

**FEITIAN**

# ePass2003 開発者ガイド



V1.3

Feitian Technologies Co., Ltd.

Website: [www.FTsafe.com](http://www.FTsafe.com)

## Revision History:

Date	Revision	Description
May. 2011	V1.0	Release of the first version
Nov. 2015	V1.1	Release of the second edition
Aug.2021	V1.2	対応 OS を追加
Nov.2025	V1.3	対応 OS を追加、サポートアルゴリズムを更新

## Software Developer's Agreement

All Products of Feitian Technologies Co., Ltd. (Feitian) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.
2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Feitian provided enhancement or upgrade to the Product.
3. Warranty – Feitian warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.
4. Breach of Warranty – In the event of breach of this warranty, Feitian's sole obligation is to replace or repair, at the discretion of Feitian, any Product free of charge. Any replaced Product becomes the property of Feitian.

Warranty claims must be made in writing to Feitian during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Feitian. Any Products that you return to Feitian, or a Feitian authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Feitian's Liability – Feitian's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Feitian be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Feitian has been advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

## - 目次 -

<b>ePass2003 .....</b>	<b>1</b>
<b>開発者ガイド .....</b>	<b>1</b>
<b>第 1 章 ePass2003 概要 .....</b>	<b>1</b>
1.1 ePass2003 アプリケーション・プログラミング・インターフェース .....	1
1.2 PC/SC インターフェースを利用した ePass2003 アプリケーション開発 .....	1
1.2.1 スマートカード・データベース・クエリー関数 .....	3
1.2.2 スマートカード・データベース管理関数 .....	3
1.2.3 リソースマネージャ・ハンドル関数 .....	4
1.2.4 リソースマネージャ・ツール関数 .....	4
1.2.5 スマートカード監視関数 .....	4
1.2.6 スマートカード及びリーダーへのアクセス関数 .....	4
1.2.7 スマートカードへのダイレクトアクセス関数 .....	5
1.3 CryptoAPI を利用した ePass2003 アプリケーションの開発 .....	5
1.3.1 情報の秘匿性 .....	6
1.3.2 身分の証明 .....	6
1.3.3 完全性の確認 .....	7
1.3.4 CSP と暗号化プロセス .....	7
1.3.5 CSP コンテキスト .....	8
1.3.6 CryptoAPI アーキテクチャ .....	9
1.4 PKCS#11 を利用した ePass2003 アプリケーションの開発 .....	11
<b>第 2 章 CSP モジュール .....</b>	<b>12</b>
2.1 概要 .....	12
2.1.1 プロファイル .....	12
2.1.2 機能 .....	12
2.2 サポートされているアルゴリズム .....	13
2.3 関数のサポート状況 .....	13
2.4 関数のパラメータ .....	15
2.4.1 CPAcquireContext .....	15
2.4.2 CPGetProvParam .....	15
2.4.3 CPRReleaseContext .....	16
2.4.4 CPSetProvParam .....	16
2.4.5 CPDeriveKey .....	16
2.4.6 CPDestroyKey .....	16

2.4.7	CPDuplicateKey .....	16
2.4.8	CPEExportKey .....	17
2.4.9	CPGenKey .....	17
2.4.10	CPGenRandom .....	17
2.4.11	CPGetKeyParam .....	17
2.4.12	CPGetUserKey .....	18
2.4.13	CPImportKey .....	18
2.4.14	CPSetKeyParam .....	18
2.4.15	CPDecrypt .....	19
2.4.16	CPEncrypt .....	19
2.4.17	CPCreateHash .....	19
2.4.18	CPDestroyHash .....	19
2.4.19	CPDuplicateHash .....	20
2.4.20	CPGetHashParam .....	20
2.4.21	CPHashData .....	20
2.4.22	CPHashSessionKey .....	20
2.4.23	CPSetHashParam .....	20
2.4.24	CPSignHash .....	20
2.4.25	CPVerifySignature .....	21
2.5	関数呼び出し時の留意事項 .....	21
2.5.1	概要 .....	21
2.5.2	開発プログラムサンプル .....	21
<b>第 3 章</b>	<b>PKCS#11 モジュール .....</b>	<b>22</b>
3.1	概要 .....	22
3.2	サポートしている PKCS#11 オブジェクト .....	23
3.3	サポートされているアルゴリズム .....	24
3.4	サポートされている PKCS#11 インターフェース関数 .....	25
<b>第 4 章</b>	<b>スマートカード・ミニドライバ モジュール .....</b>	<b>30</b>
4.1	スマートカード・ミニドライバの概要 .....	30
4.1.1	プロファイル .....	31
4.1.2	概要 .....	31
4.2	サポートされているアルゴリズム .....	32
4.3	関数のサポート状況 .....	32
4.4	関数のパラメータ .....	36
4.5	関数呼び出し時の留意事項 .....	37
4.5.1	概要 .....	37

付録・用語と略称 .....	38
----------------	----

# 第1章 ePass2003 概要

この章では ePass2003 アプリケーションの開発、ePass2003 でサポートしている API および API を利用した開発方法について説明します。

この章では以下のトピックについて説明します。

- ePass2003 アプリケーション・プログラミング・インターフェース
- PC/SC インターフェースを利用した ePass2003 アプリケーションの開発
- MS CryptoAPI インターフェースを利用した ePass2003 アプリケーションの開発
- PKCS#11 インターフェースを利用した ePass2003 アプリケーションの開発

## 1.1 ePass2003 アプリケーション・プログラミング・インターフェース

ePass2003 対応のアプリケーション開発は、PKI アプリケーションの開発とスマートカード・アプリケーションの開発の 2 つに大きく分類されます。

まず ePass2003 の PKI アプリケーションを開発するには、RSA PKCS#11 と互換性のある PKCS#11、または Microsoft CryptoAPI と互換性のある CSP for Microsoft CryptoAPI 2.0 のどちらかを利用する事ができます。これら 2 つの標準規格はほぼ全てのソフトウェア及びハードウェアメーカーで広く利用されているので、ePass2003 ではこれらの規格の API をベースとして開発されたアプリケーションであれば特別な追加開発をしなくても、そのまま組み込むことが出来ます。

もう一つは、スマートカード・アプリケーションの PC/SC インターフェースを利用した開発です。

ePass2003 の PKI アプリケーションインターフェースは PC/SC インターフェースが基本となっており、開発者はいくつかの PC/SC インターフェースを利用して利用者の要求仕様に沿ったカスタマイズを行うことができます。

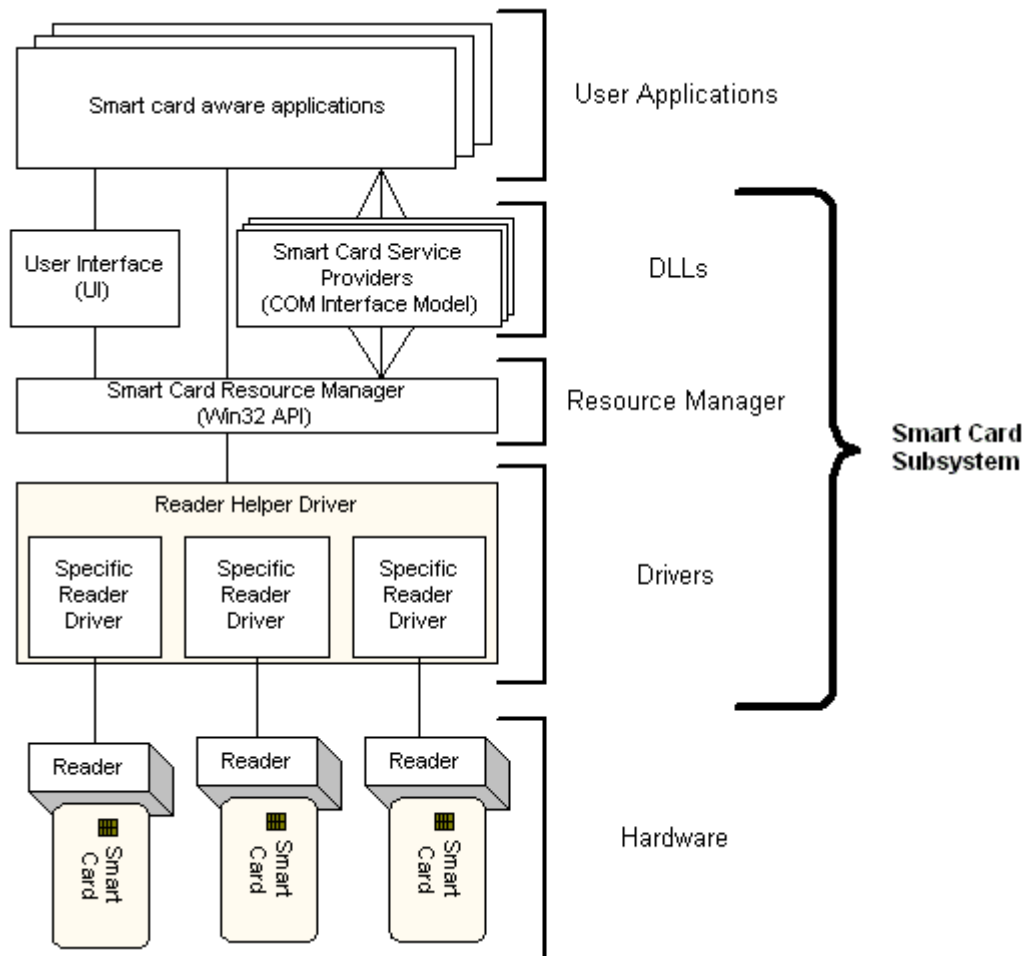
## 1.2 PC/SC インターフェースを利用した ePass2003 アプリケーション開発

Win32 プラットフォーム上のスマートカード・サブシステムは PC/SC 仕様に従って設計されています(仕様の詳細情報は、<http://www.pcscworkgroup.com> を参照して下さい)。サブシステムには以下が含まれています。

- Win32 システム・インターフェースを利用したスマートカード・リソースマネージャ
- スマートカード・リソースマネージャとのユーザ・インターフェース
- スマートカード・サービスを提供するための COM コンポーネントのセット



次の図は Win32 プラットフォーム下のスマートカード・サブシステムの構成図です。



実際上は、スマートカード・メーカーによって提供される API はスマートカード・アプリケーションによって使用されるインターフェースと切り離されているように見えるかもしれませんが、言い換えれば、スマートカード・アプリケーションは、スマートカードへアクセスするためにのみ準備されている標準の Win32 関数のサブセットを使用しているということです。スマートカード・メーカーに対しては、プログラミングインターフェースは統一されています。よって、スマートカード・メーカーによって提供されるインターフェースの変更や更新があっても、上位レベルのスマートカード・アプリケーションには影響を与えないように考慮されています。

スマートカード・リソースマネージャは中間層で動作します。スマートカード・リソースマネージャの関数セットには、スマートカード・データベース・クエリー関数、スマートカード・データベース管理関数、リソースマネージャ・ハンドル関数、リソースマネージャ・ツール関数、スマートカード監視関数、スマートカードとカードリーダーにアクセスするための関数およびスマートカードにダイレクトにアクセスするための関数などが含まれています。

## 1.2.1 スマートカード・データベース・クエリー関数

これらの関数を使って、スマートカード・タイプの特定のシステム・ユーザのリスト、特定のスマートカードのアプリケーションサービス・インターフェース、スマートカード・リーダのグループリスト、およびあるグループに属する全スマートカード・リーダのリストを検索することができます。

これらの関数を使用する場合、検索範囲はスマートカード・リソース・データベースの全体が対象になります。また、特定の検索条件と一致するものを探索することもできます。スマートカード・リソースマネージャのコンテキストを変更する場合は、ScardEstablishContext 関数を使います。いくつかの情報については、特定のコンテキストを指定しないと、セキュリティの問題でアクセスが拒否されるかもしれません。

関数	説明
SCardGetProviderId	ある特定のスマートカード(GUID)のインターフェース・サービスの ID を取得する。
SCardListCards	ある特定のシステムユーザにアクセス可能なスマートカード・タイプのリストを取得する。
SCardListInterfaces	ある特定のスマートカード(GUID)のインターフェース・サービス・コンポーネントの固有な ID を取得する。
SCardListReaderGroups	あるスマートカード・グループのリストを取得する。
SCardListReaders	特定のスマートカード・グループの全スマートカード・タイプのリストを取得する。

## 1.2.2 スマートカード・データベース管理関数

これらの関数を使って、スマートカード・リソース・データベースを管理し、且つ指定されたリソース・コンテキストを持ったデータベースを更新することができます。

関数	説明
SCardAddReaderToGroup	特定のスマートカード・グループにスマートカード・リーダを追加する。
SCardForgetCardType	あるスマートカード・タイプを削除する。
ScardForgetReader	あるスマートカード・リーダを削除する。
ScardForgetReaderGroup	あるスマートカード・リーダ・グループを削除する。
ScardIntroduceCardType	新しくスマートカード・タイプを追加する。
ScardIntroduceReader	新しくリーダ・タイプを追加する。
SCardIntroduceReaderGroup	新しくリーダ・グループを追加する。
SCardRemoveReaderFromGroup	特定のリーダ・グループからあるリーダ・タイプを削除する。

### 1.2.3 リソースマネージャ・ハンドル関数

これらの関数を使って、スマートカード・リソースマネージャのクエリや管理の関数で使われるスマートカード・オペレーション用のコンテキスト・ハンドルを生成したり、クローズしたりすることができます。

関数	説明
ScardEstablishContext	スマートカード・データベースにアクセスする為のコンテキスト・ハンドルを生成する。
ScardReleaseContext	スマートカード・データベースにアクセスする為のコンテキスト・ハンドルをクローズする。

### 1.2.4 リソースマネージャ・ツール関数

この関数を使って、SCARD\_AUTOALLOCATE フラグが指定されている場合に、システム関数によって自動的に割り付けられたメモリ領域をリリースすることができます。

関数	説明
ScardFreeMemory	SCARD_AUTOALLOCATE フラグが指定されている場合に、システム関数によって自動的に割り付けられたメモリ領域をリリースする。

### 1.2.5 スマートカード監視関数

これらの関数を使って、アプリケーションがスマートカードやリーダーの現在のステータスを追跡することができます。そのほとんどは、ハードウェアのステータスを識別するために構文 SCARD\_READERSTATE を使用します。

関数	説明
SCardLocateCards	指定された ATR のストリングと一致するスマートカードを調べる。
SCardGetStatusChange	リーダーを変更する特定セットに含まれるスマートカードの現在の有効分までの実行をブロックする。
SCardCancel	特定のリソースマネージャのコンテキスト内の稼働中のアクションをすべて終了する。

### 1.2.6 スマートカード及びリーダーへのアクセス関数

これらの機能を使って、スマートカード上で、構文 SCARD\_IO\_REQUEST で始まる制御情報を含んでいるデ

ータ・ブロックを使用して、入出力操作を行なうことにより、指定されたスマートカードに接続し、且つアクセスすることができます。

関数	説明
ScardConnect	スマートカードに接続する。
ScardReconnect	スマートカードに再接続する。
ScardDisconnect	スマートカードへの接続を終了する。
ScardBegingTransaction	スマートカードへの排他的アクセスを開始し、他のアプリケーションによるスマートカードへのアクセスは保留にする。
ScardStatus	リーダの現在のステータスを提供する。
ScardTransmit	T=0 あるいは T=1 プロトコルを使ってスマートカードとデータを送受信する。

## 1.2.7 スマートカードへのダイレクトアクセス関数

Win32 プラットフォーム下のスマートカード・サブシステムは、アプリケーションが ISO7816 規格に完全に準じていないスマートカード装置にアクセスすることを可能にします。したがって、Win32 スマートカード関数は、アプリケーションがリーダへ制御コマンドとデータを直接送ることを可能にします。これらの関数を使うためには、コントロールしたい属性の各々に対して ID を定義する必要があります。Win32 スマートカード・サブセットはさらにいくつかの既存の属性も定義します。

関数	説明
ScardControl	リーダに対するダイレクトアクセス・コントロールを提供する。
ScardGetAttrib	リーダの属性を得る。
ScardSetAttrib	リーダの属性を設定する。

Windows 2000 以上のプラットフォームでは、オペレーティング・システムがインストールされる際に、スマートカード・サブシステムのコンポーネントが自動的に構築されています。

Win32 スマートカードの関数について、より詳細な情報に関しては、MSDN ドキュメントをご参照願います。

## 1.3 CryptoAPI を利用した ePass2003 アプリケーションの開発

Microsoft CryptoAPI は Win32 プラットフォーム上でデータの暗号化とセキュリティアプリケーションを設計・開発する方のために提供されています。CryptoAPI は ASN.1 暗号化/復号化インターフェース、ハッシュインターフェース、データ暗号化/復号化インターフェース、デジタル証明書管理インターフェース及びその他の重要な暗号化方法などから構成されています。データの暗号化および復号化は対称暗号化

方式と公開鍵アルゴリズムをサポートしており、Internet Explorer、Outlook など全ての Microsoft 製品とその他多くのサードパーティ製アプリケーションは CryptoAPI をベースとして開発されています。

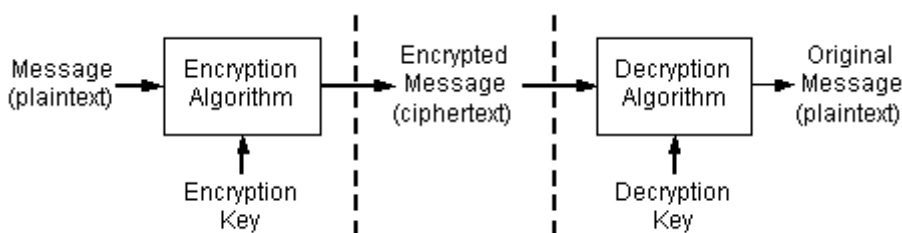
セキュアでないネットワーク上でセキュア通信を行うには、情報の秘匿性、認証、完全性の 3 つの要件が必要となります。CryptoAPI はこれらの 3 つの要件のほかに ASN.1 暗号化/復号化、データ暗号化/復号化、デジタル証明書及び証明書格納管理、CTL (Certificate Trust List) および CRL (Certificate Revocation List) 機能を提供しています。

### 1.3.1 情報の秘匿性

情報を秘匿する目的は、データ内容が正しいと認証された人のみが閲覧出来る様にすることです。一般的に情報の秘匿性は暗号法を利用する事で実現でき、データの暗号化では情報の秘匿性と通信を平文からハッシュデータに変換する事で実現しています。暗号化されたテキスト文章を暗号化キーが無い状態で推測または復元する事はほぼ不可能です。対象となるファイルは ASCII テキスト、データベースまたはその他のファイルなど様々なファイルになります。

※ここで言う「情報」とはデータのセットの事で、「平文」とは暗号化されていないデータを指し、「暗号化テキスト」とは平文が暗号化されたデータの事です。

暗号化テキストを利用する事でセキュアではないメディアやネットワーク上を、安全にデータ転送する事ができます。そして、データの受信者は受信後に平文に復号化します。



データの暗号化と復号化のコンセプトはとてもシンプルです。データの暗号化には暗号化キーが必要です。この「キー：鍵」とはドアを開けるための鍵の様な役割を果たします。復号化には復号化キーが必要になります。暗号化キーと復号化キーは同じである場合と異なる場合があります。暗号化キーは安全な場所に格納する必要があります。暗号化キーをユーザに渡す場合、転送方法はセキュアで、且つ信頼できるものでなければなりません。また、復号化キーは暗号化されたデータを復号するのに使用する為、復号化キーへのアクセス方法もまた厳重に管理する必要があります。

### 1.3.2 身分の証明

セキュアな通信を行う為の前提条件として、通信の両端はお互いの身分を知っている必要があります。身分確認の目的は通信に関わる人間や組織の身元を確認する為に行います。この身元を証明するドキュメントを証明書と呼びます。これは銀行でお金の引き出しを行う際に利用者は ID カードを利用して自己を証明するのと似ています。

例としてはパスポートなどがあります。飛行機の乗員は乗客の身元を確認する為にパスポートの確認を必要としますが、この場合の身分の証明は「パスポートの発行元が利用者の身元を証明した」という事実に基づいて信頼関係が成り立っています。

上記の例では身分証明書は実際の物理的なドキュメントになります。また、身分証明は受け取ったデータが正当な送主からのものかどうかを確認する為に利用されます。例として、AさんがBさんにデータを送った場合、Bさんは「受信したデータは、本当にAさんが送ったものなのか」を確認する必要があります。(Aさんと同姓同名の別人かも知れません) この認証を実現する為に、CryptoAPIでは電子署名と照合の関数が使われます。

なお、ネットワークとユーザには物理的な接続が無いので、証明書もネットワーク上で転送できる媒体である必要があります。また、証明書は信頼できる機関が発行したものである必要があります。電子証明書(単に証明書とも言います)はこの様な場合の身分証明として利用され、ネットワーク上で認証を行うのに有効な証明書となります。電子証明書は信頼できる組織やCA(Certificate Authority)と呼ばれる組織が発行した証明書で、証明書には公開鍵、証明書情報およびユーザ情報が含まれています。

CAは公開鍵の正当性とユーザ情報の正確性の確認が取れた際にのみ証明書を発行します。証明書の申請者とCAがやりとりする情報はUSBメモリーなどの物理的な媒体で行う事もできますが、一般的にはネットワークを介してやりとりされます。CAは申請者の要求と証明書の発行を処理するのに信頼できるサービスを利用しています。

### 1.3.3 完全性の確認

安全ではないメディアによる情報の転送は常に改竄されるリスクを伴っています。「シール」などは実際の世界で完全性を確認する為に利用されています。例えば、開封後は元に戻せない箱でシールが元の状態から損なわれていない場合、その品物は工場から出荷されてから変更されていないという事を証明できます。

同様の理由で、情報の受け取り者は正しい送り主からのものであるという証明が必要なだけでなく、情報そのものが改竄されていない事を確認する事が必要になります。完全性を確認する為に、情報と証明情報(ハッシュ値と呼びます)は一緒に送信される必要があります。情報と証明情報は電子証明書によって送信され、完全性を証明します。

### 1.3.4 CSP と暗号化プロセス

CryptoAPIはCSP(Cryptographic Service Providers)を利用してデータの暗号化/復号化および暗号化キーの管理を行います。全てのCSPは独立したモジュールになっており、理論上、CSPは特定のアプリケーションに依存せず、どのCSPでも利用する事ができますが、一部のアプリケーションではある特定のCSPのみ利用可能な場合があります。CSPとアプリケーションの関係はWindows GDIモデルによく似ており、CSPはグラフィックハードウェア・ドライバの様に動作します。

格納した暗号化キーのセキュリティ性はCSPの実装方法に依存しておりOSには依存しません。これは、



変更なしに様々なセキュリティ環境下で実行できるという事を意味します。セキュリティを維持する為にはアプリケーションプログラムと暗号化モジュールは厳密に管理される必要があります。以下はアプリケーション開発時の3つの留意事項です。

- 全ての暗号化キーは CSP 内部で生成され、アプリケーションは透過的に利用するので、アプリケーションは暗号化キーの内容に直接アクセスしてはいけません。この様にする事で暗号化キーが該当アプリケーションやダイナミックリンクライブラリから流出する事を防ぎ、様々な要因で暗号化キーが抽出されるのを防ぐ事ができます。
- アプリケーションは暗号化オペレーションの詳細な実装方法を指定してはいけません。CSP API はアプリケーションに暗号化と署名の操作を選択させる事を許可しますが、実際の実行は CSP 内部で行われるようにします。
- アプリケーション側では認証・証明などの処理を組み込まず、ユーザ認証は CSP により実行される様にします。この様にする事で将来的に指紋認証などのより厳密な認証を行う際にアプリケーション側で変更が生じないようにします。

最も簡単な CSP は Win32 ダイナミックリンクライブラリ (DLL) と署名ファイルで構成されます。正しい証明書ファイルを提供された時のみ CSP は CryptoAPI に認識され利用する事ができます。CryptoAPI は不正に署名が改竄されないように定期的に CSP の署名を確認します。

一部の CSP モジュールはローカル RPC またはハードウェアドライバプログラムによって別メモリ領域上で暗号化操作を実施します。別メモリ領域に暗号化キーを格納し暗号化操作を実行する事で鍵ペアがアプリケーションによって改竄されないようにする事ができます。

また、アプリケーションがある特定の CSP にのみ依存する事は推奨していません。例えば、Microsoft Base Cryptographic Provider では 40bit の通信キーと 512 ビットの公開鍵を利用できますが、別の CSP を利用した際には鍵長が変更される事があります。その為、アプリケーションはこれらのサイズのみを利用できるようにするのではなく、様々な CSP で利用できるようにすべきです。

### 1.3.5 CSP コンテキスト

アプリケーションが最初に呼び出す CryptoAPI 関数は「CryptAcquireContext」関数になります。この関数は特殊なキーコンテナを含んだ特殊なハンドラーを戻り値として返します。キーコンテナは事前に指定するかログオンユーザのデフォルトコンテナを利用する事ができます。また、「CryptAcquireContext」関数は新しいキーコンテナを作成する事ができます。

CSP モジュールにはそれぞれ CSP 名とタイプがあり、Windows にデフォルトでインストールされる CSP 名は「Microsoft Base Cryptographic Provider」で、タイプは「PROV\_RSA\_FULL」となっています。この CSP の名称は各 CSP で異なる名称になっている必要がありますが、タイプは同一でも構いません。

CSP の名称とタイプはアプリケーションが CSP ハンドラーを取得する為に「CryptAcquireContext」関数を実行する際に指定できます。CSP 名とタイプを指定した場合、完全に一致した CSP のみが呼び出され、

呼び出しが成功した時のみ CSP ハンドラーを返します。アプリケーションはこのハンドラーを利用して CSP および CSP 内のキーコンテナにアクセスする事ができます。

### 1.3.6 CryptoAPI アーキテクチャ

CSP アーキテクチャは主に以下の 5 つの要素で構成されています。

- 暗号化関数

暗号化関数は CSP ハンドラーへのリンクおよび生成の為に利用されます。これらの機能は CSP 名とタイプを指定する事でアプリケーションが特定の CSP を利用する事を許可することができます。

- キー生成機能は暗号化キーの生成と格納に利用されます。この機能には暗号化方式や暗号化の初期化などの機能も含まれています。
- キー交換機能はキーの交換及び転送に利用されます。

- 証明書エンコード/デコード関数

証明書エンコード/デコード関数はデータの暗号化/復号化およびデータのハッシュ値の計算に利用されます。

- 証明書格納関数

この関数は電子証明書のセットを管理する為に利用されます。

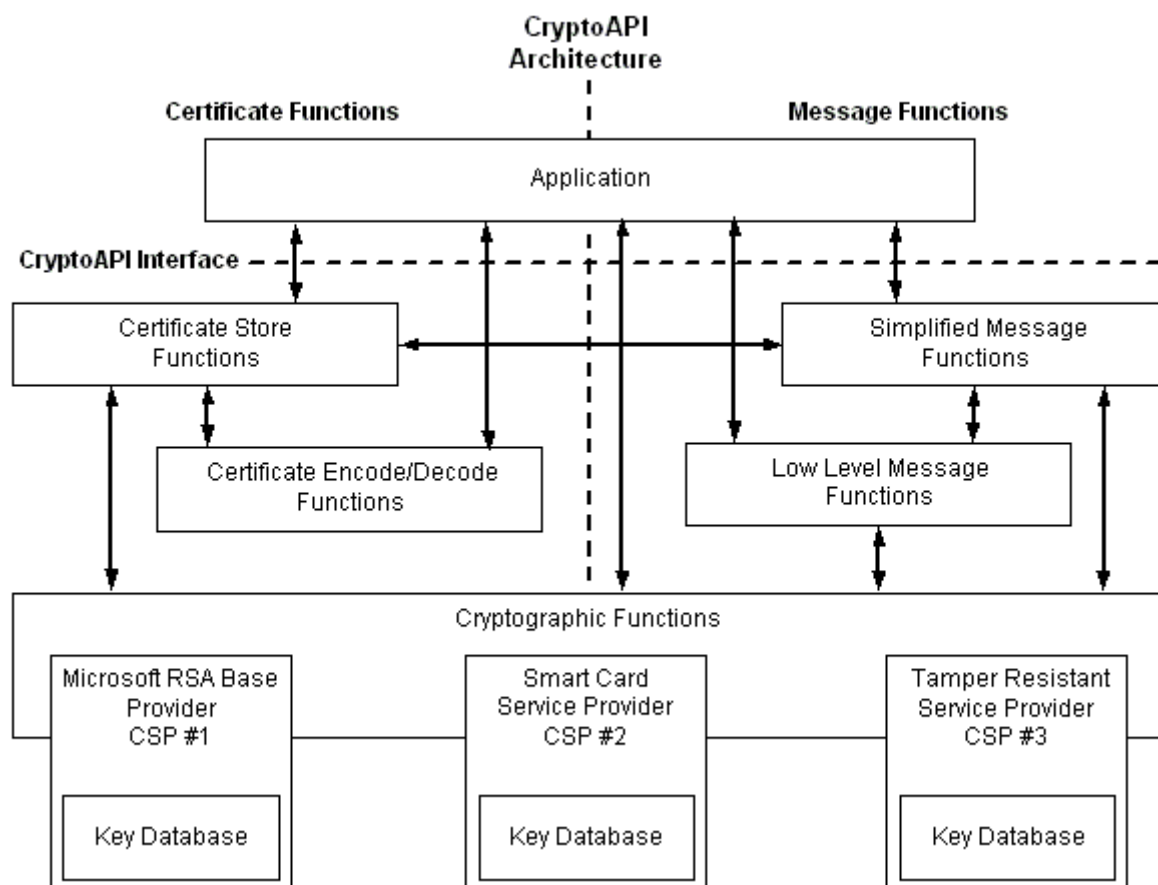
- 簡易メッセージ関数

この関数はメッセージ・データの暗号化/復号化、メッセージ・データへの署名および署名を照合するのに利用されます。

- ローレベルメッセージ関数

この関数は簡易メッセージ機能に実装されており、各メッセージに対してより詳細な操作を行います。





各機能の接頭語は以下のようになります。

関数	接頭語
暗号化関数	Crypt
電子証明書エンコード/デコード関数	Crypt
電子証明書格納関数	Store
簡易メッセージ関数	Message
ローレベルメッセージ関数	Msg

## 1.4 PKCS#11 を利用した ePass2003 アプリケーションの開発

インターネットの発展により、アプリケーションのセキュリティは非常に重要になり、さらにこうしたセキュリティ製品の発展はアプリケーション間で相互の互換性をも必要としてきました。このことから RSA 社はこれらの要求を満たす為に PKCS (Public Key Cryptographic Standard) を策定しました。

PKCS#11 は PKCS 規格の 1 つで、PKCS#11 (別名「Cryptoki」) は公開鍵アプリケーションおよび異なるメーカー間で発生する互換性の問題を解決する為に利用できます。PKCS#11 は「Cryptoki tokens」という一連のプログラミング・インターフェースモデルを定義しており、PK カードの PKCS#11 インターフェースも PKCS#11 v2.20 と互換性をもっています。

PK の PKCS#11 インターフェースを利用して開発を行うには、開発者は PKCS#11 について熟知している必要があります。PKCS#11 の規格に関連したドキュメントは RSA 社の以下の Web サイトからダウンロードすることができます。

<http://www.rsa.com/rsalabs/node.asp?id=2133>

## 第2章 CSP モジュール

この章では ePass2003 でサポートされる CryptoAPI 開発、特に ePass2003 の CSP インターフェース名、サポート関数、アルゴリズムの実装方法について説明します。

この章では以下のトピックについて説明します。

- 概要
- サポートしているアルゴリズム
- 関数のサポート状況
- 関数のパラメータ
- 関数呼び出し時の留意事項

### 2.1 概要

ePass2003 は CryptoAPI アプリケーションとシームレスな統合が行える様に標準的な CSP モジュールを提供しています。また、ePass2003 の CSP モジュールは Microsoft Crypto Service Provider と互換性があり、現在および将来に渡って CryptoAPI アプリケーションと互換性を維持することができます。

#### 2.1.1 プロファイル

- **Type:** PROV\_RSA\_FULL

この一般的な CSP タイプはデジタル署名とデータ暗号化／復号化のサポートを提供します。全ての公開鍵操作は RSA アルゴリズムを利用して処理されます。

- **Name:** EnterSafe ePass2003 CSP v1.0

Name にも表示されているように、ePass2003 のハードウェアタイプは“スマートカード”となります。

#### 2.1.2 機能

ePass2003 の CSP モジュールの機能は以下の通りです：

1. 安全な RSA キーペアの格納場所を提供している。
2. 様々な暗号化とハッシュアルゴリズムを提供している。
3. RSA オペレーションをハードウェア上でサポートする (最大: 2,048 ビット)。
4. ランダム番号生成機能をハードウェア上でサポートする。
5. マルチスレッドアクセスおよびマルチデバイス管理をサポートする。
6. 複数の証明書を利用するアプリケーションをサポートする。

7. PKCS#11 のデータフォーマットに準拠している。
8. 2つの鍵ペア (AT\_KEYEXCHANGE と AT\_SIGNATURE) と、複数の証明書による二重証明をサポートする。
9. すべての Windows プラットフォームをサポートする。(即ち、Windows 8.1/10/11, Windows Server 2016/2019/2022/2025 の 32/64 bits)
10. Office アプリケーションの暗号化/復号化、Windows ログオン、Internet Explorer および SSL Web サイトログオン、Outlook のメール暗号化などの既存の Windows アプリケーションに準拠している。

## 2.2 サポートされているアルゴリズム

ePass2003 の CSP モジュールでサポートされているアルゴリズムは以下の通りです：

アルゴリズム	標準の長さ (bit)	最小の長さ (bit)	最大の長さ (bit)	目的
CALG_RC2	40	8	1024	暗号化／復号化
CALG_RC4	40	8	2048	
CALG_DES	56	56	56	
CALG_3DES	192	192	192	
CALG_SHA1	160	160	160	ハッシュ計算
CALG_MD2	128	128	128	
CALG_MD5	128	128	128	
CALG_SSL3_SHAMD5	288	288	288	
CALG_RSA_SIGN or AT_SIGNATURE	1024	1024	4096	電子署名照合
CALG_RSA_KEYX or AT_KEYEXCHANGE	1024	1024	4096	暗号化／復号化、 電子署名照合

## 2.3 関数のサポート状況

以下のテーブルは CSP 関数のサポート状況をまとめたものです。「Not Implemented」とは CSP モジュールにはインターフェースがあるが、実装は未だ為されていない事を意味します。「Not Supported」は CSP モジュールにインターフェースが無い事を意味します。

ePass2003 の CSP タイプは PROV\_RSA\_FULL であり、以下のテーブルに記載されている中でいくつかの関数はサポートされていません。「Not Implemented」の関数では FALSE を返し、そのエラーコードが E\_NOTIMPL にセットされます。また、CryptoAPI アプリケーションは以下のインターフェース関数を直接コールする必要はありません。

関数名	機能の説明	サポート状況
<b>接続関数</b>		
CPAcquireContext	コンテキストを作成する。	Implemented
CPGetProvParam	CSP に関する情報を返す。	Implemented
CPReleaseContext	CPAcquireContext を使って作成されたコンテキストを開放する。	Implemented
CPSetProvParam	CSP のオペレーションを設定する。	Implemented
<b>キー生成関数とキー交換関数</b>		
CPDeriveKey	データハッシュからセッションキーを生成する。キーはユニーク（一意）である。	Implemented
CPDestroyKey	キーハンドルを開放する。ハンドルは無効になり、キーにはアクセスできなくなる。	Implemented
CPDuplicateKey	キーのコピーを生成する。	Not Supported
CPExportKey	CSP コンテナからキーをエクスポートする。	Implemented
CPGenKey	キーまたはキーペアを生成する。	Implemented
CPGenRandom	バッファに乱数を書き込む。	Implemented
CPGetKeyParam	暗号キーの属性を取得する。	Implemented
CPGetUserKey	CSP コンテナから永続キーペアを取得する。	Implemented
CPImportKey	プロブから CSP コンテナへキーペアをインポートする。	Implemented
CPSetKeyParam	キーの属性を設定する。	Implemented
<b>データ暗号化関数</b>		
CPDecrypt	暗号化されたデータを復元する。	Implemented
CPEncrypt	平文を暗号化する。	Implemented
<b>ハッシュ関数とデジタル署名関数</b>		
CPCreateHash	ハッシュオブジェクトを作成する。	Implemented
CPDestroyHash	ハッシュオブジェクトのハンドルを削除する。	Implemented
CPDuplicateHash	ハッシュオブジェクトのコピーを作成する。	Not Supported
CPGetHashParam	ハッシュオブジェクトの計算結果を取得する。	Implemented
CPHashData	ハッシュオブジェクトにデータを設定する。	Implemented
CPHashSessionKey	ハッシュオブジェクトの暗号キーを取得する。アプリケーションがキーにアクセスする事はない。	Not Implemented
CPSetHashParam	ハッシュオブジェクトの属性をカスタマイズする。	Implemented

CPSignHash	ハッシュオブジェクトに署名する。	Implemented
CPVerifySignature	デジタル署名を照合する。	Implemented

補足：CSP の標準規格には、OffloadModExpo という関数が定義されていますが、本 CSP モジュールではサポートされていません。

## 2.4 関数のパラメータ

### 2.4.1 CPAcquireContext

— *dwFlags*

以下の値をサポートしています：

CRYPT\_VERIFYCONTEXT, CRYPT\_NEWKEYSET, CRYPT\_DELETEKEYSET, CRYPT\_SILENT,

※CRYPT\_MACHINE\_KEYSET は対応していません。

— *pszContainer*

NULL または "" になるか、dwFlags の値によってはリーダー名の文字列になります（文字の長さは MAX\_PATH を超えてはいけません）。

### 2.4.2 CPGetProvParam

— *dwParam*

以下の値をサポートしています：

PP\_CONTAINER, PP\_ENUMALGS, PP\_ENUMALGS\_EX, PP\_ENUMCONTAINERS, PP\_IMPTYPE,  
PP\_NAME, PP\_VERSION, PP\_UNIQUE\_CONTAINER, PP\_PROVTYPE, PP\_SIG\_KEYSIZE\_INC,  
PP\_KEYX\_KEYSIZE\_INC, PP\_KEYSPEC

※PP\_KEYSET\_SEC\_DESCR, PP\_USE\_HARDWARE\_RNG などに対応していません。

— *dwFlags*

CSP の分析によると dwParam の値が PP\_ENUMALGS または PP\_ENUMALGS\_EX の時、dwFlags が CRYPT\_FIRST の際に列挙が始まり、その他の値（0 または CRYPT\_NEXT）の場合は次を列挙します。dwParam の値が PP\_ENUMCONTAINERS 時に、dwFlags が CRYPT\_FIRST(1)、CRYPT\_FIRST | CRYPT\_NEXT(3) の際に列挙が始まり、0 か CRYPT\_NEXT(2) の際に次を列挙します。dwFlags が CRYPT\_MACHINE\_KEYSET を対応していません。dwParam がその他の値の場合、dwFlags はチェックされません。

### 2.4.3 CPReleaseContext

——*dwFlags*

ゼロでなければなりません。

### 2.4.4 CPSetProvParam

——*dwParam*

以下の値をサポートしています：

PP\_KEYEXCHANGE\_PIN, PP\_SIGNATURE\_PIN.

pbData が NULL の場合はログアウトします。その他の値はサポートされていません。

——*dwFlags*

チェックされません。

### 2.4.5 CPDeriveKey

——*AlgId*

以下のアルゴリズムのみをサポートしています：

CALG\_RC2, CALG\_RC4, CALG\_DES, and CALG\_3DES.

——*dwFlags*

以下のケースの場合、エラーを返します：

(CRYPT\_CREATE\_SALT | CRYPT\_NO\_SALT), CRYPT\_PREGEN and CRYPT\_USER\_PROTECTED.

※その他の値はサポートされていません。

### 2.4.6 CPDestroyKey

該当なし。

### 2.4.7 CPDuplicateKey

サポートされていません。

### 2.4.8 CPExportKey

——*dwBlobType*

PUBLICKEYBLOB と SIMPLEBLOB のみサポートします。※PRIVATEKEYBLOB, OPAQUEKEYBLOB, PLAINTEXTKEYBLOB およびその他はサポートされていません。

——*dwFlags*

*dwBlobType* が PUBLICKEYBLOB または SIMPLEBLOB の場合、*dwFlags* はゼロです。その他の場合はこのパラメータの値は無視されます。

### 2.4.9 CPGenKey

——*Algid*

以下の値をサポートします：

CALG\_RSA\_KEYX, CALG\_RSA\_SIGN, AT\_KEYEXCHANGE, AT\_SIGNATURE, CALG\_DES, CALG\_RC2, CALG\_RC4 and CALG\_3DES. ※CALG\_3DES\_112 は次期バージョンでサポートされる予定です。

——*dwFlags*

サポートされていない CSP ではエラーメッセージを返します：

CRYPT\_CREATE\_SALT, CRYPT\_NO\_SALT, または CRYPT\_Pregen.

