Part 4: API Administration. Working Draft (2006-06-26) (2006)
11. ISO/IEC 7816-15: Identification cards – Integrated circuit cards – Part 15: Cryptographic information application. International Standard (2004)
12. ISO/IEC 7816-4: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. International Standard (2005)
13. ISO/IEC 7816-5: Identification cards – Integrated circuit cards – Part 5: Registration of application providers. International Standard (2005)
14. ISO/IEC 7816-8: Identification cards – Integrated circuit cards – Part 8: Commands for security operations. International Standard (2004)
15. Microsoft Inc.: Cryptography API: Next Generation, <http://msdn2.microsoft.com/en-us/library/aa376210.aspx>
16. Sun Microsystems. Java Card Technology, <http://java.sun.com/products/javacard/>
17. United States of America National Institute for Standards and Technology (NIST). Government Smart Card Interoperability Specification – Version 2.1 (July 2003), <http://csrc.nist.gov/publications/nistir/nistir-6887.pdf>
18. United States of America National Institute for Standards and Technology (NIST). Interfaces for Personal Identity Verification. NIST Special Publication 800-73-1 (March 2006), <http://csrc.nist.gov/publications/nistir/nistir-6887.pdf>
19. Open Card Consortium. OpenCard Framework Version 1.2, <http://www.opencard.org/docs/1.2/index.html>
20. RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard - Version 2.2. Public Key Cryptography Standards – PKCS #11 (June 2004), <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf>