JAHIS

ＪＡＨＩＳ標準　１８－００１

Japanese Association of Healthcare Information Systems Industry

ＪＡＨＩＳ
HPKI 対応 IC カードガイドライン Ver.3.0a

２０２３年２月

一般社団法人 保健医療福祉情報システム工業会
医療システム部会　セキュリティ委員会
セキュアトークン WG

ＪＡＨＩＳ　HPKI 対応 IC カードガイドライン Ver.3.0a

ま　え　が　き

本ガイドラインは保健医療福祉分野における電子署名及び電子認証を行うに際して利用される PKI の機能を搭載した IC カード及び IC カードの利用環境に対する要求事項を定めたものである。

保健医療福祉分野においては、平成 17 年 3 月に厚生労働省により「医療情報システムの安全管理に関するガイドライン」（以下、「安全管理のガイドライン」と言う）が策定され、継続的に改定が行われている。「安全管理のガイドライン」C 項最低限のガイドラインとして、6.11 章において相手先の識別と認証において PKI を利用出来ること、6.12 章において保健医療福祉分野 PKI 認証局又は認定特定認証事業者等の発行する電子証明書を用いて電子署名を施すことが記載されている。また、平成 17 年 4 月には、同省にて「保健医療福祉分野 PKI 認証局 証明書ポリシ」が策定され、国際標準に準拠した保健医療福祉分野向けの PKI（HPKI）証明書の発行ルールが確定した。さらに平成 21 年度には厚生労働省の医療情報ネットワーク基盤検討会において「保健医療福祉分野 PKI 認証局認証用（人）証明書ポリシ」の策定が行われた。これにより、署名用に続き、認証用についても HPKI 証明書の発行が行えることとなった。

JAHIS は、産業界の業界団体としてこれら国の施策に協力するとともに普及促進を図るための相互運用性の確保を図ることが重要な役割であることから、2008 年 6 月に電子署名用の IC カードに関する第 1 版を発行し、2010 年 6 月に第 2 版として電子認証用の IC カードに関する記載を追加した。本 Ver.3.0 はタイトルのバージョン表記を他の JAHIS 標準に合わせた上で、第 1 版、第 2 版を統合し、最新の動向を踏まえて追加改定を行ったものである。また本ガイドラインは、制定された「JAHIS ヘルスケア PKI を利用した医療文書に対する電子署名規格 Ver.1.1」、「ヘルスケア PKI を利用した医療文書に対する電子署名規格 PAdES 編 Ver.1.0」及び「JAHIS HPKI 電子認証ガイドライン V1.1」とともに利用されることを前提としており、同規格の下で利用される IC カードの相互運用性の仕様も定めている。これらのことから、本ガイドラインの主な対象読者は IC カード及び PKI のライブラリ開発を担当するエンジニアとなっている。

本ガイドラインは、JAHIS 会員各社の意見を集約し、「JAHIS 標準」の一つとして発行したものである。したがって、会員各社がシステムの開発・更新に当たって、本ガイドラインに基づいた開発・改良を行い、本ガイドラインに準拠していることをその製品のカタログ・仕様書等に示し、さらにその製品の使用においてユーザが理解すべき内容を説明する場合などに使われることを期待している。

なお、本ガイドラインで扱う電子署名およびシステムの要件は、参照規格や技術動向にあわせて変化する可能性がある。JAHIS としても継続的に本ガイドラインのメンテナンスを重ねてゆく所存であるが、本ガイドラインの利用者はこのことにも留意されたい。

本ガイドラインが、HPKI の普及・推進に多少とも貢献できれば幸いである。

２０２３年２月

一般社団法人 保健医療福祉情報システム工業会
医療システム部会 セキュリティ委員会 セキュアトークン WG

＜＜　告知事項　＞＞

本ガイドラインは関連団体の所属の有無に関わらず、ガイドラインの引用を明示することで自由に使用することができるものとします。ただし一部の改変を伴う場合は個々の責任において行い、本ガイドラインに準拠する旨を表現することは厳禁するものとします。

本ガイドラインならびに本ガイドラインに基づいたシステムの導入・運用についてのあらゆる障害や損害について、本ガイドライン作成者は何らの責任を負わないものとします。ただし、関連団体所属の正規の資格者は本ガイドラインについての疑義を作成者に申し入れることができ、作成者はこれに誠意をもって協議するものとします。

目　　次

1. 適用範囲

本ガイドラインでは、電子署名および電子認証を目的とした HPKI で使用される IC カード、及び IC カードの利用環境に対する要求事項を定める。

- IC カード機能・仕様
- IC カードのセキュリティ要件
- IC カードを利用する端末の機能
- IC カードを利用する端末のセキュリティ要件
- 相互運用性を確保するための IC カード内の PKI アプリケーションの仕様
- 相互運用性を確保するための IC カードを利用する際のインタフェースの仕様

なお、IC カードの発行及び私有鍵（秘密鍵）又は証明書の IC カードへの登録については本ガイドラインのスコープ外とする。

2. 引用規格・引用文献

厚生労働省・保健医療福祉分野 PKI 認証局　署名用証明書ポリシ　1.8 版（令和 4 年 10 月）
https://www.mhlw.go.jp/content/10808000/001001440.pdf

厚生労働省・保健医療福祉分野 PKI 認証局認証用（人）証明書ポリシ　1.7 版（令和 4 年 10 月）
https://www.mhlw.go.jp/content/10808000/001001441.pdf

厚生労働省・保健医療福祉分野 PKI 認証局認証用（組織）証明書ポリシ　1.1 版（平成 22 年 3 月）
https://www.mhlw.go.jp/content/000466966.pdf

JIS X 19790:2015 セキュリティ技術 – 暗号モジュールのセキュリティ要求事項

ISO/IEC 7816-4:2020 Identification cards — Integrated circuit cards — Part 4: Organization, security and commands for interchange

ISO/IEC 7816-8:2021 Identification cards — Integrated circuit cards — Part 8: Commands and mechanisms for security operations

ISO/IEC 7816-15:2016 Identification cards — Integrated circuit cards — Part 15: Cryptographic information application

PKCS#11 V2.20 [1]

Crypto API
https://learn.microsoft.com/ja-jp/windows/win32/seccrypto/cryptography-portal

Cryptography API: Next Generation (CNG)
https://learn.microsoft.com/ja-jp/windows/win32/api/ncrypt/

[1] 文書は JAHIS 内本ガイドラインのページから入手可能である。

3. 用語の定義

DF (dedicated file)

ファイル制御情報と任意選択として割付け利用可能なメモリとを含んでいる構造。

EF (elementary file)

同一ファイル識別子を共有するデータオブジェクト、レコード、又はデータ単位の集合。

HPKI (Healthcare Public Key Infrastructure)

保健医療福祉分野のために構築された公開鍵認証基盤で、医療従事者の資格等の属性を証明する機能を有する。

IC カード

１つ以上の IC チップを搭載したカード。形状は ISO/IEC 7810 によって規定される。

IEF （internal elementary file)

カードが解釈し実行するテータを格納する EF。

PIN (Personal Identification Number)

一般には特定の機能を使用する際に認証を得るためにエンティティを確認するために入力される数字の文字列を指すが、本ガイドラインではパスワードと同じく数字以外の文字をも含んだ文字列を指す。

PKI (Public Key Infrastructure)

エンティティの信頼性に関して、信頼できる第三者が公開鍵に対して電子署名を付与した公開鍵証明書によって保証する非対称暗号技術を応用した認証基盤。

WEF（working elementary file）

カードが内容を解釈しない外部データを格納する EF。

オフセット

データ単位を提供する EF でデータ単位の相対位置を示す数値。

公開鍵

エンティティの非対称鍵対のうち、公にされて使用される鍵。

私有鍵

エンティティの非対称鍵対のうち、そのエンティティにだけによって使用されるべき鍵。

証明書

人等のエンティティとその関連する公開鍵を関連付けるデジタル署名によって保護された文書。

認証局

公開鍵認証基盤で、エンティティに対して公開鍵証明書を発行することによってエンティティの存在を保証する組織体。

ハッシュ値

あるデータが与えられた場合に、ハッシュ関数（暗号的に安全性の確立された一方向性関数）によって得られたそのデータを代表する数値。

ファイル識別子 (file identifier)

ファイルを指定するために使用される 2 バイトのデータ要素。

4. HPKI 用 IC カードの機能

4.1. IC カードの種類

4.1.1. ネイティブ型 IC カード

ネイティブ OS を搭載する IC カードをネイティブ型 IC カードと呼ぶ。ネイティブ OS とは、IC カード内のすべてのソフトウェアが IC チップに依存するプログラムコードで構成される OS をいう。また、ネイティブ OS は、外部ノードから IC カードに送られるすべてのコマンドを、OS レベルで解釈実行することを特徴とする。また、そのため実行処理速度が速いことが特徴である。通常、固定メモリのネイティブ型 IC カードはカードが発行された後は、アプリケーション機能の追加、変更および削除が困難であるが、メモリをセクタ管理するものには可能なものが存在する。しかし、このタイプは次に述べるプラットフォーム型 IC カードの範疇に含まれるものとする。

4.1.2. プラットフォーム型 IC カード

プラットフォームを搭載する IC カードを、プラットフォーム型 IC カードと呼ぶ。プラットフォームとは、カードアプリケーション（外部ノードから IC カードに送られるコマンドを解釈・実行するアプリケーションソフトウェア）をダウンロードすることが可能な IC カードに搭載される OS を含むソフトウェアをいう。また、プラットフォームは、IC カード内にダウンロードされたカードアプリケーションを SELECT FILE コマンドによって選択、起動し、以降外部ノードから IC カードに送られるコマンドを、他のカードアプリケーションを選択する場合を除き当該カードアプリケーションで処理することを特徴とする。

代表的なプラットフォームとして、JavaCard、MULTOS がある。JavaCard、MULTOS のようにカードアプリケーションが IC カード内の IC チップに依存しないプログラムコードによって構成される場合もある。このような場合、プラットフォームは、IC チップに依存しないプログラムコードを、IC チップに依存するプログラムコードに変換する仮想マシン（VM）を有する。さらに広義には、IC カード内のすべてのソフトウェアが IC チップに依存するプログラムコードで構成される場合であっても、カードアプリケーションをダウンロード可能であり、SELECT FILE コマンドでカードアプリケーションを選択、起動し、以降外部ノードからのコマンドをカードアプリケーションが処理することを特徴とする機能を実現するソフトウェアの場合、プラットフォームと呼ぶ。

プラットフォーム型 IC カードは カードが発行された後でも、アプリケーション機能がプログラムのダウンロードによって追加、変更が可能であり、不必要になれば削除が可能である。

4.2. IC カードアプリケーションの構成

4.2.1. シングルアプリケーションカード

一つの IC カードの CPU 上で１種類のサービスしか行わないカードをシングルアプリケーションカードという。初期の交通系の IC カード等がこれに該当する。現在の金融系の IC カードは、IC キャッシュカード、デビッドカード、静脈認証など複数のアプリケーションが搭載されているので、この範疇には属さない。

4.2.2. マルチアプリケーションカード

マルチアプリケーションカードは１枚のカード上で複数のサービスを提供することが可能なカードのことを意味し、発行後にアプリケーションの追加、削除が可能なモデルも存在する。そのため、複数のサービス提供者によって独自のアプリケーションを追加、削除する利用シーケンスで特に活用される。

4.3. HPKI 用 IC カードに要求される機能

4.3.1. 私有鍵保存機能

私有鍵（Private Key）は公開鍵と対になる鍵である。私有鍵は、その加入者によって秘密保持すべき情報であり、公開せず、他人に漏れないように鍵の所有者だけが管理する。認証局は、いかなる場合でもこれらの鍵へのアクセス手段を提供しない。（HPKI 証明書ポリシ）
電子署名および電子認証のために使用される私有鍵は、法律によって必要とされる場合を除き、預託されないものとする（HPKI 証明書ポリシ）。また、署名目的の私有鍵の回復も行わない（HPKI 署名用証明書ポリシ）。

加入者の私有鍵が認証局で生成される場合は、セキュリティが確保された環境で IC カードへの書き込みを行う、又は IETF RFC 6712 及び RFC 4210「証明書管理プロトコル」に従ってオンライントランザクションで、又は同様に安全な方法によって、加入者に引き渡されるものとする。認証局はオリジナルの私有鍵を引き渡した後は私有鍵のコピーを所有していないことの証明ができるものとする。（HPKI 証明書ポリシ）
この場合、生成された私有鍵は、内部基礎ファイル（IEF）に格納される。

加入者の私有鍵が IC カード内で生成される場合は、GENERATE ASYMMETRIC KEY PAIR コマンドによりカード内で生成し、内部基礎ファイル（IEF）に格納する。（ISO/IEC 7816-8）

私有鍵はハッシュ値を入力とした署名演算に使用される。

4.3.2. 公開鍵証明書保存機能

加入者の私有鍵が認証局で生成される場合及び IC カード内で生成される場合共に必要となる機能である。公開鍵証明書は WEF に格納され、公開鍵の検証時には IC カードから読み出される。

4.3.3. 私有鍵生成及び公開鍵エクスポート機能（オプション）

GENERATE ASYMMETRIC KEY PAIR コマンドにより， IC カード内で鍵ペアを生成し、私有鍵は内部基礎ファイル（IEF）に格納される。また、公開鍵証明書を CA 局へ要求するために GENERATE ASYMMETRIC KEY PAIR コマンドにより、既に IC カード内で生成された公開鍵にアクセスする。(ISO/IEC 7816-8)。認証局の要件により加入者の私有鍵が IC カード内での生成を求められる場合は、必須の機能となる。

4.3.4. 私有鍵インポート機能（オプション）

IC カード内で鍵の生成が望ましいが必須ではない。外部で鍵ペアを生成した場合は私有鍵を IEF へ、公開鍵証明書を WEF へインポートする。認証局の要件により加入者の私有鍵が認証局で生成される場合は、必須の機能となる。

4.3.5. 公開鍵証明書エクスポート機能

IC カード内に保存された公開鍵証明書は公開鍵の検証のために署名に付属させる場合等に WEF を指定して読み出す。

4.3.6. 署名機能

PERFORM SECURITY OPERATION コマンドの COMPUTE DIGITAL SIGNATURE 処理により，デジタル署名の計算を行う。

署名計算の中で使用する私有鍵を指定し，ハッシュ値が IC カードに入力される。

4.3.7. 認証機能

PERFORM SECURITY OPERATION コマンドの COMPUTE DIGITAL SIGNATURE 処理により，

デジタル署名の計算を行う。
署名計算の中で使用する私有鍵を指定し，チャレンジ値がICカードに入力される。

4.3.8. 私有鍵活性化機能

私有鍵を署名などの運用に使用することができる状態にすることを活性化（Activate）という。私有鍵の活性化データが認証局で生成される場合は、活性化データが加入者に伝えられた後は、認証局においては完全に破棄し、保管しないものとする。また、伝えられた活性化データは、認証局で定められた規定に従い、加入者により安全に保護するものとする。私有鍵の活性化データを加入者が生成する場合は、認証局で定められた規定に従い、加入者により安全に保護するものとする。（HPKI 証明書ポリシより）

私有鍵の活性化の手段としては、加入者による利用者認証、生体認証、端末が使用する IC カードがドメインで認められたものかを IC カードが認証する端末認証などがあるが、本ガイドラインでは、PIN 照合による利用者認証のみを規定している。

加入者確認の本人確認の為の認証鍵と署名鍵を同じものを用いた場合、署名時のハッシュ値を認証時にチャレンジコードとして IC カードに送り込むとレスポンスコードとして署名された値が自動的に送付されてしまう。そのため、認証鍵と署名鍵は分けなくてはならない。署名は本来、署名する内容に対して確認し意識して署名をするもので、そのために署名に対しては必ず、加入者の意思確認のための確認として私有鍵の活性化ステップを入れなくてはならない。

4.4. IC カードのセキュリティ機能

4.4.1. 概要

IC カード及び IC カードに搭載されたアプリケーションの安全性を 4.4.2 で説明し、IC カード及びアプリケーションを安全に利用するために持つセキュリティ機能を 4.4.3 で説明する。

4.4.2. アプリケーションの安全性

1） 偽造・複製が困難

耐タンパ性によってカードに内蔵された IC チップ上の情報を保護している。具体的には、以下のような複製を行うための情報取得が困難な技術的な対策によって保護している。

・樹脂を封入してカードからチップを取り出しするのを難しくする
・IC チップを多層化することによって顕微鏡での観察を困難にする
・異常に利用されたことを検出するセンサーを組み込み、異常を検知するとデータを消去するなど電気的な解析を困難にする
・情報を記録するメモリをランダムに配置することで、再現を困難にする
・消費電力や電圧の変化からのデータ解析を防ぐため、消費電力や計算時間等を均一あるいはランダムにする

2） 不正使用が困難

暗証番号・パスワードあるいは暗号鍵の確認によって、正当性を IC カードが確認したアクセスだけが許可される仕組みを持っている。アクセス制御は、アプリケーション単位、ファイル単位に設定することが可能で、読出し・書込み・書換えなどの操作ごとに異なるアクセス権を設定することが可能となっている。

暗証番号・パスワードおよび暗号鍵は、値を設定できるが読み出すことのできない情報として IC カード内で管理されるので、カード製造者や管理者であっても不正な利用によってその値を取得することはできない。暗証番号・パスワードあるいは暗号鍵の確認は、あらかじめ設定された回数の照合や認証に失敗すると鍵の利用を自動的に停止（閉塞）することが可能なので、不正な利用を多数回繰り返すことも難しい仕組みを持っている。

3） アプリケーションの独立性

複数のアプリケーションが 1 枚の IC カード上に存在する場合、各々のアプリケーション内のファイルや鍵、セキュリティ属性は独立である。そのため、あるアプリケーション内のデータへの操作が別のアプリケーションのデータに影響を与えることはない。また、あるアプリケーション内の認証結果によるセキュリティ状態が、別のアプリケーションに引き継がれることはない。そのため、あるアプリケーションへの操作が別のアプリケーションに影響を与えることはなく、それぞれのアプリケーションがあたかも独立な IC カードであるように安全性を保ったまま動作させることが可能である。

4.4.3. カードアプリケーションに要求されるセキュリティ機能

1） 端末認証（外部認証機能）

利用される端末の正当性を外部認証の機能によって確認する機能。 端末が鍵を知っていることによる端末の正当性を確認する。例えば、GET CHALLENGE コマンドと、それに続く EXTERNAL AUTHENTICATE コマンドを使用する。オンライン接続の端末認証に応用した場合、接続先のサーバ認証にも利用できる。（注）

2） 利用者認証

A） PIN の照合により PIN を知っていることによる利用者の確認。例えば VERIFY コマンドを使用する。

B） 鍵を知っている（持っている）ことによる利用者の確認。例えば、GET CHALLENGE コマンドと、それに続く EXTERNAL AUTHENTICATE コマンドを使用する。（注）

3） セキュアメッセージング

コマンド及びその応答に対してその一部又は全体を暗号によって保護するための方法。コマンド実行のとき通信されるデータを暗号によって保護する。データの隠蔽機能と完全性チェック機能を持つことが出来る。端末と IC カード間での通信の安全性が保証されない場合に用いられる通信の安全性確保の機能である。（注）

4） 内部認証

カード（アプリケーション）の正当性を端末が認証するための機能である。接続端末装置から送られるチャレンジを、INTERNAL AUTHENTICATE コマンドにより IC カード内に格納されている内部認証鍵と計算しその結果出力を端末が判断し、IC カードの正当性を確認する。オンライン接続の端末認証に応用した場合、サーバが IC カードの正当性を認証する為に利用できる。（注）

注：今回の署名及び認証機能の相互運用性確保の範囲では、利用シーケンスに含まない。利用した場合には、相互運用の妨げになる可能性があることに留意する必要がある。

5. 相互運用性確保のための仕様

5.1. 相互運用性確保

5.1.1. 相互運用性を確保する範囲

本ガイドラインが目標とする相互運用性の確保は、異なる HPKI 認証局が発行した IC カードを用いても、クライアントの電子署名アプリケーションが電子署名あるいは電子認証を実行可能とすることにある。図 1 にその概念を示す。

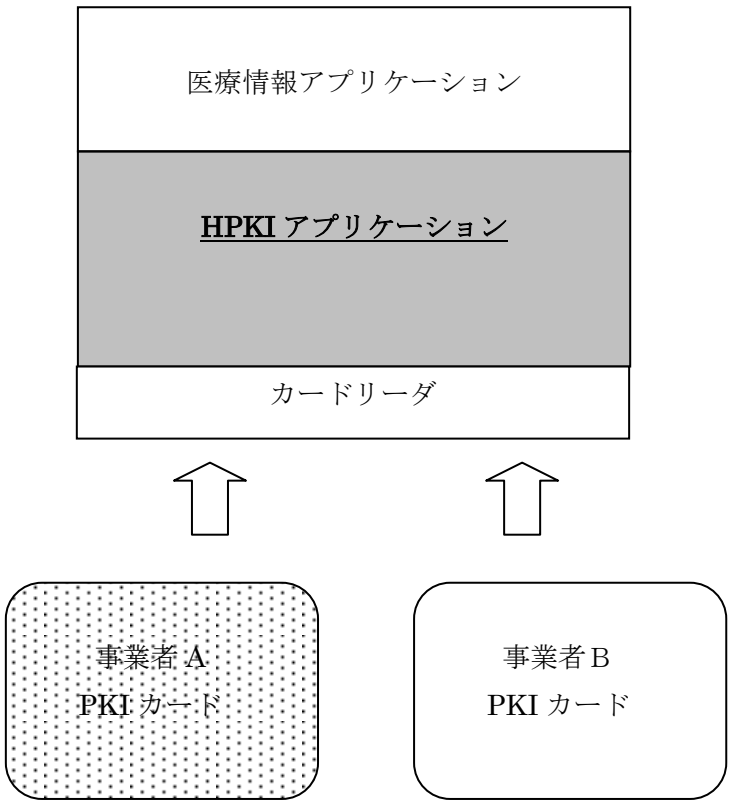


図１　相互運用の範囲

このような相互運用性を検討する場合、図２に示す通りクライアント側のソフトウェアを以下の層に分けて検討する必要がある。

1) HPKI アプリケーション層
2) PKI 機能を提供する汎用ミドルウェア層
3) IC カードの入出力を提供する汎用 IC カードリーダインターフェース層

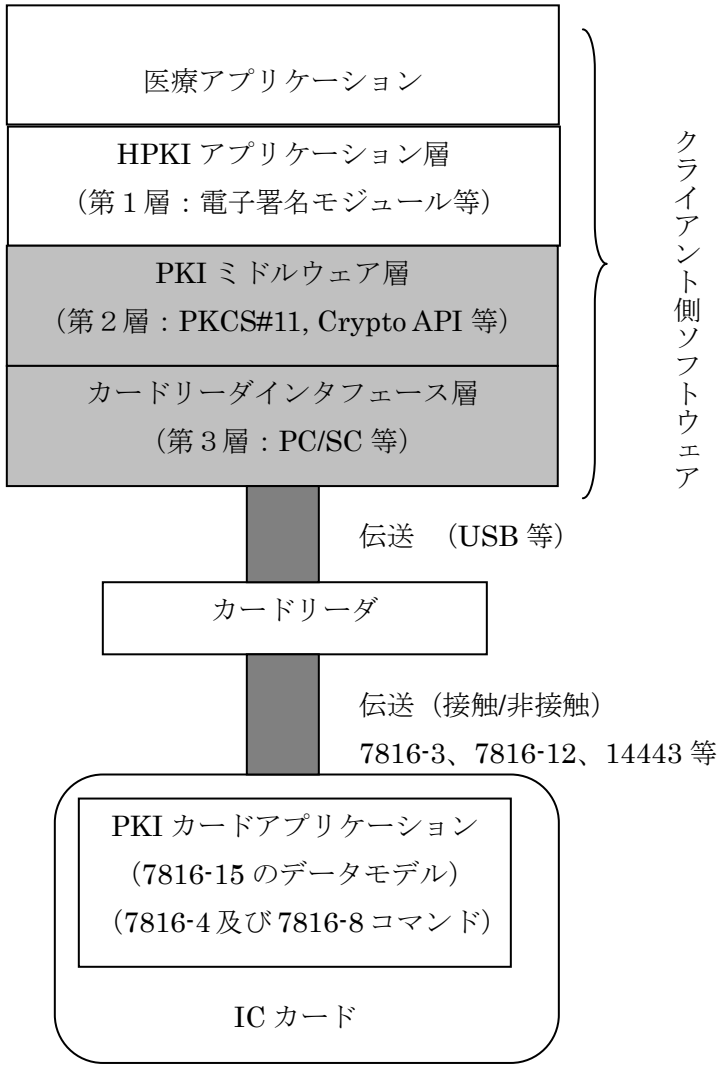


図２　PKI カードアプリケーションの基本構造

第１層は HPKI を利用する署名などの機能を提供する層で、本ガイドラインの対象外となる。HPKI サービスを提供することになる。

第 2 層は、HPKI に特化しない汎用的な PKI の機能を提供する層で、PKCS #11 あるいは Crypto API（CAPI）など、暗号機能を提供するモジュールが相当する。具体的には、IC カード内の PKI カードアプリケーションのデータモデルに従い、要求された PKI 機能を実現するカードコマンドに変換することによって IC カード内の PKI カードアプリケーションの機能を利用するための機能を提供する。PKI カードアプリケーションのデータモデルに関連する標準が ISO/IEC 7816-15 であり、カードコマンドに関連する標準が ISO/IEC 7816-4 及び ISO/IEC 7816-8 となる。第 2 層に相当するモジュールの機能とインタフェースを規定することによって、異なる HPKI 認証局が発行した IC カードであっても電子署名及び電子認証の機能を実行することが可能となり、相互運用性を確保したことになる。特に各 HPKI 認証局が提供する PKI カードアプリケーションの相互運用性を高めるためには、PKI カードアプリケーションのデータモデル、それを利用するためのカードコマンドの相互運用性を高める必要がある。これまでの多くの PKI 環境では、発行者、仕様の異なる IC カードによって同じ機能を提供する必要性がなかったため、この部分の相互運用性が考慮されておらず、問題を生じることがあった。

第3層は、汎用的なICカードリーダとの入出力を実現する層で、第2層から受けたICカードに対する命令（コマンド）を、ICカードリーダを通じてICカードに伝える機能を提供する。プロトコルのハンドリング、ICカード及びカードリーダの制御などが含まれる。Windowsでは、PC/SCと呼ばれるモジュールがこの層に相当する。第3層のモジュールの機能とインタフェースの規定によって、対象となるICカードとカードリーダの物理的な違い(例えば、接点付きと非接触など)によらない、汎用的なICカード入出力を実現することができる。これによって、異なるインタフェースのICカードや異なるカードリーダを利用した場合でも、ICカードとICカード内のPKIカードアプリケーションに対して同じ入出力を実現することができる。
本ガイドラインでは、第2層のインタフェースを5.2に記述する。また、第2層で利用するICカード内のPKIカードアプリケーションのデータモデルを5.3に、カードコマンドを5.4で説明する。

5.1.2. 相互運用性を確保するための条件

本ガイドラインでは、異なるHPKI認証局が発行したICカードのPKIカードアプリケーションでの相互運用性確保のための仕様を以下の条件で示すものとする。

1） 相互運用の範囲

ICカードは、発行、個人向け初期化、利用、変更、廃棄などのライフサイクルのいずれかの状態に属する。本ガイドラインの相互運用性は、利用の部分、特に電子署名および電子認証を行う部分のみの相互運用を範囲とする。

ICカードには、電子署名用及び電子認証用のPKIカードアプリケーションが搭載されるものとする。

2） 証明書の適用範囲

HPKI認証局が発行した証明書で、証明書の適用範囲は認証局の証明書ポリシ(CP)に従ったものとする。適用範囲外での利用（例：スマートカードログオン）に関しては、動作を保証しない。

3） 複数の証明書保持

電子署名用PKIカードアプリケーション及び電子認証用PKIカードアプリケーションには、それぞれのエンドエンティティの証明書だけではなく、HPKI認証局の証明書（複数の場合がある）、厚労省のHPKIルート認証局の自己署名証明書を含めた証明書が格納できる。

4） 複数の資格への対応

複数の資格を持つ医療従事者は、業務によって証明書を使い分ける必要がある。本ガイドラインでは、1枚の証明書には1つのhcRole（保健医療福祉分野での役割、資格）が格納されているものとする。複数の資格を使い分ける際には、複数のPKIカードアプリケーションによって、解決するものとする。当面は、1枚のICカードには1資格に対応したPKIカードアプリケーションが搭載されると想定する。将来は複数のhcRoleを記述した証明書が発行される可能性、あるいは複数の資格を持つエンドエンティティに対してそれぞれのhcRoleを記述した複数の証明書を1枚のICカードに格納する可能性はあるが、本ガイドラインの相互運用性の範囲には含まないものとする。

5） ミドルウェアのHPKI対応

HPKIの証明書にはhcRoleが記述されているが、ミドルウェア層ではhcRoleを用いたHPKI独自の処理は行わない。必要な場合には、HPKIアプリケーション層あるいは医療アプリケーションで行うものとする。

6） ICカードの利用環境

カードの利用環境は、「医療情報システムの安全管理に関するガイドライン」などに沿って管理された医療機関が管理する端末を想定する。悪意をもったソフトウェアなどが存在せず、安全に稼動するよう管理されているものとする。そのため、ICカードと利用する端末の間の認証、通信の安全性を確保するセキュアメッセージングなどは本ガイドラインの範囲外とする。

7） 対象とするICカード

利用されるカードは、4.1で示すICカードの種類があるが、特に限定しない。発行時の構成を変えないIC

カード、発行後にアプリケーションを追加可能な IC カード、接点付き IC カード、非接触 IC カード等、いずれもが含まれるものとする。

8） 標準準拠

長期に渡る相互運用性を確保するため、既存の JIS あるいは ISO に準拠した仕様とする。

9） 私有鍵と証明書のアクセス権

電子認証の際には、初回の PIN 入力によりカード保有者認証を行うものとする。電子署名の際には、毎回 PIN 入力によるカード保有者認証を行うものとする。PIN の変更は、相互運用性を確保する範疇には含めないものとする。証明書の読み出しは、アクセス制御を行わないものとする。

１０） パスワード（PIN）の入力

アプリケーションの性質により、電子署名アプリケーション側で PIN を保持し、連続した署名実施の際にユーザ側の PIN 入力を省略することは許容する。

5.1.3. 電子署名カードアプリケーションと電子認証カードアプリケーションの共存

１つの認証事業者が電子署名用及び電子認証用の両方の証明書を発行する場合、1 枚の IC カードに２つのアプリケーションを搭載することが可能となる。その場合の基本的な構造を図３に示す。

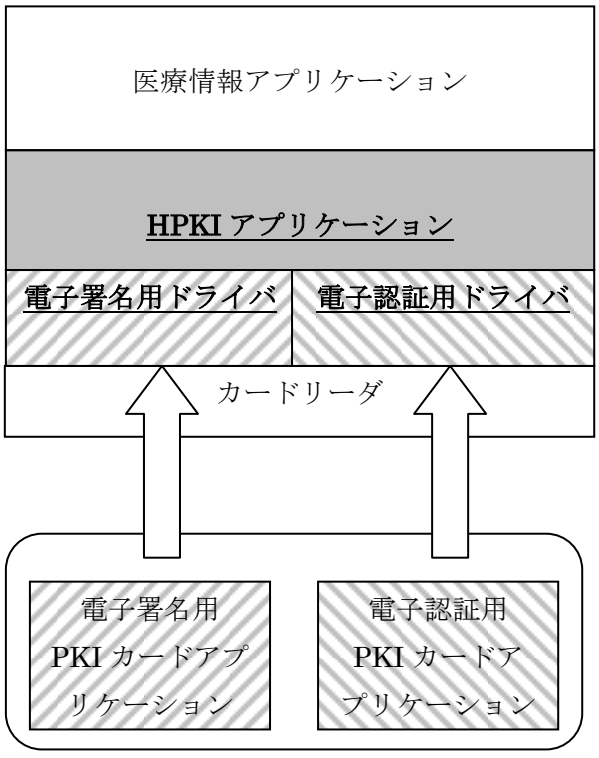


図３　電子署名用アプリケーションと電子認証用アプリケーションの共存

電子署名用の PKI アプリケーションは、電子署名用のドライバを通じて利用され、電子認証用の PKI アプリケーションは、電子認証用のドライバを通じて利用されるものとする。

5.2. アプリケーションプログラムとのインタフェース

5.2.1. 概要

本ガイドラインでは、PKCS #11 インタフェース、Crypto API インタフェース、及び Cryptography API: Next Generation の 3 種類のインタフェースを想定する。これらのインタフェースを実装する場合に、それぞれが満たさなければならない最小限の機能を 5.2.2、5.2.3、及び 5.2.4 で説明する。

共通の条件は、以下の通りとする。

- マルチスレッドから 1 枚のカードへの同時アクセスは想定しない。複数のスレッドからアクセスする必要がある場合には、実行条件に注意する必要がある。
- 署名の際には、インタフェースレベルでは毎回 PIN を設定することを推奨する。但し、アプリケーション側で PIN を保持して利用することは妨げないので、アプリケーションの性質によって決定する必要がある。
- PIN に設定されるパスワードの文字セットは、ASCII とする（CryptoAPI の制限による）。

5.2.2. PKCS #11 インタフェース

PKCS #11 のライブラリ識別名の一覧を表 1 に示す。

表 1　PKCS #11 のライブラリ識別名

No	種類	ライブラリ識別名
1	電子署名用	HpkiSigP11（例:HpkiSigP11_abc.dll）
2	電子認証用	HpkiAuthP11（例:HpkiAuthP11_aab.dll）

相互運用を実現するために必要となる、PKCS #11 のサポートすべき機能の一覧を表 2 に示す。

表 2　PKCS #11 の関数一覧

No	API 名	概要
1	C_GetFunctionList	関数ポインタリストを取得する。
2	C_Initialize	PKCS#11 ライブラリを初期化する。
3	C_Finalize	PKCS#11 ライブラリを終了する。
4	C_GetInfo	ライブラリ情報を取得する。
5	C_GetSlotList	スロットリストを取得する。
6	C_GetSlotInfo	スロット情報を取得する。
7	C_GetTokenInfo	トークン情報を取得する。
8	C_GetMechanismList	サポートメカニズム（アルゴリズム）を取得する。
9	C_GetMechanismInfo	メカニズム（アルゴリズム)情報を返す。
10	C_OpenSession	セッションを確立する。
11	C_CloseSession	セッションを切断する。
12	C_CloseAllSessions	すべてのセッションを切断する。
13	C_GetSessionInfo	セッション状態を取得する。
14	C_Login	トークンをログイン状態にする。
15	C_Logout	トークンをログアウト状態にする。
16	C_FindObjectsInit	オブジェクトの検索を開始する。
17	C_FindObjects	オブジェクトの検索を行う。
18	C_FindObjectsFinal	オブジェクトの検索を終了する。
19	C_GetAttributeValue	オブジェクトの属性値を取得する。
20	C_SignInit	署名処理を初期化する。
21	C_Sign	データに署名を行う。

サポートすべき API の仕様は以下の通りとなる。なお、戻り値および構造体に関しては、PKCS#11 を参照のこと。

(1) C_GetFunctionList

API 名	C_GetFunctionList		
概要	関数ポインタリストを取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetFunctionList) (CK_FUNCTION_LIST_PTR_PTR ppFunctionList) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_FUNCTION_LIST_PTR_PTR	OUT	関数アドレスリストポインタ

(2) C_Initialize

API 名	C_Initialize		
概要	PKCS#11 ライブラリを初期化する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_Initialize) (CK_VOID_PTR pInitArgs) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_VOID_PTR	IN	NULL ポインタを指定

(3) C_Finalize

API 名	C_Finalize		
概要	PKCS #11 ライブラリを終了する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_Finalize) (CK_VOID_PTR pReserved) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_VOID_PTR	IN	NULL ポインタを指定

(4) C_GetInfo

API 名	C_GetInfo		
概要	ライブラリ情報を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetInfo) (CK_INFO_PTR pInfo) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_INFO_PTR	IN/OUT	ライブラリ情報ポインタ
備考	取得可能なライブラリ情報は以下の通り。 CK_INFO::cryptokiVersion: PKCS #11 規格バージョン: 2.20 CK_INFO::manufacturerID: ライブラリ製造者名: CK_INFO::flags: ビットフラグ: 0 CK_INFO::libraryDescription: ライブラリ記述文:「HPKI 3.0」注 CK_INFO::libraryVersion: ライブラリバージョン		

注：旧版に対応したライブラリからは「HPKI2.0」が返されるが、後方互換性のためエラーとしないこと。

(5) C_GetSlotList

API 名	C_GetSlotList		
概要	スロットリストを取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetSlotList) (CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList, CK_ULONG_PTR pulCount) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容

引数	CK_BBOOL	IN	TRUE: カード有りのスロットリストを返す FALSE: 接続されているすべてのスロットリストを返す
	CK_SLOT_ID_PTR	IN/OUT	スロット ID リストポインタ
	CK_ULONG_PTR	IN/OUT	スロット ID リスト件数

(6) C_GetSlotInfo

API 名	C_GetSlotInfo		
概要	スロット情報を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetSlotInfo) (CK_SLOT_ID slotID, CK_SLOT_INFO_PTR pInfo) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID
	CK_SLOT_INFO_PTR	IN/OUT	スロット情報ポインタ
備考	取得可能なスロット情報は以下の通り。 CK_SLOT_INFO::slotDescription: スロット記述文: PC/SC リーダ名称 CK_SLOT_INFO::manufacturerID: スロット製造者名: なし(32 バイトの空白文字列) CK_SLOT_INFO::flags: ビットフラグ: カード有無 (カードがあれば CKF_TOKEN_PRESENT) CK_SLOT_INFO::hardwareVersion: スロットハードウェアバージョン: 0.0 CK_SLOT_INFO::firmwareVersion: スロットファームウェアバージョン: 0.0		

(7) C_GetTokenInfo

API 名	C_GetTokenInfo		
概要	トークン情報を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetTokenInfo) (CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID
	CK_TOKEN_INFO_PTR	OUT	トークン情報ポインタ

備考	取得可能なトークン情報は以下の通り。 CK_TOKEN_INFO::label：ラベル名： 「HPKI Application」：カードにラベル有りの場合 なし(32 バイトの空白文字列)：カードにラベル無しの場合 CK_TOKEN_INFO::model：デバイスモデル名： 「ISO 7816-15:2016」注 CK_TOKEN_INFO::flags：ビットフラグ： 乱数生成器有(CKF_RNG)｜ ユーザログイン要(CKF_LOGIN_REQUIRED)｜ ユーザ PIN の初期化有(CKF_USER_PIN_INITIALIZED) CK_TOKEN_INFO::ulMaxPinLen：PIN の最大長： 16 CK_TOKEN_INFO::ulMinPinLen：PIN の最小長： 4 値は、附属書Ｂの構造例を用いた場合の例である。

注：旧版に対応したライブラリからは「JIS X 6320-15:2006」、「ISO 7816-15:2004」が返されるが、後方互換性のためエラーとしないこと。

(8) C_GetMechanismList

API 名	C_GetMechanismList		
概要	サポートメカニズム(アルゴリズム)を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismList) (CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList, CK_ULONG_PTR pulCount) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID
	CK_MECHANISM_TYPE_PTR	OUT	メカニズムタイプポインタ
	CK_ULONG_PTR	IN/OUT	メカニズムタイプ件数
備考	取得可能なサポートメカニズムは以下の通り。 Sign 用: CKM_RSA_PKCS		

(9) C_GetMechanismInfo

API 名	C_GetMechanismInfo		
概要	メカニズム(アルゴリズム)情報を返す。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismInfo) (CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID
	CK_MECHANISM_TYPE	IN	メカニズムタイプ

	CK_MECHANISM_INFO_PTR	IN/OUT	メカニズム情報ポインタ
備考	設定する情報は以下の通り。 type = CKM_RSA_PKCS の場合 CK_MECHANISM_INFO::ulMinKeySize： 最小鍵長: 2048 CK_MECHANISM_INFO::ulMaxKeySize： 最大鍵長: 2048 CK_MECHANISM_INFO::flags： ビットフラグ ： C_SignInit 利用可能(CKF_SIGN) 値は、附属書Ｂの構造例を用いた場合の例である。（鍵長：4096 も可）		

(10) C_OpenSession

API 名	C_OpenSession		
概要	セッションを確立する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_OpenSession) (CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication, CK_NOTIFY Notify, CK_SESSION_HANDLE_PTR phSession) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID
	CK_FLAGS	IN	CKF_SERIAL_SESSION を指定
	CK_VOID_PTR	IN	NULL ポインタを指定
	CK_NOTIFY	IN	NULL ポインタを指定
	CK_SESSION_HANDLE_PTR	IN/OUT	セッションハンドルポインタ

(11) C_CloseSession

API 名	C_CloseSession		
概要	セッションを切断する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_CloseSession) (CK_SESSION_HANDLE hSession) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル

(12) C_CloseAllSessions

API 名	C_CloseAllSessions
概要	すべてのセッションを切断する。
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_CloseAllSessions) (CK_SLOT_ID slotID) ;
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)

	型	I/O	内容
引数	CK_SLOT_ID	IN	スロット ID

(13) C_GetSessionInfo

API 名	C_GetSessionInfo		
概要	セッション状態を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetSessionInfo) (CK_SESSION_HANDLE hSession, CK_SESSION_INFO_PTR pInfo) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_SESSION_INFO_PTR	OUT	セッション状態ポインタ
備考	以下の状態を返す。 ログインしていないとき: CKS_RO_PUBLIC_SESSION (CKS_RW_PUBLIC_SESSION でも可) ログインしているとき: CKS_RO_USER_FUNCTIONS (CKS_RW_USER_FUNCTIONS でも可)		

(14) C_Login

API 名	C_Login		
概要	トークンをログイン状態にする。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_Login) (CK_SESSION_HANDLE hSession, CK_USER_TYPE userType, CK_UTF8CHAR_PTR pPin, CK_ULONG ulPinLen) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_USER_TYPE	IN	CKU_USER を指定
	CK_UTF8CHAR_PTR	IN	パスワード文字列ポインタ。文字列は、ASCII とする。
	CK_ULONG	IN	パスワード文字列長

(15) C_Logout

API 名	C_Logout		
概要	トークンをログアウト状態にする。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_Logout) (CK_SESSION_HANDLE hSession) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		

	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル

(16) C_FindObjectsInit

API 名	C_FindObjectsInit		
概要	オブジェクトの検索を開始する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsInit) (CK_SESSION_HANDLE hSession, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_ATTRIBUTE_PTR	IN	属性テーブルポインタ
	CK_ULONG	IN	属性テーブル数
備考	以下の属性による検索を行う。 CKA_CLASS CKO_CERTIFICATE または CKO_PRIVATE_KEY CKA_TOKEN True CKA_LABEL 証明書の名前 または 私有鍵の名前 CKA_ID 証明書の番号 または 私有鍵の番号 CKA_CERTIFICATE_TYPE CKC_X_509 CKA_VALUE 証明書の値 CKA_KEY_TYPE CKK_RSA CKA_MODULUS 公開鍵の N CKA_PUBLIC_EXPONENT 公開鍵の E		

(17) C_FindObjects

API 名	C_FindObjects		
概要	オブジェクトの検索を行う。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_FindObjects) (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE_PTR phObject, CK_ULONG ulMaxObjectCount, CK_ULONG_PTR pulObjectCount) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_OBJECT_HANDLE_PTR	IN/OUT	オブジェクトハンドルポインタ
	CK_ULONG	IN	最大オブジェクト数
	CK_ULONG_PTR	IN/OUT	オブジェクト数ポインタ

(18) C_FindObjectsFinal

API 名	C_FindObjectsFinal
概要	オブジェクトの検索を終了する。

関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsFinal) (CK_SESSION_HANDLE hSession) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル

(19) C_GetAttributeValue

API 名	C_GetAttributeValue		
概要	オブジェクトの属性値を取得する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_GetAttributeValue) (CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject, CK_ATTRIBUTE_PTR pTemplate, CK_ULONG ulCount) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_OBJECT_HANDLE	IN	オブジェクトハンドル
	CK_ATTRIBUTE_PTR	OUT	属性テーブルポインタ
	CK_ULONG	IN	属性テーブル数
備考	以下の属性に対する値が取得可能である。 CKA_CLASS CKO_CERTIFICATE または CKO_PRIVATE_KEY CKA_TOKEN True CKA_PRIVATE 表３ オブジェクト属性一覧参照。 CKA_LABEL 表３ オブジェクト属性一覧参照。 CKA_ID 表３ オブジェクト属性一覧参照。 CKA_CERTIFICATE_TYPE CKC_X_509 CKA_VALUE 表３ オブジェクト属性一覧参照。 CKA_ISSUER 表３ オブジェクト属性一覧参照。 CKA_SERIAL_NUMBER 表３ オブジェクト属性一覧参照。 CKA_SUBJECT 表３ オブジェクト属性一覧参照。 CKA_KEY_TYPE CKK_RSA CKA_SIGN True CKA_EXTRACTABLE False CKA_ALWAYS_AUTHENTICATE 表３ オブジェクト属性一覧参照。 CKA_MODULUS 表３ オブジェクト属性一覧参照。 CKA_PUBLIC_EXPONENT 表３ オブジェクト属性一覧参照。 値は、附属書Ｂの構造例を用いた場合の例である。		

表３ オブジェクト属性一覧

#	オブジェクト	属性名	属性値
1	私有鍵	CKA_PRIVATE	True
		CKA_LABEL	Private key of HPKI

		CKA_ID	17H
		CKA_ALWAYS_AUTHENTICATE	EF.PrKD 内の CommonObjectAttributes.userConsent の値が設定されている時 True、userConsent 省略時 False
		CKA_MODULUS	エンドエンティティの証明書から取得した値
		CKA_PUBLIC_EXPONENT	エンドエンティティの証明書から取得した値
2	エンドエンティティの証明書	CKA_PRIVATE	False
		CKA_LABEL	HPKI END ENTITY CERTIFICATE
		CKA_ID	17H
		CKA_VALUE	エンドエンティティの証明書の値
		CKA_ISSUER	EF.CD 内のエンドエンティティの証明書の X509CertificateAttributes.issuer の値、issuer 省略時は NULL ポインタ
		CKA_SERIAL_NUMBER	EF.CD 内のエンドエンティティの証明書の X509CertificateAttributes. serialNumber の値、serialNumber 省略時は NULL ポインタ
		CKA_SUBJECT	EF.CD 内のエンドエンティティの証明書の X509CertificateAttributes.subject の値、subject 省略時は NULL ポインタ
3	厚労省の CA 証明書	CKA_PRIVATE	False
		CKA_LABEL	MHLW CA CERTIFICATE
		CKA_ID	19H
		CKA_VALUE	厚労省の CA 証明書の値
		CKA_ISSUER	EF.CD 内の厚労省の CA 証明書の X509CertificateAttributes.issuer の値、issuer 省略時は NULL ポインタ
		CKA_SERIAL_NUMBER	EF.CD 内の厚労省の CA 証明書の X509CertificateAttributes. serialNumber の値、serialNumber 省略時は NULL ポインタ
		CKA_SUBJECT	EF.CD 内の厚労省の CA 証明書の X509CertificateAttributes.subject の値、subject 省略時は NULL ポインタ
4	各事業者の CA 証明書	CKA_PRIVATE	False
		CKA_LABEL	HPKI ROOT CA CERTIFICATE
		CKA_ID	1AH
		CKA_VALUE	各事業者の CA 証明書の値
		CKA_ISSUER	EF.CD 内の各事業者の CA 証明書の X509CertificateAttributes.issuer の値、issuer 省略時は NULL ポインタ
		CKA_SERIAL_NUMBER	EF.CD 内の各事業者の CA 証明書の X509CertificateAttributes. serialNumber の値、serialNumber 省略時は NULL ポインタ
		CKA_SUBJECT	EF.CD 内の各事業者の CA 証明書の X509CertificateAttributes.subject の値、subject 省略時は NULL ポインタ

5	各事業者の中間 CA 証明書	CKA_PRIVATE	False
		CKA_LABEL	HPKI CA CERTIFICATE
		CKA_ID	1BH
		CKA_VALUE	各事業者の中間 CA 証明書の値
		CKA_ISSUER	EF.CD 内の各事業者の中間 CA 証明書の X509CertificateAttributes.issuer の値、issuer 省略時は NULL ポインタ
		CKA_SERIAL_NUMBER	EF.CD 内の各事業者の中間 CA 証明書の X509CertificateAttributes. serialNumber の値、serialNumber 省略時は NULL ポインタ
		CKA_SUBJECT	EF.CD 内の各事業者の中間 CA 証明書の X509CertificateAttributes.subject の値、subject 省略時は NULL ポインタ

注： 値は、附属書Bの構造例を用いた場合の例である。

(20) C_SignInit

API 名	C_SignInit		
概要	署名処理を初期化する。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_SignInit) (CK_SESSION_HANDLE hSession, CK_MECHANISM_PTR pMechanism, CK_OBJECT_HANDLE hKey) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル
	CK_MECHANISM_PTR	IN	メカニズム情報ポインタ mechanism: CKM_RSA_PKCS のみ指定可
	CK_OBJECT_HANDLE	IN	オブジェクトハンドル

(21) C_Sign

API 名	C_Sign		
概要	データに署名を行う。		
関数インタフェース	CK_DEFINE_FUNCTION(CK_RV, C_Sign) (CK_SESSION_HANDLEhSession, CK_BYTE_PTR pData, CK_ULONG ulDataLen, CK_BYTE_PTR pSignature, CK_ULONG_PTR pulSignatureLen) ;		
戻り値	CK_RV (CKR_OK: 成功 その他: 失敗)		
	型	I/O	内容
引数	CK_SESSION_HANDLE	IN	セッションハンドル

	CK_BYTE_PTR	IN	データポインタ
	CK_ULONG	IN	データ長
	CK_BYTE_PTR	IN/OUT	署名データポインタ
	CK_ULONG_PTR	IN/OUT	署名データ長ポインタ

表４　PKCS #11 の型一覧

NO	型名	概要
1	CK_INFO CK_INFO_PTR	PKCS #11 ライブラリ情報
2	CK_SLOT_ID CK_SLOT_ID_PTR	スロット ID
3	CK_SLOT_INFO CK_SLOT_INFO_PTR	スロット情報
4	CK_TOKEN_INFO CK_TOKEN_INFO_PTR	トークン情報
5	CK_SESSION_HANDLE CK_SESSION_HANDLE_PTR	セッションハンドル
6	CK_USER_TYPE	ユーザタイプ
7	CK_SESSION_INFO CK_SESSION_INFO_PTR	セッション情報
8	CK_OBJECT_HANDLE CK_OBJECT_HANDLE_PTR	オブジェクトハンドル
9	CK_ATTRIBUTE CK_ATTRIBUTE_PTR	属性タイプ、値、長さを含む構造体
10	CK_MECHANISM_TYPE CK_MECHANISM_TYPE_PTR	メカニズムタイプ
11	CK_MECHANISM CK_MECHANISM_PTR	メカニズムタイプを含む、メカニズムを示す構造体
12	CK_MECHANISM_INFO CK_MECHANISM_INFO_PTR	メカニズム情報
13	CK_RV	ライブラリの戻り値
14	CK_NOTIFY	コールバック情報
15	CK_FUNCTION_LIST CK_FUNCTION_LIST_PTR CK_FUNCTION_LIST_PTR_PTR	PKCS #11 ライブラリのバージョン、関数ポインタを含む構造体

5.2.3. Crypto API インタフェース

本ガイドラインでは、Crypto API のサポートすべき機能の一覧を表 5 に示す。

表 5　Crypto API

No	API 名	概要
1	CryptAcquireContext	鍵コンテナのハンドルを生成する
2	CryptReleaseContext	鍵コンテナのハンドルを解放する。
3	CryptGetProvParam	CSP のパラメータの値を取得する。
4	CryptSetProvParam	CSP のパラメータの値を設定する。
5	CryptDestroyKey	鍵の破棄を行う。
6	CryptGetKeyParam	鍵のパラメータの値を取得する。
7	CryptGetUserKey	鍵コンテナ内の鍵ハンドルを取得する。
8	CryptCreateHash	ハッシュオブジェクトの生成を行う。
9	CryptDestroyHash	ハッシュオブジェクトの破棄を行う。
10	CryptSetHashParam	ハッシュオブジェクトのパラメータを設定する。
11	CryptSignHash	ハッシュ値に署名を行う。

サポートすべき API の仕様は以下の通りとなる。なお、戻り値、構造体は、CryptoAPI の仕様を参照のこと。

(1) CryptAcquireContext

API 名	CryptAcquireContext		
概要	鍵コンテナのハンドルを生成する		
関数インタフェース	CryptAcquireContext(HCRYPTPROV* phProv, LPCTSTR pszContainer, LPCTSTR pszProvider, DWORD dwProvType, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV*	OUT	ハンドル格納場所のポインタ
	LPCTSTR	IN	NULL 文字で終了するコンテナ名称 NULL を設定すること。
	LPCTSTR	IN	NULL 文字で終了する CSP 名称であって、電子署名用 CSP ならば"HPKI Crypto Service Provider for Non Repudiation"、電子認証用 CSP ならば "HPKI Crypto Service Provider for Authentication" で始まる文字列を設定する。[2]

[2] CSP の名称は、"HPKI Crypto Service Provider for Non Repudiation ABC"等プロバイダーが提供するものである。現状で唯一の名称にする制度はないので、配慮が必要である。

	DWORD	IN	プロバイダタイプ PROV_RSA_AES、PROV_RSA_FULL あるいは PROV_RSA_SIG を設定すること。
	DWORD	IN	動作に関するパラメータ 0: 利用者証明書、利用者私有鍵使用時に設定する。
備考	dwFlags に 0 を指定した場合、パスワード入力画面が表示される。 CRYPT_SILENT を設定した場合ユーザインタフェースは表示されないので、CryptSetProvParam で PP_SIGNATURE_PIN を指定して PIN の情報を渡す必要がある。		

(2) CryptReleaseContext

API 名	CryptReleaseContext		
概要	鍵コンテナのハンドルを解放する		
関数インタフェース	CryptReleaseContext(HCRYPTPROV phProv, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV	IN	CryptAcquireContext で取得したハンドル
	DWORD	IN	動作に関するパラメータ 0を設定すること。

(3) CryptGetProvParam

API 名	CryptGetProvParam		
概要	CSP のパラメータの値を取得する。		
関数インタフェース	CryptGetProvParam(HCRYPTPROV hProv, DWORD dwParam, BYTE* pbData, DWORD* pdwDataLen, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV	IN	CryptAcquireContext で取得したハンドル
	DWORD	IN	値を取得するためのパラメータ。サポートする値を表 6 に示す
	BYTE *	OUT	値を格納するバッファのポインタ NULL を指定した場合は値の読み込みは行われず、pdwDataLen に値の格納に必要な長さが設定される。

	DWORD*	IN/OUT	値の長さを保持するバッファへのポインタ 関数呼び出し時には、pbData バッファに割り当てられたメモリサイズを設定する。関数終了時には、値の格納に必要な長さが設定される。
	DWORD	IN	動作に関するパラメータ 0 を設定すること。

表 6　CryptGetProvParam の dwParam でサポートする値

#	値	内容
1	PP_IMPTYPE	CSP の実装形を DWORD 型で返す。
2	PP_NAME	CSP 名称を CHAR 型の NULL ターミネート文字列で返す。 電子署名用 CSP ならば"HPKI Crypto Service Provider for Non Repudiation"、電子認証用 CSP ならば"HPKI Crypto Service Provider for Authentication"を返す。
3	PP_VERSION	CSP のバージョンを DWORD 型で返す。
4	PP_PROVTYPE	CSP のプロバイダタイプを DWORD 型で返す。
5	PP_CERTCHAIN	CSP に含まれる CA の証明書（チェイン）を取得する。

注：CA の証明書のチェインの順番は特に規定しない

CA のフォーマットは、ASN.1 でエンコードされた

```
SET OF Certificate ; -- X.509 certificates
```

に従うものとする。

(4) CryptSetProvParam

API 名	CryptSetProvParam		
概要	CSP のパラメータの値を設定する。		
関数インタフェース	CryptGetProvParam(HCRYPTPROV hProv, DWORD dwParam, const BYTE* pbData, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV	IN	CryptAcquireContext で取得したハンドル
	DWORD	IN	値を取得するためのパラメータ サポートする値を表 7 に示す
	const BYTE *	IN	値を格納するバッファのポインタ。呼ぶ前に値を設定する。
	DWORD	IN	動作に関するパラメータ 0 を設定すること。

表 7　CryptSetProvParam の dwParam でサポートする値

#	値	内容
1	PP_SIGNATUER_PIN	CSP を署名で利用するための PIN を設定する。pbData には、NULL で終わる ASCII 文字列を渡す。

CryptAquireContext で CRYPT_SILENT を設定した場合、PP_SIGNATURE_PIN を指定して PIN の情報を渡す必要がある。指定しない場合には、CSP 内のユーザインタフェースによって PIN の入力が促される。

(5) CryptDestroyKey

API 名	CryptDestroyKey		
概要	鍵の破棄を行う。		
関数インタフェース	CryptDestroyKey(HCRYPTKEY hKey);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTKEY	IN	破棄する鍵のハンドル

(6) CryptGetKeyParam

API 名	CryptGetKeyParam		
概要	鍵のパラメータの値を取得する。(IC カードに格納された利用者証明書(DER 形式)を返す。)		
関数インタフェース	CryptGetKeyParam(HCRYPTKEY hKey, DWORD dwParam, BYTE* pbData, DWORD* pdwDataLen, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTKEY	IN	値を取得する鍵のハンドル
	DWORD	IN	値を取得するパラメータ KP_CERTIFICATE: IC カードに格納された利用者証明書を返す。
	BYTE*	OUT	値を格納するバッファのポインタ NULL を指定した場合は値の読み込みは行われず、pdwDataLen に値の格納に必要な長さが設定される。

	DWORD*	IN/OUT	値の長さを保持するバッファへのポインタ 関数呼び出し時には、pbData バッファに割り当てられたメモリサイズを設定する。関数終了時には、値の格納に必要な長さが設定される。
	DWORD	IN	動作に関するパラメータ 0 を設定すること。

(7) CryptGetUserKey

API 名	CryptGetUserKey		
概要	鍵コンテナ内の鍵ハンドルを取得する		
関数インタフェース	CryptGetUserKey(HCRYPTPROV hProv, DWORD dwKeySpec, HCRYPTKEY* phUserKey);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV	IN	CryptAcquireContext で取得したハンドル
	DWORD	IN	鍵の種類 AT_SIGNATURE: 署名用鍵を設定すること。
	HCRYPTKEY*	OUT	鍵ハンドルをコピーするバッファのアドレス

(8) CryptCreateHash

API 名	CryptCreateHash		
概要	ハッシュオブジェクトの生成を行う。		
関数インタフェース	CryptCreateHash(HCRYPTPROV hProv, ALG_ID Algid, HCRYPTKEY hKey, DWORD dwFlags, HCRYPTHASH* phHash);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTPROV	IN	CryptAcquireContext で取得したハンドル

	ALG_ID	IN	CALG_SHA1: SHA-1 アルゴリズム[注] CALG_SHA_256: SHA-256 アルゴリズム CALG_SHA_384: SHA-384 アルゴリズム CALG_SHA_512: SHA-512 アルゴリズム
	HCRYPTKEY	IN	キードハッシュの場合の鍵ハンドル 0を設定すること。(本 CSP ではキードハッシュはサポートしない
	DWORD	IN	動作に関するパラメータ 0を設定すること。
	HCRYPTHASH*	OUT	生成したハッシュオブジェクトのハンドルをコピーするバッファのポインタ

注： SHA-1 アルゴリズムは「電子政府における調達のために参照すべき暗号のリスト（CRYPTREC 暗号リスト)」では運用監視暗号リストに掲載されているので、互換性維持以外の目的での利用は推奨されていない点に留意する必要がある。

(9) CryptDestroyHash

API 名	CryptDestroyHash		
概要	ハッシュオブジェクトの破棄を行う。		
関数インタフェース	CryptDestroyHash(HCRYPTHASH hHash);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTHASH	IN	破棄するハッシュオブジェクトのハンドル

(10) CryptSetHashParam

API 名	CryptSetHashParam		
概要	ハッシュオブジェクトのパラメータを設定する。		
関数インタフェース	CryptSetHashParam(HCRYPTHASH hHash, DWORD dwParam, const BYTE* pbData, DWORD dwFlags);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTHASH	IN	パラメータを設定するハッシュオブジェクトのハンドル

	DWORD	IN	設定するパラメータ HP_HASHVAL: pbData バッファに格納された値をハッシュ値としてハッシュオブジェクトに設定する。
	const BYTE*	IN	パラメータに設定するデータのポインタ
	DWORD	IN	動作に関するパラメータ 0を設定すること。

(11)CryptSignHash

API 名	CryptSignHash		
概要	ハッシュ値に署名を行う。		
関数インタフェース	CryptSignHash(HCRYPTHASH hHash, DWORD dwKeySpec, LPCTSTR sDescription, DWORD dwFlags, BYTE* pbSignature, DWORD* pdwSigLen);		
戻り値	BOOL（TRUE:成功 FALSE:失敗） GetLastError にてエラー情報が取得できる。		
	型	I/O	内容
引数	HCRYPTHASH	IN	署名を行うハッシュオブジェクトのハンドル
	DWORD	IN	鍵の種類 署名用鍵 AT_SIGNATURE を設定すること。
	LPCTSTR	IN	ハッシュの概要についての NULL ターミネート文字列 NULL を設定すること
	DWORD	IN	署名時のフラグ 0 を設定すること。
	BYTE*	OUT	署名データを格納するバッファのポインタ NULL を指定した場合は値の書き込みは行われず、pdwSigLen に値の格納に必要な長さが設定される。
	DWORD*	IN/OUT	署名データの長さを保持するバッファへのポインタ 関数呼び出し時には、pbSignature バッファに割り当てられたメモリサイズを設定する。関数終了時には、署名データの格納に必要な長さが設定される。

5.2.4. Cryptography API: Next Generation インタフェース

本ガイドラインでは、Cryptography API: Next Generation (以下 CNG) のサポートすべき機能の一覧を表 8 に示す。

表 8　CNG

No	API 名	概要
1	NCryptOpenStorageProvider	KSP をロードし初期化する。
2	NCryptOpenKey	KSP に存在する鍵のハンドルを取得する。
3	NCryptGetProperty	KSP のプロパティの値を取得する。
4	NCryptSetProperty	KSP のプロパティの値を設定する。
5	NCryptFreeObject	KSP のオブジェクトを解放する。
6	NCryptFreeBuffer	NCryptEnumAlgorithms から取得したメモリを解放する。
7	NCryptIsAlgSupported	アルゴリズムのサポート状況を取得する。
8	NCryptEnumAlgorithms	サポートするアルゴリズムの一覧を取得する。
9	NCryptExportKey	鍵をエクスポートする。
10	NCryptSignHash	ハッシュ値に署名を行う。

サポートすべき API の仕様は以下の通りとなる。なお、戻り値、構造体は、CNG の仕様を参照のこと。

(1) NCryptOpenStorageProvider

API 名	NCryptOpenStorageProvider		
概要	KSP をロードし初期化する。		
関数インタフェース	NCryptOpenStorageProvider(NCRYPT_PROV_HANDLE *phProvider, LPCWSTR pszProviderName, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_PROV_HANDLE *	OUT	プロバイダハンドル格納場所のポインタ
	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列のプロバイダ名称であって、電子署名用プロバイダならば“HPKI Key Storage Provider for Non Repudiation”、電子認証用プロバイダならば“HPKI Key Storage Provider for Authentication”で始まる文字列を設定する。[注3]
	DWORD	IN	動作に関するパラメータ 0 を設定する。

(2) NCryptOpenKey

API 名	NCryptOpenKey
概要	KSP に存在する鍵のハンドルを取得する。

[注3] プロバイダの名称は、“HPKI Key Storage Provider for Non Repudiation ABC”等プロバイダが提供するものである。現状で唯一の名称にする制度はないので、配慮が必要である。

関数インタフェース	NCryptOpenKey(NCRYPT_PROV_HANDLE hProvider, NCRYPT_KEY_HANDLE *phKey, LPCWSTR pszKeyName, DWORD dwLegacyKeySpec, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_PROV_HANDLE	IN	NCryptOpenStorageProvider で取得したハンドル
	NCRYPT_KEY_HANDLE *	OUT	鍵ハンドル格納場所のポインタ
	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列の鍵の名称 “Private key of HPKI”を設定する。
	DWORD	IN	CSP との互換性のために使用する鍵の種類 AT_SIGNATURE または 0 を設定する。
	DWORD	IN	動作に関するパラメータ 0: 必要に応じてユーザインタフェースを表示 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

(3) NCryptGetProperty

API 名	NCryptGetProperty		
概要	KSP のプロパティの値を取得する。		
関数インタフェース	NCryptGetProperty(NCRYPT_HANDLE hObject, LPCWSTR pszProperty, PBYTE pbOutput, DWORD cbOutput, DWORD *pcbResult, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_HANDLE	IN	プロバイダまたは鍵のハンドル
	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列のプロパティの名称。サポートする値を表 9 に示す。
	PBYTE	OUT	プロパティの値の格納場所のポインタ。格納に必要なサイズを計算する場合には NULL を設定する。
	DWORD	IN	pbOutput のサイズ

	DWORD *	OUT	プロパティの値のサイズ
	DWORD	IN	動作に関するパラメータ 0: 必要に応じてユーザインタフェースを表示 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

表 9 NCryptGetProperty の pszProperty でサポートする値

#	値	内容
1	NCRYPT_NAME_PROPERTY	プロバイダまたは鍵の名称を返す。
2	NCRYPT_VERSION_PROPERTY	プロバイダのバージョンを返す。
3	NCRYPT_ALGORITHM_GROUP_PROPERTY	NCRYPT_RSA_ALGORITHM_GROUP を返す。
4	NCRYPT_ALGORITHM_PROPERTY	NCRYPT_RSA_ALGORITHM を返す。
5	NCRYPT_CERTIFICATE_PROPERTY	鍵に関連付く証明書を取得する。
6	NCRYPT_LENGTH_PROPERTY	鍵の bit 数を取得する。
7	NCRYPT_PROVIDER_HANDLE_PROPERTY	指定された鍵ハンドルのプロバイダハンドルを取得する。
8	NCRYPT_WINDOW_HANDLE_PROPERTY	NCryptSetProperty で設定された Window ハンドルを取得する。

(4) NCryptSetProperty

API 名	NCryptSetProperty		
概要	KSP のプロパティの値を設定する。		
関数インタフェース	NCryptSetProperty(NCRYPT_HANDLE hObject, LPCWSTR pszProperty, PBYTE pbInput, DWORD cbInput, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_HANDLE	IN	プロバイダまたは鍵のハンドル
	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列のプロパティの名称。サポートする値を表 10 に示す。
	PBYTE	IN	設定するプロパティの値
	DWORD	IN	pbInput のサイズ (バイト)
	DWORD	IN	動作に関するパラメータ(複数指定可) NCRYPT_PERSIST_ONLY_FLAG: ユーザに保持されるプロパティのみを設定する。 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

表 10　NCryptGetProperty の pszProperty でサポートする値

#	値	内容
1	NCRYPT_WINDOW_HANDLE_PROPERTY	ダイアログを表示する際の親 Window のハンドルを設定する。
2	NCRYPT_PIN_PROPERTY	鍵を使用するための PIN を設定する。 dwFlags に NCRYPT_SILENT_FLAGS が設定された関数において PIN を必要とする場合、本プロパティが設定されていないとエラーになり、戻り値として NTE_SILENT_CONTEXT が返る。

(5) NCryptFreeObject

API 名	NCryptFreeObject		
概要	KSP のオブジェクトを解放する。		
関数インタフェース	NCryptFreeObject(NCRYPT_HANDLE hObject);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_HANDLE	IN	プロバイダまたは鍵のハンドル

(6) NCryptFreeBuffer

API 名	NCryptFreeBuffer		
概要	NCryptEnumAlgorithms から取得したメモリを解放する。		
関数インタフェース	NCryptFreeBuffer(PVOID pvInput);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	PVOID	IN	解放するメモリのアドレス

(7) NCryptIsAlgSupported

API 名	NCryptIsAlgSupported		
概要	アルゴリズムのサポート状況を取得する。		
関数インタフェース	NCryptIsAlgSupported(NCRYPT_PROV_HANDLE hProvider, LPCWSTR pszAlgId, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_PROV_HANDLE	IN	NCryptOpenStorageProvider で取得したハンドル

	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列のアルゴリズム識別子の名称 NCRYPT_RSA_ALGORITHM が有効な識別子
	DWORD	IN	動作に関するパラメータ 0: 必要に応じてユーザインタフェースを表示 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

(8) NCryptEnumAlgorithms

API 名	NCryptEnumAlgorithms		
概要	サポートするアルゴリズムの一覧を取得する。		
関数インタフェース	NCryptEnumAlgorithms(NCRYPT_PROV_HANDLE hProvider, DWORD dwAlgOperations, DWORD *pdwAlgCount, NCryptAlgorithmName **ppAlgList, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_PROV_HANDLE	IN	NCryptOpenStorageProvider で取得したハンドル
	DWORD	IN	一覧を取得するアルゴリズムの種別
	DWORD *	OUT	アルゴリズムの数の格納場所のポインタ
	NCryptAlgorithmName **	OUT	アルゴリズムの一覧が格納される構造体のポインタのアドレス アルゴリズムの一覧が格納されたメモリは NCryptFreeBuffer を使用して解放する。
	DWORD	IN	動作に関するパラメータ 0: 必要に応じてユーザインタフェースを表示 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

(9) NCryptExportKey

API 名	NCryptExportKey
概要	鍵をエクスポートする

関数インタフェース	NCryptExportKey(NCRYPT_KEY_HANDLE hKey, NCRYPT_KEY_HANDLE hExportKey, LPCWSTR pszBlobType, NCryptBufferDesc *pParameterList, PBYTE pbOutput, DWORD cbOutput, DWORD *pcbResult, DWORD dwFlags);		
戻り値	SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)		
	型	I/O	内容
引数	NCRYPT_KEY_HANDLE	IN	エクスポートする鍵ハンドル
	NCRYPT_KEY_HANDLE	IN	エクスポートする鍵を暗号化するための鍵ハンドル NULL を設定する。
	LPCWSTR	IN	NULL 文字で終了する Unicode 文字列のエクスポートする BLOB タイプ BCRYPT_PUBLIC_KEY_BLOB: BCRYPT_RSAKEY_BLOB 構造体として公開鍵をエクスポートする。 BCRYPT_RSAPUBLIC_BLOB: BCRYPT_RSAKEY_BLOB 構造体として RSA の公開鍵をエクスポートする。 LEGACY_RSAPUBLIC_BLOB: CryptoAPI でインポート可能な形式の RSA 公開鍵をエクスポートする。
	NCryptBufferDesc *	IN	鍵のパラメータ情報 NULL を設定する。
	PBYTE	OUT	エクスポートする鍵の格納場所のポインタ エクスポートする鍵のサイズを計算する場合には NULL を設定する。
	DWORD	IN	pbOutput のサイズ
	DWORD *	OUT	エクスポートする鍵のサイズ
	DWORD	IN	動作に関するパラメータ 0: 必要に応じてユーザインタフェースを表示 NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない

(10) NCryptSignHash

API 名	NCryptSignHash
概要	ハッシュ値に署名を行う

<table>
<tr><td>関数インタフェース</td><td colspan="3">NCryptSignHash(
NCRYPT_KEY_HANDLE hKey,
VOID *pPaddingInfo,
PBYTE pbHashValue,
DWORD cbHashValue,
PBYTE pbSignature,
DWORD cbSignature,
DWORD *pcbResult,
DWORD dwFlags
);</td></tr>
<tr><td>戻り値</td><td colspan="3">SECURITY_STATUS (ERROR_SUCCESS:成功 その他:失敗)</td></tr>
<tr><td></td><td>型</td><td>I/O</td><td>内容</td></tr>
<tr><td rowspan="10">引数</td><td>NCRYPT_KEY_HANDLE</td><td>IN</td><td>NCryptOpenKey で取得したハンドル</td></tr>
<tr><td>VOID *</td><td>IN</td><td>パディング情報の構造体のポインタ
dwFlags に指定したパディング情報に応じた構造体を設定する。</td></tr>
<tr><td>PBYTE</td><td>IN</td><td>署名対象のハッシュ値</td></tr>
<tr><td>DWORD</td><td>IN</td><td>署名対象のハッシュ値のサイズ(バイト)</td></tr>
<tr><td>PBYTE</td><td>OUT</td><td>署名値格納場所のポインタ
署名値のサイズを計算する場合には NULL を設定する。</td></tr>
<tr><td>DWORD</td><td>IN</td><td>pbSignature のサイズ</td></tr>
<tr><td>DWORD *</td><td>OUT</td><td>署名値のサイズ</td></tr>
<tr><td>DWORD</td><td>IN</td><td>動作に関するパラメータ
BCRYPT_PAD_PKCS1: PKCS #1 パディングを使用する。pPaddingInfo には BCRYPT_PKCS1_PADDING_INFO を設定する。
BCRYPT_PAD_PSS: PSS パディングを使用する。pPaddingInfo には BCRYPT_PSS_PADDING_INFO を設定する。
NCRYPT_SILENT_FLAG: ユーザインタフェースを表示させない</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>備考</td><td colspan="3">pPaddingInfo に設定するパディング情報の構造体のメンバ pszAlgId には以下のいずれかのアルゴリズムを設定する。
BCRYPT_SHA256_ALGORITHM: SHA-256
BCRYPT_SHA384_ALGORITHM: SHA-384
BCRYPT_SHA512_ALGORITHM: SHA-512</td></tr>
</table>

5.3. PKI アプリケーションの構造

5.3.1. 基本構造

PKI カードアプリケーションのデータ構造は、ISO/IEC 7816-15 で規定されている。本ガイドラインでは、

この仕様を採用することとし、その概要を説明する。具体的なアプリケーションの構造の例を、附属書 B に示す。

ISO/IEC 7816-15 は、IC カードの中に暗号演算を行うための鍵や、証明書を格納する際の仕様を定めている。RSA セキュリティ社が暗号機能の標準として検討した PKCS #15 をベースとしており、どのような暗号情報オブジェクトが、カード内のどこにあり、どのようなアクセス条件になっているのかを記述できる仕様となっている。基本的な構造を図４に示す。

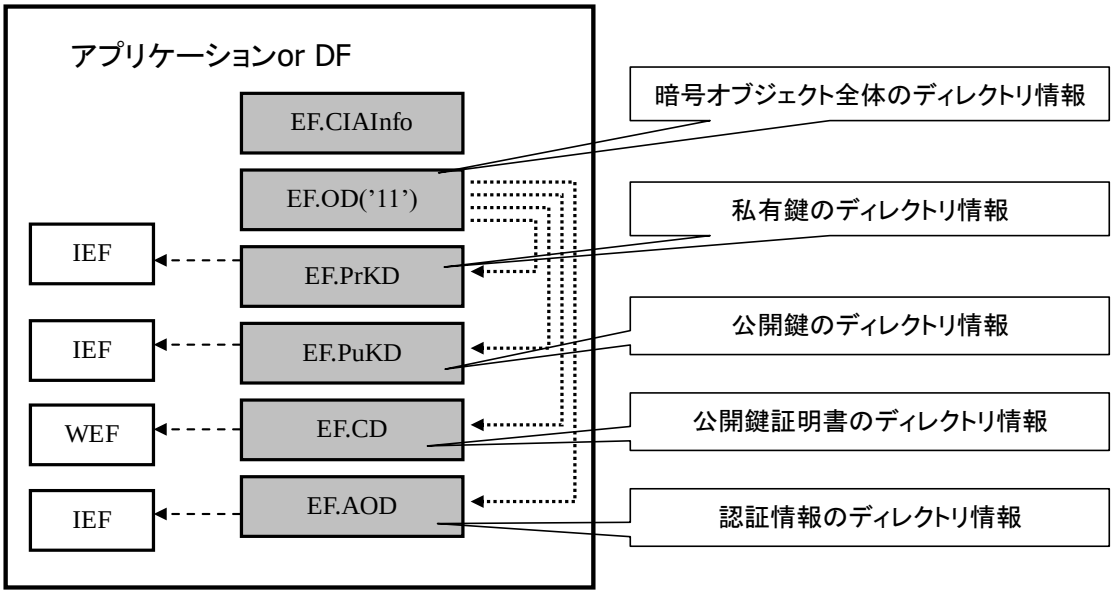


図４　ISO/IEC 7816-15 の基本構成

暗号情報のディレクトリ情報を記述したファイル EF.OD は、各暗号情報オブジェクトのディレクトリ情報へのリンク（参照情報）を保持している。その情報から取得した各暗号情報オブジェクトのディレクトリ情報を取得し、最終的な暗号情報オブジェクトの所在、利用条件（セキュリティ属性）などを取得し、暗号情報オブジェクトを利用することになる。

5.3.2. 電子署名用 PKI カードアプリケーションと電子認証用 PKI カードアプリケーションの共存

電子署名用アプリケーションと電子認証用アプリケーションは、異なる AID によって識別される DF（カードアプリケーション）に格納される。それぞれ独立の EF.CIAInfo を持つ、独立な構造として存在する。HPKI で用いられるカードには、電子署名用 PKI アプリケーションあるいは電子認証用 PKI アプリケーションのいずれか１つ、あるいは両方の PKI アプリケーションが搭載される可能性がある。両 PKI アプリケーションがカードに搭載された場合の構造例を図５に示す。

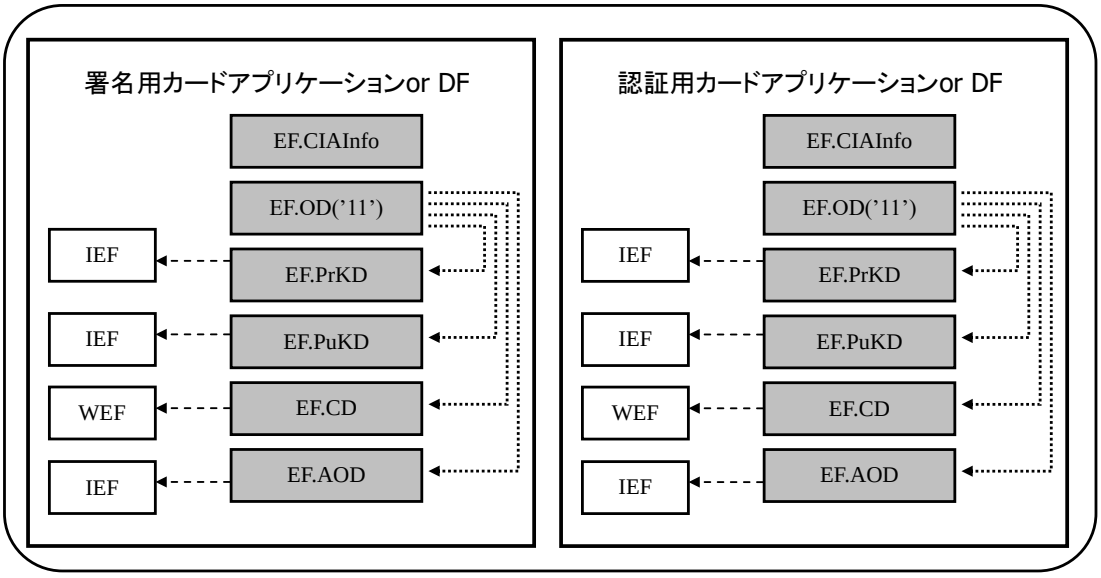


図 5 署名用アプリケーションと認証用アプリケーションを独立にした場合の構造例

5.3.3. アプリケーションが共存する場合の留意事項

実装される IC カードにより、機能の差が生じる可能性がある。特に論理チャネルの取り扱いは、カードによって基本論理チャネルしかサポートしていない場合が存在する。そのため、カードを利用する HPKI アプリケーションは、署名用カードアプリケーションと認証用カードアプリケーションが同じカード上に共存する場合には、電子署名演算が完了するまで排他制御を行うなど、利用シーケンスに混乱が生じないよう配慮する必要がある。

5.4. PKI アプリケーションのコマンド仕様

5.4.1. 概要

5.3.1 で説明した ISO/IEC 7816-15 に基づくデータモデル内の各暗号情報オブジェクト（鍵、証明書等）を利用するためのカードコマンドは、ISO/IEC 7816-4、ISO/IEC 7816-8 によって規定される。本節では、そのコマンドの概要について説明する。

5.4.2. 対象とするコマンド

電子署名に必要となる演算の実行に関連する IC カードに対するコマンドは、ISO/IEC 7816-4 及び ISO/IEC 7816-8 によって規定されている。表 11 に必要となるコマンドの一覧を示す。各コマンドの詳細については、附属書 C に記述する。

表 11　相互運用性確保に必要となるコマンド

コマンド	規定される標準
SELECT	ISO/IEC 7816-4
VERIFY	ISO/IEC 7816-4
READ BINARY	ISO/IEC 7816-4
MANAGE SECURITY ENVIRONMENT	ISO/IEC 7816-4
PERFORM SECURITY OPERATION	ISO/IEC 7816-8

5.4.3. コマンドに対する制限

このガイドラインの適用範囲内の相互運用性を確保するために必要となる各コマンド共通の制限として以下を挙げる

・セキュアメッセージングは使用しない

附属書Ａ（参考）PKI カードアプリケーション利用のシーケンス

A.1　概要

署名のシーケンスは、ISO/IEC 7816-8 の附属書Ａのシーケンスに従うものとする。本附属書では、A.2 で利用シーケンスに関する前提条件を示し、A.3 にカードレベルでの利用シーケンス例を、A.4 に PKCS #11 インタフェースの利用シーケンス例を、A.5 に CryptoAPI の利用シーケンス例を、A.6 に CNG の利用シーケンス例を示す。本附属書で示すシーケンス及び値は、あくまで一例であり、実際の認証事業者が発行するカードの利用の際のシーケンス及び値に一致することを保証するものではない。実際のカード及びカードに搭載されたアプリケーションの利用に当たっては、各事業者のカード及びアプリケーション仕様、提供されるソフトウェアモジュールの仕様に従って動作を確認する必要がある。

A.2　利用シーケンスに関する前提条件及び概要

A.2.1　前提条件

カードレベルでの利用シーケンス、PKCS#11、CAPI および CNG の利用は単一ではなく、複数想定することができる。A.3、A.4、A.5、及び A.6 では、次の条件を前提としている。

1）　端末の構成

端末１台に IC カードリーダ１台が接続され、IC カード１枚が挿入される。

2）　カード上の HPKI カードアプリケーション

１枚のカード上には、１つの電子署名用 HPKI カードアプリケーションまたは１つの電子認証用 HPKI カードアプリケーション、あるいは１枚のカード上に両カードアプリケーションが１つずつ搭載されているものとする。複数の電子署名用 HPKI カードアプリケーション及び複数の電子認証用 HPKI アプリケーションが搭載されている場合は想定しない。

3）　鍵ペア

１つの PKI アプリケーションには、１つの私有鍵と、それに対応した１枚の利用者証明書が格納されているものとする。私有鍵を活性化する PIN は１つ存在するものとする。１つの PKI アプリケーションに複数の私有鍵、複数の利用者証、複数の PIN がある場合は想定しない。

4）　上位証明書の格納

上位証明書については、PKI アプリケーション１つにルート CA 及び中間 CA を含む１組の上位証明書が格納されているものとする。１つの PKI アプリケーションに複数組の上位証明書が格納されている場合は想定しない。

5）　上位証明書の区別

利用する PKI カードアプリケーション内に上位証明書が格納されている場合、証明書の BasicConstraints.cA などによって利用者証明書との区別が可能になっているものとする。

6）　ディレクトリ情報及び公開鍵証明書のアクセス制御

ISO/IEC 7816-15 に従ったディレクトリ情報及び公開鍵証明書の読み出しは、認証しなくとも可能であるものとする。ディレクトリ情報及び公開鍵証明書の取得に認証が必要な場合は想定しない。

A.2.2 および A.2.3 に代表的な２つのカード利用のシーケンスの概略を示す。A.3、A.4、A.5 及び A.6 では、A.2.2 のシーケンスをとった場合の詳細シーケンスを示す。

A.2.2　ISO/IEC 7816-15 に従った利用シーケンスの概要

本ガイドラインに従った PKI カードアプリケーションは、ISO/IEC 7816-15 に従っている。そのため、利用にあたって、カード内の ISO/IEC 7816-15 に従ったカードアプリケーションを検索するシーケンスに従った利用を想定することができる。その場合、以下の条件の下でカード PKI アプリケーションを検索するこ

ととなる。

① PKI カードアプリケーションは、ISO/IEC 7816 に従った AID が付与されている。すなわち、ISO/IEC 7816-15 で規定される RID“E8 28 BD 08 0F”で始まる最長 16 バイトの AID となる。

② カードには、複数の ISO/IEC 7816-15 に準拠したカードアプリケーション（電子署名用カードアプリケーションと電子認証用カードアプリケーション等）が搭載されている可能性がある。

③ HPKI に対応した PKI カードアプリケーションであるか否かは、最終的には証明書を取得して確認する。

以上の条件に従うと、図 A.1 に従った利用シーケンスに従って利用することとなる。

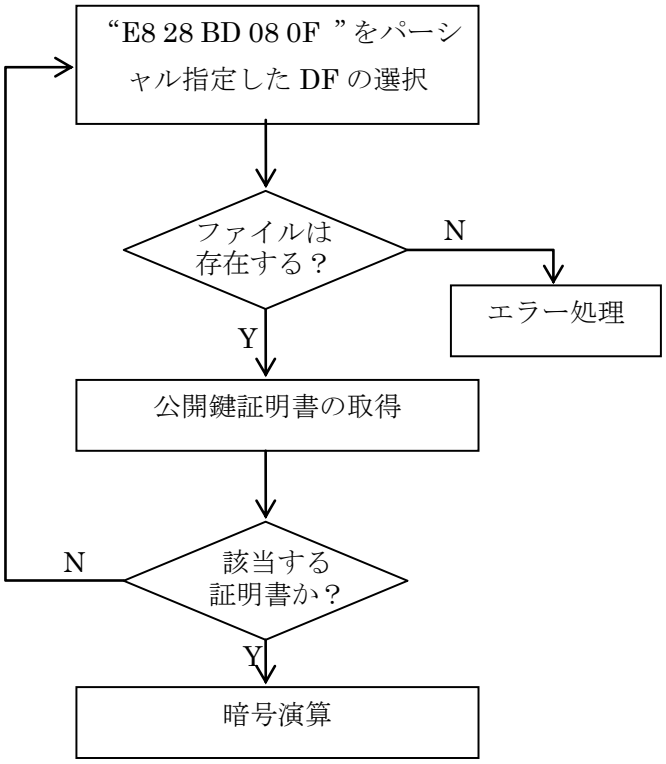


図　A.1　 ISO/IEC 7816-15 に従った PKI カードアプリケーションの検索と利用

A.2.3　対象とするカードアプリケーションの AID を用いた利用シーケンスの概要

カード上アプリケーションを検索するよりも直接 AID を指定して選択したほうが効率が良い。条件は以下の通りとなる。

① PKI カードアプリケーションは、ISO/IEC 7816-15 に従った AID が付与されている。また、HPKI 電子署名用及び電子認証用カードアプリケーションは、厚労省が指定した AID を持つ。

② カードには、複数の ISO/IEC 7816-15 のカードアプリケーション（電子署名用アプリケーションと電子認証用アプリケーション等）が搭載されている可能性がある。

③ HPKI に対応した PKI アプリケーションであるか否かは、最終的には HPKI アプリケーションあるいは医療情報アプリケーションが証明書を取得して確認する。

以上の条件に従うと、図 A.2 に従った利用シーケンスに従って利用することとなる。

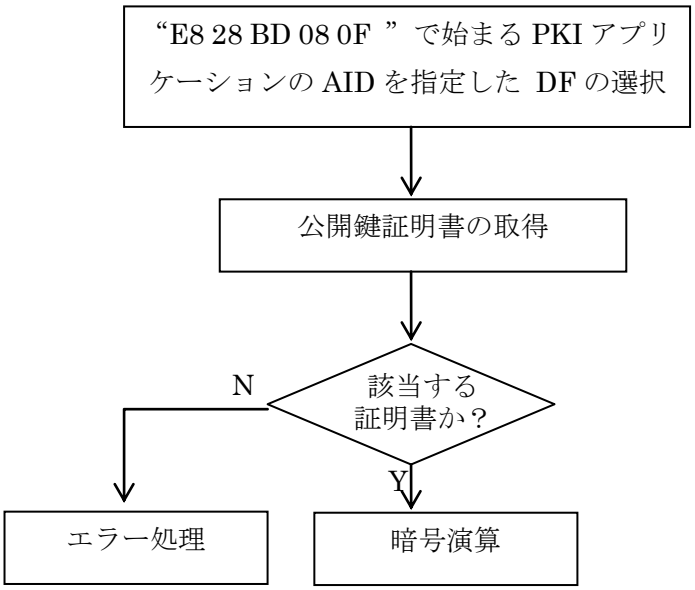


図　A.2　AID を直接指定した PKI アプリケーションの選択と利用

A.3　IC カードの利用のシーケンス

IC カードに対するコマンドレベルでの手順は、以下のシーケンスに従う。

A.3.1　アプリケーション選択

1）アプリケーション/DF の選択

SELECT FILE

・“E8 28 BD 08 0F”から始まる AID を DF 名として指定して選択する。

・（1）コマンド

CLA	“00“
INS	“A4“ = SELECT
P1	“04“ = DF 名選択
P2	“00“ = 最初または唯一のファイル選択
Lc	“XX“ = データフィールドの長さ
データ	“E8 28 BD 08 0FXX XX XX XX XX XX XX XX XX XX XX XX“（注）
Le	“00“ = FCI を受け取る