生成されるキー長はこのパラメータの最初の 2 バイトです（キーのデフォルト長は 0 です）最後の 2 バイトは無視されます。

### 2.4.10 CPGenRandom

該当なし。

### 2.4.11 CPGetKeyParam

この関数は CALG\_RSA\_KEYX, CALG\_RSA\_SIGN, AT\_KEYEXCHANGE, AT\_SIGNATURE, CALG\_DES, CALG\_RC2, CALG\_RC4 及び CALG\_3DES のキータイプのみサポートします。

——*dwParam*

キータイプが CALG\_RSA\_KEYX, CALG\_RSA\_SIGN, AT\_KEYEXCHANGE 及び AT\_SIGNATURE の場



合、値は KP\_PERMISSIONS, KP\_CERTIFICATE, KP\_BLOCKLEN, KP\_KEYLEN または KP\_ALGID になります。

キータイプが CALG\_RC2 の場合、値は KP\_BLOCKLEN, KP\_EFFECTIVE\_KEYLEN, KP\_KEYLEN, KP\_ALGID または KP\_SALT になります。

キータイプが CALG\_RC4 の場合、値は KP\_BLOCKLEN (戻り値 0), KP\_KEYLEN, KP\_ALGID または KP\_SALT になります。

キータイプが CALG\_3DES 及び CALG\_DES の場合、値は KP\_BLOCKLEN, KP\_KEYLEN または KP\_ALGID になります。

#### ——dwFlags

ゼロである必要があります。

### 2.4.12 CPGetUserKey

#### ——dwParam

以下の値をサポートしています：

AT\_KEYEXCHANGE, AT\_SIGNATURE, and (AT\_KEYEXCHANGE | AT\_SIGNATURE).

### 2.4.13 CImportKey

#### ——pbData

この keyBlob は SIMPLEBLOB, PUBLICKEYBLOB と PRIVATEKEYBLOB をサポートしています。

#### ——dwFlags

この値は無視されます。

### 2.4.14 CPSetKeyParam

#### ——dwParam

CALG\_RC2, CALG\_DES 及び CALG\_3DES のキータイプの場合、値は KP\_IV です。

CALG\_RC2 のキータイプの場合、値は KP\_EFFECTIVE\_KEYLEN です。

CALG\_RC2 及び CALG\_RC4 のキータイプの場合、値は KP\_SALT または KP\_SALT\_EX です。

CALG\_RSA\_KEYX, CALG\_RSA\_SIGN, AT\_KEYEXCHANGE 及び AT\_SIGNATURE のキータイプの場合、

値は KP\_CERTIFICATE です。

——*dwFlags*

ゼロでなければなりません。

### 2.4.15 CPDecrypt

以下のキータイプをサポートしています：

CALG\_RSA\_KEYX, AT\_KEYEXCHANGE, CALG\_RC2, CALG\_DES, CALG\_3DES and CALG\_RC4.

——*dwFlags*

ゼロでなければなりません。

### 2.4.16 CPEncrypt

以下のキータイプをサポートしています：

CALG\_RSA\_KEYX, AT\_KEYEXCHANGE, CALG\_RC2, CALG\_DES, CALG\_3DES and CALG\_RC4.

——*dwFlags*

ゼロでなければなりません。

### 2.4.17 CPCreateHash

——*AlgId*

以下のアルゴリズムをサポートしています：

CALG\_MD2, CALG\_MD5, CALG\_SHA1 and CALG\_SSL3\_SHAMD5.

——*dwFlags*

ゼロでなければなりません。

### 2.4.18 CPDestroyHash

該当なし。

### 2.4.19 CPDuplicateHash

サポートされていません。

### 2.4.20 CPGetHashParam

*—dwParam*

HP\_ALGID, HP\_HASHSIZE と HP\_HASHVAL 値をサポートしています。

*—dwFlags*

ゼロでなければなりません。

### 2.4.21 CPHashData

*—dwFlags*

ゼロでなければなりません。CRYPT\_USERDATA の値はサポートされていません。

### 2.4.22 CPHashSessionKey

実装されていません。FALSE を返し、E\_NOTIMPL に ErrorCode をセットします。

### 2.4.23 CPSetHashParam

*—dwParam*

HP\_HASHVAL の値のみサポートされます。

*—dwFlags*

ゼロでなければなりません。

### 2.4.24 CPSignHash

*—sDescription*

この値は無視されます。

*—dwFlags*

CRYPT\_NOHASHOID のみサポートされます。他の値は無視されます。

## 2.4.25 CPVerifySignature

—*sDescription*

この値は無視されます。

—*dwFlags*

いかなる値もサポートしていません。

## 2.5 関数呼び出し時の留意事項

### 2.5.1 概要

全ての CSP 関数で最初に呼ばれるのが CPAcquireContext 関数です。上位アプリケーションはどのキーコンテナで操作するのかを確認する為にこの関数を呼び出します。各キーコンテナは同一タイプの 1 つの RSA キーペアと複数のセッションキーを格納する事ができます。RSA キーペアは永続的なオブジェクトで、セッションキーは実行時にのみ存在するキーです。アプリケーションがコンテナ内の秘密鍵にアクセスする必要がある場合は、ePass2003 の CSP モジュールはユーザの認証を必要とします。

開発者がダイアログボックスを表示したくないという場合は、CRYPT\_SILENT フラグにセットする事ができます。しかし ePass2003 では CPSetProvParam をユーザ識別データの設定用としてサポートしていない為、秘密鍵と格納されたデータへのアクセスするオペレーションは全てエラーとなります。

### 2.5.2 開発プログラムサンプル

開発者は SDK の¥ Samples¥CryptAPI 内にある ePass2003 の CryptoAPI インターフェースを使ったサンプルプログラムを利用する事ができます。なお、いくつかのサンプルは Microsoft の Platform SDK が必要となります。

## 第3章 PKCS#11 モジュール

この章では ePass2003 によってサポートされている PKCS#11 について紹介します。特に ePass2003 の PKCS#11 インターフェース名、サポート関数、アルゴリズムの実装について説明します。

この章では以下のトピックについて説明します。

- 概要
- サポートされている PKCS#11 オブジェクト
- サポートされているアルゴリズム
- サポートされている PKCS#11 インターフェース関数

### 3.1 概要

ePass2003 PKCS#11 インターフェースは Win32 ダイナミックリンクライブラリ（DLL）として提供されます。開発者は静的に lib ファイルを利用するか動的にアクセスをすることができます。PKCS#11 に関連するファイルは以下の通りです。

File	SDK Path
pkcs11.h	\Include\pkcs11 (RSA により提供)
pkcs11f.h	\Include\pkcs11 (RSA により提供)
pkcs11t.h	\Include\pkcs11 (RSA により提供)
Cryptoki.h	\Include\pkcs11 (RSA により提供)
cryptoki_ext.h	¥Include¥pkcs11 (ePass2003 拡張アルゴリズムと戻り値を含む)
cryptoki_win32.h	¥Include¥pkcs11 (Windows 環境で最初の 3 つのヘッダーファイルに必要とされるタイプ定義)
cryptoki_linux.h	¥Include¥pkcs11 (Linux 環境で最初の 3 つのヘッダーファイルに必要とされるタイプ定義)
auxiliary.h	\Include\pkcs11 (拡張関数に関する ePass2003 の定義)
eps2003csp11.lib	¥Lib (PKCS#11 インターフェース・ライブラリー)

eps2003csp11.dll が ePass2003 の中心となるライブラリーファイルであり、システムフォルダにインストールされます。RSA PKCS#11 標準規格で定義されているインターフェース関数を全て実装しています。