注： PKI アプリケーションによって指定された値を設定する。

（2）レスポンス

	長さ	意味
データ	5～20	FCI(TAG=“84“) “6F” L1 “84“ L2　xx xx xx xx xx xx xx xx (xx は最長 16 バイト)
SW1	1	
SW2	1	

A.3.2　証明書の読み出し

注：IC カード内の特定の HPKI アプリケーションの証明書を取得する。

1）アプリケーション/DF の選択

SELECT FILE

・事前にアプリケーションを確認しており、選択する DF は事前にわかっているものとする。

（1）コマンド

CLA	“00“
INS	“A4“ = SELECT
P1	“04“ = DF 名選択
P2	“00“ = DF 選択
Lc	“XX“ = データフィールドの長さ(最長 16 バイト)
データ	“E8 28 BD 08 0F XX XX XX XX“
Le	“00“ = FCI を受け取る

（2）レスポンス

	長さ	意味
データ	5～20	FCI (TAG=“84“)テンプレート
SW1	1	
SW2	1	

2）EF.CIAInfo の読み出し

READ BINARY (EF 識別子=“12“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“92“ = 短縮 EF 識別子指定
P2	“00“ = オフセット
Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

3）EF.OD の読み出し

READ BINARY (EF 識別子=“11“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“91“ = 短縮 EF 識別子指定
P2	“00“ = オフセット
Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	1～256	データ

SW1	1	
SW2	1	

4）EF.CD 読み出し

READ BINARY (EF 識別子="15")

（1）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"95" =短縮 EF 識別子指定
P2	"00" = オフセット
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

5）証明書読み出し

READ BINARY (EF 識別子="18")

（1）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"98" = 短縮 EF 指定
P2	"00" = オフセット
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

ファイルの終わりに達していない場合には、READ BINARY のコマンドを続けて発行し、ファイルの終わりまで読み出す。

（3）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"01" = カレント EF 指定、オフセット（15 ビットの上位）
P2	"00" = オフセット（15 ビットの下位 8 ビット）
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（4）レスポンス

	長さ	意味
データ	可変	データ
SW1	1	
SW2	1	

（以降ファイルの終わりに達するまで、オフセットを変えながら READ BINARY コマンドを続け

て発行する）

READ BINARY (EF 識別子="19")

（1）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"99" = 短縮 EF 識別子指定
P2	"00" = オフセット
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	可変	データ
SW1	1	
SW2	1	

（以降ファイルの終わりに達するまで、オフセットを変えながら READ BINARY コマンドを続けて発行する）

READ BINARY (EF 識別子="1A")

（1）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"9A" =短縮 EF 識別子指定
P2	"00" = オフセット
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	可変	データ
SW1	1	
SW2	1	

（以降ファイルの終わりに達するまで、オフセットを変えながら READ BINARY コマンドを続けて発行する）

(READ BINARY (EF 識別子="1B"))

（1）コマンド

CLA	"00"
INS	"B0" = READ BINARY
P1	"9B" = 短縮 EF 識別子指定
P2	"00" = オフセット
Le	"00" = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	可変	データ
SW1	1	
SW2	1	

（以降ファイルの終わりに達するまで、オフセットを変えながら READ BINARY コマンドを続けて発行する）

A.3.3 署名計算

注：IC カード内の特定の HPKI アプリケーションの私有鍵によって署名演算する。事前に証明書が確認されて、DF が既知であることが必要。

1）アプリケーション/DF の選択

SELECT FILE

・選択する DF は事前にわかっているものとする。（証明書を確認して対象となる DF は既知である）。

（1）コマンド

CLA	“00“
INS	“A4“ = SELECT
P1	“04“ = DF 名選択
P2	“00“ = DF 選択
Lc	XX = データフィールドの長さ(最長 16 バイト)
データ	“E8 28 BD 08 0F XX XX XX XX XX XX“
Le	“00“ = FCI を受け取る

（2）レスポンス

	長さ	意味
データ	5～20	FCI(TAG=’84’)テンプレート
SW1	1	
SW2	1	

2）EF.CIAInfo の読み出し

READ BINARY(EF 識別子=“12“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“92“ = 短縮 EF 識別子指定
P2	“00“ = オフセット
Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

（2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

3）EF.OD の読み出し

READ BINARY (EF 識別子=“11“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“91“ = 短縮 EF 識別子指定
P2	“00“ = オフセット

Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

4）EF.AOD の読み出し

READ BINARY (EF 識別子=“13“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“93“ = 短縮 EF 識別子指定
P2	“00“ = オフセット
Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

2）レスポンス

	長さ	意味
データ	1～256	データ
SW1	1	
SW2	1	

5）EF.PrKD 読み出し

READ BINARY (EF 識別子=“14“)

（1）コマンド

CLA	“00“
INS	“B0“ = READ BINARY
P1	“94“ = 短縮 EF 識別子指定
P2	“00“ = オフセット
Le	“00“ = 読み出しバイト数(256 バイト以内のファイルの終りまで)

2）レスポンス

	長さ	意味
データ	1～255	データ
SW1	1	
SW2	1	

6）署名

VERIFY (EF 識別子=“16“) （毎回）

（1）コマンド

CLA	“00“
INS	“20“ = VERIFY
P1	“00“
P2	“96“ = 短縮 EF 識別子指定
Lc	データ長
	データ（PIN）

2）レスポンス

	長さ	意味
SW1	1	
SW2	1	

MANAGE SECURITY ENVIRONMENT

SET DST (P1P2="41 B6") ＋データ（鍵へのリファレンス: EF 識別子="0017"）

（1）コマンド

CLA	"00"
INS	"22" = MANAGE SECURITY ENVIRONMENT
P1	"41" = 計算
P2	"B6" = 署名生成
Lc	"04" = データフィールドの長さ
データ	"81 02 00 17" = 署名生成鍵の IEF 指定

2）レスポンス

	長さ	意味
SW1	1	
SW2	1	

PERFORM SECURITY OPERATION

P1P2="9E 9A"＋データ（パディングしたハッシュ値）

（1）コマンド

CLA	"00"
INS	"2A" = PERFORM SECURITY OPERATION
P1	"9E" = 署名計算
P2	"9A" = データフィールドのデータに署名
Lc	XX = データフィールドの長さ
データ	署名するハッシュデータ

2）レスポンス

	長さ	意味
データ	XX	署名結果
SW1	1	
SW2	1	

A.4　PKCS #11 利用のシーケンス

(A) 初期処理

ライブラリのロード	電子署名用(電子認証用)PKCS #11 ライブラリをロードする。
↓	
C_GetFunctionList	関数ポインタリストの取得 ppFunctionList : API アドレスポインタ
↓	
C_Initialize	PKCS #11 ライブラリの初期化

	pInitArgs：NULL_PTR

(B) 終了処理

C_Finalize	PKCS #11 ライブラリの終了処理 pReserved：NULL_PTR
↓	
ライブラリのアンロード	電子署名用(電子認証用)PKCS #11 ライブラリをアンロードする。

(C) 証明書取得

初期化処理	(A)初期処理参照
↓	
C_GetSlotList	スロットリスト数の取得 tokenPresent：CK_TRUE pSlotList：NULL_PTR pulCount：スロット数格納アドレス

IC カード内の HPKI アプリケーションとスロット ID を対応させる必要がある。そのためには、電子署名用(電子認証用)ID “E8 28 BD 08 0F XX XX … XX XX”を DF 名として指定して選択。

↓	
C_GetSlotList	スロットリストの取得 tokenPresent：CK_TRUE pSlotList：スロットリスト格納アドレス pulCount：スロット数格納アドレス
↓	
C_OpenSession	アプリケーションとトークン間のセッションの確立 slotID：C_GetSlotList で取得したスロットリストの 1 番目(pSlotList[0]) flags：CKF_SERIAL_SESSION pApplication：NULL_PTR Notify：NULL_PTR phSession：セッションハンドル格納アドレス

A.2.2 のシーケンスに従い、IC カード内のアプリケーションから全ての証明書を読み出す。

↓	
C_FindObjectsInit	証明書検索操作の初期設定 hSession：C_OpenSession で取得したハンドル pTemplate：以下の属性を指定 (1) type=CKA_CLASS value=CKO_CERTIFICATE (2) type=CKA_TOKEN value=CK_TRUE ulCount：設定する属性数(2)
↓	
C_FindObjects	証明書の検索 hSession：C_OpenSession で取得したハンドル phObject：オブジェクトハンドル格納アドレス

	ulMaxObjectCount：オブジェクトハンドル格納領域数(4) pulObjectCount：発見したオブジェクト数格納アドレス
	↓
ループ	① (pulObjectCount-1)までループ (カウンタをj、初期値を0とする)
	↓
C_GetAttributeValue	証明書サイズの取得 hSession：C_OpenSessionで取得したハンドル hObject：C_FindObjectsで取得したオブジェクトハンドルリストのj番目(phObject[j]) pTemplate：以下の属性を指定 (1) type=CKA_LABEL(※1) value=NULL_PTR ulValueLen=サイズ格納領域アドレス (2) type=CKA_VALUE(※2) value=NULL_PTR ulValueLen=サイズ格納領域アドレス ulCount：設定する属性数(2)
	↓
C_GetAttributeValue	証明書の取得 hSession：C_OpenSessionで取得したハンドル hObject：C_FindObjectsで取得したオブジェクトハンドルリストのj番目(phObject[j]) pTemplate：以下の属性を指定 (1) type=CKA_LABEL(※1) value=ラベル格納アドレス ulValueLen=格納領域サイズ (2) type=CKA_VALUE(※2) value=証明書格納アドレス ulValueLen=格納領域サイズ ulCount：設定する属性数(2)
	↓
証明書のチェック	ラベル(※1)または証明書(※2)を用いて、求める証明書(ex. エンドエンティティの証明書等)であるかを判定する。 求める証明書と一致すれば、次へ進む。 求める証明書と一致しなければ、カウンタjを1加算して①に戻る。
	↓
C_FindObjectsFinal	証明書検索操作の終了処理 hSession：C_OpenSessionで取得したハンドル
	↓
C_CloseSession 又は、 C_CloseAllSessions	セッションの切断 C_CloseSessionの場合 hSession：C_OpenSessionで取得したハンドル C_CloseAllSessionsの場合 slotID：C_OpenSessionに指定したスロットID

↓

終了処理	(B)終了処理参照

(D) 署名生成処理

署名に必要な私有鍵は、n(modulus)と e(publicExponent)から検索する。そのため、事前にすべての証明書を検索し、ラベルまたは証明書データによって求めている証明書か否かを判定しておく必要がある。

初期化処理	(1)初期処理参照

↓

C_GetSlotList	スロットリスト数の取得 tokenPresent：CK_TRUE pSlotList：NULL_PTR pulCount：スロット数格納アドレス

IC カード内の HPKI アプリケーションとスロット ID を対応させる必要がある。そのためには、電子署名用(電子認証用)ID “E8 28 BD 08 0F XX XX … XX XX”を DF 名として指定して選択。

↓

C_GetSlotList	スロットリストの取得 tokenPresent：CK_TRUE pSlotList：スロットリスト格納アドレス pulCount：スロット数格納アドレス

↓

C_OpenSession	アプリケーションとトークン間のセッションの確立 slotID：C_GetSlotList で取得したスロットリストの 1 番目(pSlotList[0]) flags：CKF_SERIAL_SESSION pApplication：NULL_PTR Notify：NULL_PTR phSession：セッションハンドル格納アドレス

↓

C_FindObjectsInit	証明書検索操作の初期設定 hSession：C_OpenSession で取得したハンドル pTemplate：以下の属性を指定 (1) type=CKA_CLASS value=CKO_CERTIFICATE (2) type=CKA_TOKEN value=CK_TRUE ulCount：設定する属性数(2)

A.2.2 のシーケンスに従い、IC カード内のアプリケーションから全ての証明書を読み出す。

↓

C_FindObjects	証明書の検索 hSession：C_OpenSession で取得したハンドル phObject：オブジェクトハンドル格納アドレス ulMaxObjectCount：オブジェクトハンドル格納領域数(4) pulObjectCount：発見したオブジェクト数格納アドレス

↓

ループ	① (pulObjectCount-1)までループ (カウンタをj、初期値を0とする)

↓

C_GetAttributeValue	証明書サイズの取得 hSession：C_OpenSession で取得したハンドル hObject：C_FindObjects で取得したオブジェクトハンドルリストのj番目(phObject[j]) pTemplate：以下の属性を指定 (1) type=CKA_LABEL(※1) value=NULL_PTR ulValueLen=サイズ格納領域アドレス (2) type=CKA_VALUE(※2) value=NULL_PTR ulValueLen=サイズ格納領域アドレス ulCount：設定する属性数(2)

↓

C_GetAttributeValue	証明書の取得 hSession：C_OpenSession で取得したハンドル hObject：C_FindObjects で取得したオブジェクトハンドルリストのj番目(phObject[j]) pTemplate：以下の属性を指定 (1) type=CKA_LABEL(※1) value=ラベル格納アドレス ulValueLen=格納領域サイズ (2) type=CKA_VALUE(※2) value=証明書格納アドレス ulValueLen=格納領域サイズ ulCount：設定する属性数(2)

↓

証明書のチェック	ラベル(※1)または証明書(※2)を用いて、エンドエンティティの証明書であるかを判定する。 求める証明書と一致すれば、次へ進む。 求める証明書と一致しなければ、カウンタjを1加算して①に戻る。

↓

C_FindObjectsFinal	証明書検索操作の終了処理 hSession：C_OpenSession で取得したハンドル

↓

EとNの取得	エンドエンティティの証明書から公開鍵のEとNを取得

↓

C_Login	トークンへのログイン hSession：C_OpenSession で取得したハンドル userType：CKU_USER pPin：パスワード文字列 ulPinLen：パスワード文字列長

A.3.3 の PIN 照合を行う。PIN 照合は、署名計算直前に必要となる場合がある。

↓

C_FindObjectsInit	私有鍵検索操作の初期設定 hSession：C_OpenSession で取得したハンドル pTemplate：以下の属性を指定 (1) type=CKA_CLASS value=CKO_PRIVATE_KEY (2) type=CKA_TOKEN value=CK_TRUE (3) type=CKA_MODULUS value=公開鍵の N (4) type=CKA_PUBLIC_EXPONENT value=公開鍵の E ulCount：設定する属性数(4)

↓

C_FindObjects	私有鍵の検索 hSession：C_OpenSession で取得したハンドル phObject：オブジェクトハンドル格納アドレス ulMaxObjectCount：オブジェクトハンドル格納領域数(1) pulObjectCount：発見したオブジェクト数格納アドレス

↓

C_FindObjectsFinal	私有鍵検索操作の終了処理 hSession：C_OpenSession で取得したハンドル

↓

ダイジェスト値の生成	署名対象データのダイジェスト値を生成

↓

DigestInfo の作成	ダイジェスト値を使用して DigestInfo を作成

↓

C_SignInit	署名処理の初期設定 hSession：C_OpenSession で取得したハンドル pMechanism：CKM_RSA_PKCS hKey：C_FindObjects で取得したオブジェクトハンドル

↓

C_Sign	署名値長取得 hSession：C_OpenSession で取得したハンドル pData：DigestInfo ulDataLen：DigestInfo のデータ長 pSignature：NULL_PTR pulSignatureLen：署名値長格納アドレス

↓

C_Sign	署名 hSession：C_OpenSession で取得したハンドル pData：DigestInfo ulDataLen：DigestInfo のデータ長 pSignature：署名値格納アドレス pulSignatureLen：署名値長格納アドレス

ここで、A.3.3 のハッシュの設定、署名計算の演算を実行する必要がある。PIN の照合はここで行っても良い。

↓

C_Logout	トークンからのログアウト hSession：C_OpenSession で取得したハンドル

↓

(連続して署名を行う場合)	(1 回の署名毎に C_Login～C_Logout を繰り返す。 認証を連続して行う場合は 1 回の署名毎に C_Login～C_Logout の繰り返しは必須ではない。)

↓

C_CloseSession 又は、 C_CloseAllSessions	セッションの切断 C_CloseSession の場合 hSession：C_OpenSession で取得したハンドル C_CloseAllSessions の場合 slotID：C_OpenSession に指定したスロット ID

↓

終了処理	(B)終了処理参照

A.5 CAPI 利用のシーケンス

本シーケンス例で想定している条件は以下の通り：

(1) 「電子署名用 CSP」、「電子認証用 CSP」から見た場合、それぞれの PKI カードアプリケーションに対して、IC カード内にデフォルトコンテナ（電子署名用あるいは電子認証用の RSA 鍵ペア及び利用者証明書）が１つ存在する。

(A) 利用者証明書取得処理

CryptAcquireContext	暗号プロバイダとコンテナを指定して、プロバイダハンドルを取得する。 phProv：プロバイダハンドル戻り値。 szContainer：コンテナ名 NULL を設定する（デフォルトコンテナ）。 szProvider：プロバイダ名を設定する。 電子署名用 CSP ならば"HPKI Crypto Service Provider for Non Repudiation"、電子認証用 CSP ならば"HPKI Crypto Service Provider for Authentication"を設定する。 dwProvType：プロバイダタイプ PROV_RSA_AES を設定する。 dwFlags：0 を設定する。

↓

CryptGetUserKey	コンテナの保持する RSA 鍵ペアのハンドルを取得する。 hProv：CryptAcquireContext で取得したプロバイダハンドル dwKeySpec：鍵種別として AT_SIGNATURE を指定。 phUserKey ：鍵ハンドル戻り値。

↓

CryptGetKeyParam	鍵ハンドルに付随する利用者証明書データのデータ長を取得する。 hKey：CryptGetUserKey で取得した鍵ハンドル。

	dwParam：取得するデータ種別として、KP_CERTIFICATE を指定。 pbData：取得データ戻り値バッファとして NULL を指定 pdwDataLen：データ長戻り値。 dwFlags：0 を指定。
	↓
CryptGetKeyParam	鍵ハンドルに付随する利用者証明書データのデータを取得する。 hKey：CryptGetUserKey で取得した鍵ハンドル。 dwParam：取得するデータ種別として、KP_CERTIFICATE を指定。 pbData：取得データ戻り値バッファ。前段で得たデータ長分のバッファを確保し、そのアドレスを指定。 pdwDataLen：データ長。前段で得たデータ長を指定。 dwFlags：0 を指定。 **※　証明書チェインの検証については、必要に応じて別途実施する。**
	↓
CryptDestroyKey	鍵ハンドルを破棄する。 hKey：CryptGetUserKey で得た鍵ハンドル。
	↓
CryptReleaseContext	プロバイダハンドルを解放する。 hProv：プロバイダハンドル。 dwFlags：0 を指定。

(B)　署名処理

CryptAcquireContext	暗号プロバイダとコンテナを指定して、プロバイダハンドルを取得する。 phProv：プロバイダハンドル戻り値。 szContainer：コンテナ名 NULL を設定する（デフォルトコンテナ）。 szProvider,：プロバイダ名を設定する。 電子署名用 CSP ならば"HPKI Crypto Service Provider for Non Repudiation"、電子認証用 CSP ならば"HPKI Crypto Service Provider for Authentication"を設定する。 dwProvType,：プロバイダタイプ PROV_RSA_AES を設定する。 dwFlags：0 を設定する。
	↓
CryptGetUserKey	コンテナの保持する RSA 鍵ペアのハンドルを取得する。 hProv：CryptAcquireContext で取得したプロバイダハンドル dwKeySpec：鍵種別として AT_SIGNATURE を指定。 phUserKey　：鍵ハンドル戻り値。
	↓
CryptGetKeyParam	鍵ハンドルに付随する利用者証明書データのデータ長を取得する。 hKey：CryptGetUserKey で取得した鍵ハンドル。 dwParam：取得するデータ種別として、KP_CERTIFICATE を指定。 pbData：取得データ戻り値バッファとして NULL を指定 pdwDataLen：データ長戻り値。 dwFlags：0 を指定。

↓

CryptGetKeyParam	鍵ハンドルに付随する利用者証明書データのデータを取得する。 hKey：CryptGetUserKey で取得した鍵ハンドル。 dwParam：取得するデータ種別として、KP_CERTIFICATE を指定。 pbData : 取得データ戻り値バッファ。前段で得たデータ長分のバッファを確保し、そのアドレスを指定。 pdwDataLen：データ長。前段で得たデータ長を指定。 dwFlags：0 を指定。 **※　証明書チェインの検証については、必要に応じて別途実施する。**

↓

CryptCreateHash	ハッシュオブジェクトの生成 hProv：プロバイダハンドルを指定。 Algid：ハッシュアルゴリズム CALG_SHA_256 を指定。 hKey：　NULL を指定。 dwFlags：0 を指定。 phHash：ハッシュハンドル戻り値。

↓

ハッシュ値の生成	署名対象データのハッシュ値を生成

↓

CryptSetHashParam	ハッシュオブジェクトのパラメータを設定 hHash：ハッシュハンドル。 dwParam：ハッシュ値設定のため HP_HASHVAL を指定。 pbData：設定するハッシュ値。 dwFlags：0 を指定。

↓

CryptSignHash	ハッシュ値に署名を行った結果のデータ長を取得する。 hHash：ハッシュハンドル dwKeySpec：署名に使用する鍵の種別として AT_SIGNATURE を指定。 szDescription：NULL を指定。 dwFlags：0 を指定。 pbSignature：NULL を指定。 pdwSigLen：署名長戻り値。

↓

CryptSetProvParam	署名用鍵の PIN を設定する。 hProv：プロバイダハンドル。 dwParam：署名鍵用の PIN 値として PP_SIGNATURE_PIN を指定。 pbData：ユーザ PIN 値。 dwFlags：0 を指定。 **※ここで PIN を設定しない場合、次の CryptSignHash で CSP によって PIN 入力ダイアログが表示される。**

↓

CryptSignHash	ハッシュ値に署名を行った結果を取得する。 hHash：ハッシュハンドル dwKeySpec：署名に使用する鍵の種別として AT_SIGNATURE を指定。

	szDescription：NULL を指定。 dwFlags：0 を指定。 pbSignature : 前段で得たデータ長分のバッファを確保し、そのアドレスを指定。 pdwSigLen：前段で得たデータ長を指定。

↓

CryptDestroyHash	ハッシュオブジェクト破棄 hHash：ハッシュオブジェクトのハンドル

↓

CryptDestroyKey	鍵ハンドルを破棄する。 hKey：CryptGetUserKey で得た鍵ハンドル。

↓

CryptReleaseContext	プロバイダハンドルを解放する。 hProv：プロバイダハンドル。 dwFlags：0 を指定。

A.6　CNG 利用のシーケンス

(A)　利用者証明書取得処理

NCryptOpenStorageProvider	プロバイダ名を指定して、プロバイダハンドルを取得する。 phProvider: プロバイダハンドル格納用ポインタ pszProviderName: プロバイダ名 dwFlags: 0

↓

NCryptOpenKey	鍵ハンドルを取得する。 hProvider: NCryptOpenStorageProvider で取得したハンドル phKey: 鍵ハンドル格納用ポインタ pszKeyName: “Private key of HPKI” dwLegacyKeySpec: 0 dwFlags: 0

↓

NCryptGetProperty	証明書サイズを取得する。 hObject: NCryptOpenKey で取得したハンドル pszProperty: NCRYPT_CERTIFICATE_PROPERTY pbOutput: NULL cbOutput: 0 pcbResult: サイズ格納用ポインタ dwFlags: 0

↓

NCryptGetProperty	証明書を取得する。

	hObject: NCryptOpenKey で取得したハンドル pszProperty: NCRYPT_CERTIFICATE_PROPERTY pbOutput: 証明書格納用ポインタ cbOutput: 格納領域サイズ pcbResult: サイズ格納用ポインタ dwFlags: 0

↓

NCryptFreeObject	鍵ハンドルを解放する。 hObject: NCryptOpenKey で取得したハンドル

↓

NCryptFreeObject	プロバイダハンドルを解放する。 hObject: NCryptOpenStorageProvider で取得したハンドル

(B) 署名処理

NCryptOpenStorageProvider	プロバイダ名を指定して、プロバイダハンドルを取得する。 phProvider: プロバイダハンドル格納用ポインタ pszProviderName: プロバイダ名 dwFlags: 0

↓

NCryptOpenKey	鍵ハンドルを取得する。 hProvider: NCryptOpenStorageProvider で取得したハンドル phKey: 鍵ハンドル格納用ポインタ pszKeyName: “Private key of HPKI” dwLegacyKeySpec: 0 dwFlags: 0

↓

NCryptSetProperty	ダイアログ表示用の Window ハンドルを設定する。 hObject: NCryptOpenKey で取得したハンドル pszProperty: NCRYPT_WINDOW_HANDLE_PROPERTY pbInput: Window ハンドル cbInput: Window ハンドルのサイズ (バイト) dwFlags: 0

↓

ハッシュ値の生成	署名対象データのハッシュ値を生成する。(SHA-256)

↓

NCryptSignHash	署名値のサイズを取得する。 hKey: NCryptOpenKey で取得したハンドル pPaddingInfo: 以下の値が設定された

	BCRYPT_PKCS1_PADDING_INFO 構造体を指定 　pszAlgId=BCRYPT_SHA256_ALGORITHM pbHashValue: ハッシュ値 cbHashValue: ハッシュ値のサイズ pbSignature: NULL cbSignature: 0 pcbResult: サイズ格納用ポインタ dwFlags: BCRYPT_PAD_PKCS1

↓

NCryptSignHash	署名を実施する。 hKey: NCryptOpenKey で取得したハンドル pPaddingInfo: 以下の値が設定された BCRYPT_PKCS1_PADDING_INFO 構造体を指定 　pszAlgId=BCRYPT_SHA256_ALGORITHM pbHashValue: ハッシュ値 cbHashValue: ハッシュ値のサイズ pbSignature: 署名値格納用ポインタ cbSignature: 署名値のサイズ pcbResult: サイズ格納用ポインタ dwFlags: BCRYPT_PAD_PKCS1

↓

NCryptFreeObject	鍵ハンドルを解放する。 hObject: NCryptOpenKey で取得したハンドル

↓

NCryptFreeObject	プロバイダハンドルを解放する。 hObject: NCryptOpenStorageProvider で取得したハンドル

附属書 B（参考）PKI アプリケーションの構造例

B.1　概要

複数の事業者が HPKI 認証局の運用を行うことを前提に、カードに搭載される PKI カードアプリケーションに関して、以下の条件の下で、相互運用性確保の仕様を検討する。

・各 HPKI 認証局が独自のカードを発行する。
・証明書は、医療従事者等のエンドエンティティの証明書と認証局の証明書となる。
・HPKI で署名用証明書を利用した署名演算には、署名を行うたびに PIN の照合が必要とする。
・ファイル構造、コマンド、シーケンスは、標準に従う

B.2 以降で示すのは一例であって、実際の事業者の発行するカード及びカードに搭載されたアプリケーションの構造とは異なる。実際のカード及びカードに搭載されたアプリケーションの利用に当たっては、各事業者のカード及びアプリケーション仕様、提供されるソフトウェアモジュールの仕様に従って動作を確認する必要がある。

B.2　ファイル構造

DF/アプリケーションの構造例を図 B.1 に示す。

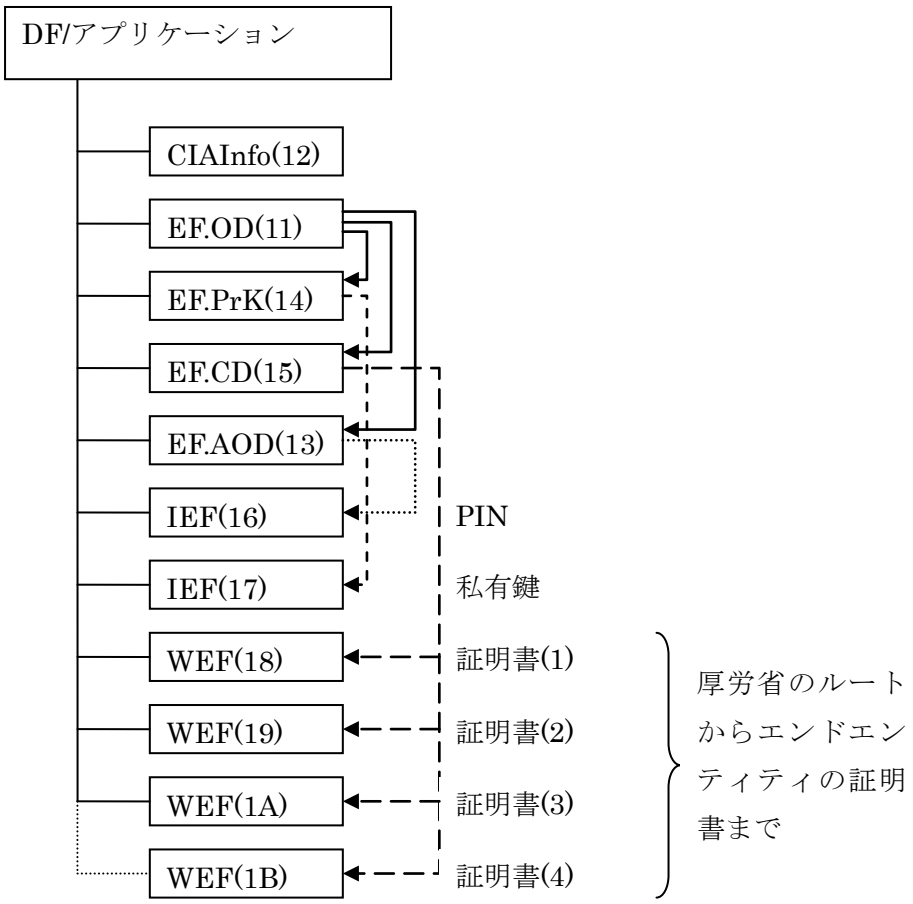


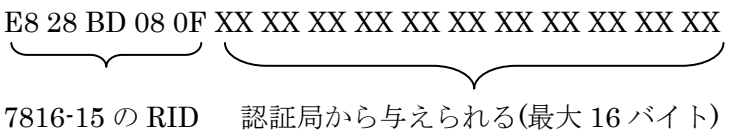
図 B.1　ISO/IEC 7816-15 に準拠したファイル構造例

注 1）追加の信頼点の証明書に関しては、考慮していない。

注 2）カッコ内の数字(XX)は、短縮 EF 識別子を示す。
注 3）カード内に複数のアプリケーションが存在する場合でも、EF.DIR は存在しないものとする

B.3　PKI アプリケーションの識別（AID）

ISO/IEC 7816-15 に従う。RID は、ISO/IEC 7816-15 で決められるので、個別アプリケーション識別子(PIX)部分を各 HPKI 認証局が定める ID とする。



注）XX の値は、厚生労働省が指定する。全体で最大 16 バイトとなる。

B.4　各 EF の内容

B.4.1　概要

ISO/IEC 7816-15 に従って規定する。すべて、短縮 EF 識別子を用いる。表 B.1 に一例を示す。暗号情報の一例を B.4.2、B.4.3、B.4.4、B.4.5 及び B.4.6 に示す。

表 B.1　必要となるファイル一覧

ファイル	短縮 EF 識別子	EF 識別子	ファイルタイプ	内容
CIAInfo	'12'(ISO/IEC 7816-15)で指定)	'5032'　ISO/IEC 7816-15 指定)	透過	B.4.2：　カード情報
OD	'11'(ISO/IEC 7816-15 で指定)	'5031'（ISO/IEC 7816-15 で指定)	透過	B.4.3：オブジェクト情報
AOD	'13'　(例)	事業者が定める	透過	B.4.4：認証オブジェクト情報
PrKD	'14'　(例)	事業者が定める	透過	B.4.5：私有鍵情報
CD	'15'　(例)	事業者が定める	透過	B.4.6：証明書情報
IEF	'16'　(例)	事業者が定める		PIN 格納 (VERIFY 用)
IEF	'17'　(例)	事業者が定める		私有鍵格納 (署名用)
WEF	'18'　(例)	事業者が定める	透過	証明書格納(エンティティ)
WEF	'19'　(例)	事業者が定める	透過	信頼点の証明書格納(HPKI ROOT 認証局の自己署名証明書)
WEF	'1A'　(例)	事業者が定める	透過	下位認証局への証明書格納
WEF	'1B'　(例)	事業者が定める	透過	下位認証局への証明書：追加

B.4.2　EF.CIAInfo

```
CIAInfo ::= {
        version v2,
        label  "HPKI Application" , -- オプション
        cardflags { authRequired, prnGeneration } -- 暗号演算に利用者認証必要、乱数生成あり
}
```

B.4.3　EF.OD

```
authObjects : -- EF.AOD へのパス
        path : {
                efidOrPath '98'H -- EF.AOD の短縮 EF 識別子 ’13’
        },
privateKeys : -- EF.PrKD へのパス
    path : {
                efidOrPath 'A0'H -- EF.PrKD の短縮 EF 識別子 ’14’
    },
certificates : -- EF.CD へのパス
        path : {
                efidOrPath 'A8'H -- EF.CD の短縮 EF 識別子 ’15’
        }
```

B.4.4　EF. AOD

```
pwd : {   -- パスワード
        commonObjectAttributes {
                label "PIN ", --  オプション
                flags { modifiable }, --    オプション 変更可能
        },
        classAttributes {
                authId '16'H – 識別するための ID
        },
        typeAttributes {
                pwdFlags {
                        case-sensitive, --
                         local, --  このアプリケーション/DF 内でのみ有効
                         initialized --  初期化されている
                },
                pwdType utf8, -- UTF8   文字セットは CryptAPI の条件により、ASCII となる
                minLength 4, --  最小の長さ
                storedLength 16, --  格納の長さ
                maxLength 16, --    最大の長さ
                pwdReference ‘96’H-- Verify  コマンドの P2 パラメータ(アプリ固有の PIN)
                                        IEF の短縮 EF 識別子  '16'
        }
}
```

B.4.5　EF.PrKD

```
privateRSAKey : {
        commonObjectAttributes {
                label "Private key of HPKI",
                flags { private },    -- 利用に認証が必要
                authId '16'H – 必要となる AOD エントリ
                userConsent 1 – 私有鍵によって署名を付加する際に、毎回認証を行う（電子署名 PKI カード
```

アプリケーションの場合は必要。電子認証用の場合には必要ない）

```
            accessControlRules {
                {
                    accessMode { execute }, -- 実行
                    securityCondition: {
                        authId : '16'H, -- pointer to the AOD-entry
                    }
                }
            }
        }
        classAttributes {
            iD '17'H, -- 識別のための ID（EF 識別子を利用し、証明書と共通にする）
            usage { nonRepudiation },   （電子認証用の場合には sign）
            },
        typeAttributes {
            value {
                efidOrPath 'B8'H -- IEF の短縮 EF 識別子 '17'
            },
            modulusLength 2048
        }
}
```

B.4.6 EF.CD

```
x509Certificate : {
    commonObjectAttributes {
        label "HPKI END ENTITY CERTIFICATE",
    }
    classAttributes {
        iD '17'H --識別のための ID（私有鍵と共通にする）
    },
    typeAttributes {
        value indirect :
        path : {
            efidOrPath 'C0'H -- 格納された WEF の短縮 EF 識別子 '18'
        }
    }
},
x509Certificate : {
    commonObjectAttributes {
        label " MHLW CA CERTIFICATE ",  --厚労省の CA 証明書
    }
    classAttributes {
        iD '19'H --識別のための ID（EF 識別子と同じとする）
        authority TRUE -- CA の証明書
    },
    typeAttributes {
```

```
            value indirect :
            path : {
                efidOrPath 'C8'H --  格納された WEF の短縮 EF 識別子  '19'
            }
        }
}
x509Certificate : {
        commonObjectAttributes {
            label   " HPKI ROOT CA CERTIFICATE",   --各事業者の CA

        }
        classAttributes {
            iD '1A'H --識別のための ID（EF 識別子と同じとする）
            authority TRUE – 厚労省の証明書
        },
        typeAttributes {
            value indirect :
            path : {
                efidOrPath 'D0'H – 格納された WEF の短縮 EF 識別子  '1A'
            }
        }
}
```

中間 CA が存在する場合にはもう 1 つ X509 オブジェクトを追加する

```
x509Certificate : {
        commonObjectAttributes {
            label " HPKI CA CERTIFICATE ",   --各事業者の中間 CA
        }
        classAttributes {
            iD '1B'H --識別のための ID（EF 識別子と同じとする）
            authority TRUE – 中間 CA の証明書
        },
        typeAttributes {
            value indirect :
            path : {
                efidOrPath 'D8'H – 格納された WEF の短縮 EF 識別子  '1B'
            }
        }
}
```

附属書 C（参考）PKI アプリケーション利用のコマンド

C.1　コマンド一覧

本ガイドラインでは、以下のコマンドについて満たすべき最低限の仕様を提示する。

SELECT FILE (ISO/IEC 7816-4)
VERIFY (ISO/IEC 7816-4)
READ BINARY (ISO/IEC 7816 -4)
MANAGE SECURITY ENVIRONMENT (ISO/IEC 7816-4)
PERFOM SECURITY OPERATION (ISO/IEC 7816-8)

C.2 で各コマンドの最低限満たすべき仕様を示す。ステータスワードは、代表的な例であって、カードの実装に応じて ISO/IEC 7816-4 で規定される他のステータスワードを適切に出力してもよい。

C.2 SELECT

C.2.1 条件

- アプリケーション/DF のパーシャルセレクトが可能
- FCI にて選択されたアプリケーション/DF の AID が取得可能

C.2.2 コマンドメッセージ

DF 選択

CLA	INS	P1	P2	Lc	データ	Le
“00”	“A4”	“XX”	“XX”			
(1)	(1)	(1)	(1)	(1)	(1～16)	(0 or 1)

EF 選択

CLA	INS	P1	P2	Lc	データ
“00”	“A4”	“XX”	“XX”		(EF 識別子)
(1)	(1)	(1)	(1)	(1)	(2)

パラメータ	長さ	意味	備考
P1	1	選択制御子	
P2	1	選択オプション	
Lc	1	ファイル識別子 or ファイル名の長さ	
データ	1～16	ファイル識別子 or ファイル名	
Le	1	FCI データのバイト数	

P1 コーディング

b8	b7	b6	b5	b4	b3	b2	b1	意味
0	0	0	0	0	0	x	x	ファイル ID による選択
-	-	-	-	-	-	1	0	カレント DF 直下の EF
0	0	0	0	0	1	0	0	DF 名選択

P2 コーディング

b8	b7	b6	b5	b4	b3	b2	b1	意味
0	0	0	0	0	0	-	-	FCI オプションテンプレート応答
0	0	0	0	1	1-	-	-	FCI 応答なし(EF 識別子による選択時)
0	0	0	0	-	-	0	0	最初または唯一のファイル
0	0	0	0	-	-	1	0	次ファイル（パーシャル DF 名指定可）

C.2.3　レスポンスメッセージ

レスポンス APDU

データ	SW1	SW2
(1)	(1)	(1)

パラメータ	長さ	意味	備考
データ	5～30	FCI(TAG=”84”)テンプレート	TAG=”84” :DF 名
SW1	1		
SW2	1		

注）レスポンスの SW1 及び SW2 が”90 00”以外の場合のデータ長は、0 となる

データ部

“6F”	長さ	“84”	長さ	DF 名
(1)	(1)	(1)	(1)	(1～16)

C.2.4　ステータスワード

SW1	SW2	ステータスワードの意味
“90”	“00”	正常終了
“62”	“83”	選択された DF が閉そくしている 選択された EF の親 DF が閉そくしている
“67”	“00”	Lc および Le フィールドが間違っている。 APDU の長さが間違っている。
“68”	“81”	指定された論理チャネル番号によるアクセス機能を提供しない
	“82”	CLA バイトで指定されたセキュアメッセージング機能を提供しない
“6A”	“81”	機能が提供されない
	“82”	アクセス対象意ファイルがない
	“86”	P1 P2 の値が正しくない
	“87”	Lc の値が P1 P2 と矛盾している

C.3　VERIFY

C.3.1　条件

・　短縮 EF 識別子での実行

C.3.2　コマンドメッセージ

CLA	INS	P1	P2	Lc	データ
“00”	“20”	“XX”	“XX”		
(1)	(1)	(1)	(1)	(1 or なし)	(可変 or なし)

パラメータ	長さ	意味	備考
P1	1	なし（”00”固定）	
P2	1	参照データの限定	

Lc	1 or なし		
データ	可変 or なし	照合データ	

P2 コーディング

B8	b7	b6	b5	b4	b3	b2	b1	意味
1	-	-	-	-	-	-	-	特定の参照データ
-	0	0	0	0	0	0	0	カレント EF
1	0	0	x	x	x	x	x	短縮 EF 識別子（"11111"以外）

C.3.3 レスポンスメッセージ

レスポンス APDU

SW1	SW2
(1)	(1)

パラメータ	長さ	意味	備考
SW1	1		
SW2	1		

C.3.4 ステータスワード

SW1	SW2	ステータスワードの意味
“90”	“00”	正常終了
“63”	“00” “CX”	照合不一致とする 照合不一致[X によって、残りの再試行可能回数(0-15)を示す]
“67”	“00”	Lc および Le フィールドが間違っている。 APDU の長さが間違っている。
“68”	“81”	指定された論理チャネル番号によるアクセス機能を提供しない
	“82”	CLA バイトで指定されたセキュアメッセージング機能を提供しない
“69”	“81”	ファイル構造と矛盾したコマンド
	“83”	認証方法を受け付けない
	“84”	参照された IEF が閉そくしている
“6A”	“81”	機能が提供されない
	“82”	短縮 EF 識別子で指定した IEF がない
	“86”	P1 P2 の値が正しくない
	“87”	Lc の値が P1 P2 と矛盾している
	“88”	参照された鍵が正しく設定されていない

C.4 READ BINARY

C.4.1 条件

・ 短縮 EF 識別子での実行

C.4.2 コマンドメッセージ

CLA	INS	P1	P2	Lc

“00”	“B0”	“XX”	“XX”	
(1)	(1)	(1)	(1)	(1)

パラメータ	長さ	意味	備考
P1-P2	各 1	読み出し対象短縮 EF 識別子及び先頭のバイナリデータのオフセット	
Le	1		

P1-P2 コーディング

P1								P2	意味
B8	b7	b6	b5	b4	b3	b2	b1		
0	-	-	-	-	-	-	-	--	カレント EF 指定
0	x	x	x	x	x	x	x	xx	オフセット(15 ビット)
-	0	0	0	0	0	0	0	--	カレント EF 指定
1	0	0	x	x	x	x	x	--	短縮 EF 識別子（”11111”以外）
1	0	0						xx	オフセット(8 ビット)

C.4.3　レスポンスメッセージ

レスポンス APDU

データ	SW1	SW2
(可変)	(1)	(1)

パラメータ	長さ	意味	備考
データ	可変	読み出されたデータ	
SW1	1		
SW2	1		

C.4.4　ステータスワード

SW1	SW2	ステータスワードの意味
“90”	“00”	正常終了
“62”	“83”	DF が閉そくしている
“67”	“00”	Lc および Le フィールドが間違っている。 APDU の長さが間違っている。
“68”	“81”	指定された論理チャネル番号によるアクセス機能を提供しない
	“82”	CLA バイトで指定されたセキュアメッセージング機能を提供しない
“69”	“81”	ファイル構造と矛盾したコマンド
	“83”	認証方法を受け付けない
“6A”	“81”	機能が提供されない
	“82”	短縮 EF 識別子で指定したファイルがない
	“86”	P1 P2 の値が正しくない
	“87”	Lc の値が P1 P2 と矛盾している

C.5 MANAGE SECURITY ENVIRONMENT

C.5.1 条件

・ 署名鍵の指定

C.5.2 コマンドメッセージ

CLA	INS	P1	P2	Lc	データ
“00”	“22”	“XX”	“XX”		
(1)	(1)	(1)	(1)	(1)	(可変)

パラメータ	長さ	意味	備考
P1	1	“41”: SET	
P2	1	P1 が SET あるいは GET CRT の場合は、テンプレートのタグ	“B6” （電子署名用 CRT）
Lc	1	データ長	
データ	4	Tag= ”81”, Len= ”02” , EF 識別子	

C.5.3 レスポンスメッセージ

レスポンス APDU

SW1	SW2
(1)	(1)

パラメータ	長さ	意味	備考
SW1	1		
SW2	1		

C.5.4 ステータスワード

SW1	SW2	ステータスワードの意味
“90”	“00”	正常終了
“62”	“83”	DF が閉そくしている（カレント SE 利用できない）
“67”	“00”	Lc および Le フィールドが間違っている。 APDU の長さが間違っている。
“68”	“81”	指定された論理チャネル番号によるアクセス機能を提供しない
	“82”	CLA バイトで指定されたセキュアメッセージング機能を提供しない
“69”	“82”	セキュリティステータスが満足されない
	“85”	コマンドの使用条件が満足されない
“6A”	“80”	データフィールドのタグが正しくない
	“85”	Lc の値が、TLV 構造に矛盾している
	“86”	P1 P2 の値が正しくない

C.6 PERFORM SECURITY OPERATION

C.6.1 条件

・ 私有鍵での電子署名暗号演算の実行
・ パディングはカード外で行う

C.6.2 コマンドメッセージ

CLA	INS	P1	P2	Lc	データ	Le
“00”	“2A”	“XX”	“XX”			
(1)	(1)	(1)	(1)	(1)	(可変)	(1 or 3)

パラメータ	長さ	意味	備考
P1	1	“9E”（デジタル署名）	Encipher 処理
P2	1	“9A” (データフィールドが署名)	
Lc	1	データ長	
データ	可変	署名されるデータ	
Le	1		

C.6.3 レスポンスメッセージ

レスポンス APDU

データ	SW1	SW2
(可変)	(1)	(1)

パラメータ	長さ	意味	備考
データ	可変	電子署名データ	
SW1	1		
SW2	1		

C.6.4 ステータスワード

SW1	SW2	ステータスワードの意味
“90”	“00”	正常終了
“62”	“83”	DF が閉そくしている（カレント SE 利用できない）
“67”	“00”	Lc および Le フィールドが間違っている。 APDU の長さが間違っている。
“68”	“81”	指定された論理チャネル番号によるアクセス機能を提供しない
	“82”	CLA バイトで指定されたセキュアメッセージング機能を提供しない
“69”	“82”	セキュリティステータスが満足されない
	“85”	コマンドの使用条件が満足されない
“6A”	“80”	データフィールドのタグが正しくない
	“85”	Lc の値が、TLV 構造に矛盾している
	“86”	P1 P2 の値が正しくない

附属書 D（参考）IC カードリーダライタとのインタフェース

IC カードリーダライタとのインタフェースとしてプラットフォームにより以下の二つが存在する。

1）PC/SC （MS-Windows におけるインタフェース）
2）PCSC-Lite(LINUX や Mac OS など UNIX 系の OS におけるインタフェース)

IC カードリーダライタが、PC などに接続される物理的インタフェースには、RS-232C、PCMCIA、USB 等がある。PC/SC や PCSC-Lite は、こうした物理的インタフェースの違いや、異なるベンダーの IC カードリーダライタの論理的なインタフェースの仕様の違いも吸収し、IC カードリーダライタのアプリケーションに対して IC カードリーダライタへの汎用的な API を提供する。HPKI 用 IC カードガイドラインにおいても、特定の IC カードリーダライタへの依存性を最小限にするためにするため、PC/SC や PCSC-Lite といったフレームワーク上で動作することを推奨している。ここでは、PC/SC、PCSC-Lite、更に、USB インタフェースを持つ IC カードリーダライタにおけるドライバの標準である USB CCID(Chip/Smart Card Interface Devices)について概説する。

◇**PC/SC** (Personal Computer/Smart Card)

PC/SC は、ベンダー各社が製造する IC カード、IC カードリーダライタを、Windows 環境上で相互利用できるようにするためのインタフェース仕様として、米マイクロソフト社を中心とした複数のベンダーからなる PC/SC ワークグループによって定められ、1997 年 12 月にバージョン 1.0 がリリースされ、 2005 年 6 月には、2017 年現在の最新版であるバージョン 2.01 がリリースされている。
PC/SC においては、IC カードリーダライタのベンダーが、PC/SC に準拠した、個々の IC カードリーダライタのドライバを提供する必要がある。
IC カードリーダライタのアプリケーション(HPKI 用 IC カードガイドラインにおいては、PKCS#11 などの PKI 機能を提供する汎用ミドルウェア）は、PC/SC の API を利用することにより、個別の IC カードリーダライタのドライバを直接ハンドリングせず、ドライバから独立した実装が可能になっている。
Windows 環境では、マイクロソフト社の WHQL(Windows Hardware Quality Labs)が、IC カードリーダライタのようなハードウェアとドライバの認定を行っている。IC カードリーダライタと、対応するドライバに関して PC/SC に準拠した形での WHQL の認定が行われており、比較的安定した動作を行う製品が数多く販売されている。

◇PCSC-Lite

PCSC-Lite は、LINUX や Mac OS など UNIX 系の OS において、PC/SC と同様の目的のために仕様が作成され、オープンソースソフトウェアとして提供されている。PCSC-Lite は、仕様的には、PC/SC のサブセットであるが、LINUX など UNIX 系の OS での実装可能な仕様となっている。PC/SC 同様、あるベンダーIC カードリーダライタを利用するためには、その IC カードリーダライタのためのドライバを PCSC-Lite に組み入れることになる。ただし、PCSC-Lite においては、マイクロソフト社のような認定制度(WHQL による認定)は存在せず、また、多くの場合、IC カードリーダ・ライタベンダーは、ドライバを提供していない。代わって、オープンソースコミュニティが、PCSC-Lite 準拠した、IC カードリーダライタのドライバを提供している。そのため、現時点では、PCSC-Lite で動作させることのできる IC カードリーダライタは、PC/SC ほど多くはない。利用に当たっては、動作の範囲に注意する必要がある。

◇USB CCID　(Chip/Smart Card Interface Devices)

IC カードリーダライタの物理的インタフェースには、USB が非常に多く利用されつつある。USB の物理的インタフェースの IC カードリーダライタのインタフェースの仕様として、USB CCID (Chip/Smart Card Interface Devices)がある。USB CCID の仕様は、USB working group により 2001 年 3 月に Rev1.0、2005 年 4 月に Rev1.1 がリリースされている。

PC/SC に準拠した Windows プラットフォームの USB CCID のドライバは、マイクロソフトにより提供されている。また、PCSC-Lite 準拠した USB CCID のドライバは、オープンソースコミュニティにより提供されている。そのため、USB CCID に準拠した IC カードリーダライタの場合、IC カードリーダライタ固有のドライバの提供を行う必要がない。こうしたことから、新しい USB の IC カードリーダライタは、USB CCID に準拠した製品が多い。

付録―１．参考文献

ISO 17090-3 Health informatics – Public key infrastructure – Part 3: Policy management of certification authority

PC/SC Specifications https://www.pcscworkgroup.com/specifications/download/

電子政府における調達のために参照すべき暗号のリスト（CRYPTREC 暗号リスト）
https://www.cryptrec.go.jp/list.html

付録―２．作成者名簿

作成者（社名五十音順）

有馬　一閣	㈱NTT データ
沼田　祐太朗	共同印刷㈱
下野　兼揮	㈱グッドマン
平田　泰三	JAHIS 特別委員
松本　泰	セコム㈱
半田　富己男	大日本印刷㈱
浅野　之治	凸版印刷㈱
遠藤　方洋	凸版印刷㈱
山岡　弘明	富士通Ｊａｐａｎ㈱
梶山　孝治	富士フイルムヘルスケア㈱
茗原　秀幸	三菱電機㈱
太田　英憲	三菱電機インフォメーションシステムズ㈱
酒巻　一紀	三菱電機インフォメーションシステムズ㈱
清水　可奈子	三菱電機インフォメーションシステムズ㈱
谷内田　益義	㈱リコー
喜多　紘一	(一社) 保健医療福祉情報安全管理適合性評価協会

改定履歴		
日付	バージョン	内容
2018/05/08	Ver. 3.0	・HPKI 対応 IC カードガイドライン　第 1 版及び第 2 版の統合 ・参照規格等の最新情報への更新 ・Windows の新しいインタフェースの仕様追加 ・技術の進展に合わせた例の修正
2023/01/19	Ver 3.0a	・JIS X 6320 シリーズの廃止に伴う修正 ・相互運用性に関する記述の追加 ・引用規格・引用文献および参考文献の最新情報への更新 ・暗号の危殆化に関する記載の追加

（ＪＡＨＩＳ標準　18－001）
２０２３年２月発行

ＪＡＨＩＳ　HPKI 対応 IC カードガイドライン Ver .3.0a

発行元　一般社団法人　保健医療福祉情報システム工業会
〒１０５－０００４　東京都港区新橋２丁目５番５号
（新橋２丁目ＭＴビル５階）

電話　03-3506-8010　　FAX　03-3506-8070