開発者がこれらのインターフェースを利用する場合で、且つ PKCS#11 規格の標準インターフェースのみを利用する場合は、プロジェクトに cryptoki\_win32.h ファイル（Windows プラットフォーム）あるいは cryptoki\_linux.h ファイル（Linux プラットフォーム）を必ずインクルードする必要があります。

又、開発者が ePass2003 の特有な拡張関数とアルゴリズムを利用する場合は、cryptoki\_ext.h ファイルのみをインクルードします。このヘッダーファイルはその他のヘッダーファイルを全てインクルードしています。

開発者が LoadLibrary で esp2003csp11.dll を利用したくないという場合には、esp2003csp11.lib をプロジ

ェクトにインクルードして静的に利用することも可能です。

## 3.2 サポートしている PKCS#11 オブジェクト

ePass2003 PKCS#11 モジュールは以下のオブジェクトの生成と利用の双方をサポートしています。

クラスオブジェクト	説明
CKO_DATA	アプリケーションによって定義されたオブジェクト。オブジェクトの構造はアプリケーションによって決定される。データのレンダリングもアプリケーションによって処理される。
CKO_SECRET_KEY	対称暗号化アルゴリズムのキー
CKO_CERTIFICATE	X.509 デジタル証明書オブジェクト
CKO_PUBLIC_KEY	RSA 公開鍵オブジェクト
CKO_PRIVATE_KEY	RSA 秘密鍵オブジェクト
CKO_MECHANISM	アルゴリズムオブジェクト

全てのオブジェクトはそのオブジェクトの存続期間によって2つのグループに分ける事ができます。

一つ目のグループは永続的に格納されるオブジェクトのグループです。このグループのオブジェクトはアプリケーションによって削除されるまで ePass2003 上の安全なストレージゾーンに格納され続けます。もう一つのグループはセッションオブジェクトです。このグループのオブジェクトは一時的なセッション通信時に利用され、セッションが終了するとオブジェクトも同様に削除されます。

存続期間を決めるオブジェクトの属性は CKA\_TOKEN で、ブール値です。全てのオブジェクトはこの属性を持っており、開発者は ePass2003 のメモリーサイズの上限を考慮して各々のオブジェクトの存続期間を決める必要があります。

基本的には、重要なオブジェクトのみを ePass2003 のメモリーに格納するようにします。

存続期間に加え、PKCS#11 オブジェクトは異なるアクセス特権を持っています。全てのオブジェクトは異なるアクセス特権によって2つのタイプに分けることができます。

一つ目のタイプはどのユーザでもアクセスができるパブリックオブジェクトです。もう一つのタイプは認証を行ったユーザのみがアクセスできるプライベートオブジェクトです。

アクセスタイプを決めるオブジェクトの属性は CKA\_PRIVATE で、ブール値です。全てのオブジェクトはこの属性を持っています。アプリケーションは実際の用法を考慮してパブリックまたはプライベートのどちらかを選択する事ができますが、プライベート格納領域とパブリック格納領域には上限があることを考慮する必要があります。これら2つの格納領域は独立しており、アプリケーションは各々の領域を定義する必要がありますが、サイズは ePass2003 の初期化時（出荷時）に設定され、以後はサイズ変更を行う事が出来ないので十分な注意が必要です。

### 3.3 サポートされているアルゴリズム

以下のテーブルは ePass2003 PKCS#11 モジュールでサポートされているアルゴリズムです：

暗号化アルゴリズム	暗号化 / 復号化	署名確認	ハッシュ	キーペア生成	パッケージ
CKM_RSA_PKCS_KEY_PAIR_GEN				√	
CKM_RSA_PKCS	√	√			√
CKM_MD2_RSA_PKCS		√			
CKM_MD5_RSA_PKCS		√			
CKM_SHA1_RSA_PKCS		√			
CKM_SHA224_RSA_PKCS		√			
CKM_SHA256_RSA_PKCS		√			
CKM_SHA384_RSA_PKCS		√			
CKM_SHA512_RSA_PKCS		√			
CKM_RSA_X9_31_KEY_PAIR_GEN				√	
CKM_RSA_X9_31		√			
CKM_SHA1_RSA_X9_31		√			
CKM_RSA_PKCS_OAEP	√				
CKM_RSA_PKCS_PSS		√			
CKM_SHA1_RSA_PKCS_PSS		√			
CKM_SHA256_RSA_PKCS_PSS		√			
CKM_SHA224_RSA_PKCS_PSS		√			
CKM_SHA384_RSA_PKCS_PSS		√			
CKM_SHA512_RSA_PKCS_PSS		√			
CKM_RSA_X_509	√	√			√
CKM_EC_KEY_PAIR_GEN				√	
CKM_ECDSA		√			
CKM_ECDSA_SHA1		√			
CKM_DH_PKCS_KEY_PAIR_GEN				√	
CKM_RC2_KEY_GEN				√	
CKM_RC2_ECB	√				
CKM_RC2_CBC	√				
CKM_RC2_CBC_PAD	√				
CKM_RC4_KEY_GEN				√	
CKM_RC4	√				
CKM_DES_KEY_GEN				√	

CKM_DES_ECB	✓				
CKM_DES_CBC	✓				
CKM_DES_CBC_PAD	✓				
CKM_DES3_KEY_GEN				✓	
CKM_DES3_ECB	✓				
CKM_DES3_CBC	✓				
CKM_DES3_CBC_PAD	✓				
CKM_AES_KEY_GEN				✓	
CKM_AES_ECB	✓				
CKM_AES_CBC	✓				
CKM_AES_CBC_PAD	✓				
CKM_MD2			✓		
CKM_MD5			✓		
CKM_SHA_1			✓		
CKM_SHA224			✓		
CKM_SHA256			✓		
CKM_SHA384			✓		
CKM_SHA512			✓		

以下のテーブルは ePass2003 PKCS#11 モジュールでサポートされているキー長です：

暗号化アルゴリズム	キー長（鍵長）
CKM_RSA_KEY_PAIR_GEN	1024, 2048, 3072, 4096 ビット
CKM_RC2_KEY_GEN	1 - 128 バイト
CKM_RC4_KEY_GEN	1 - 256 バイト
CKM_DES_KEY_GEN	8 バイト
CKM_DES3_KEY_GEN	24 バイト
CKM_GENERIC_SECRET_KEY_GEN	1 - 256 バイト
CKM_AES_KEY_GEN	128, 192, 256 ビット

### 3.4 サポートされている PKCS#11 インターフェース関数

PKCS#11 は Cryptoki ハードウェアの為に万国共通の標準規格ですが、各ハードウェアベンダーの実装方法には若干の差異があり、ePass2003 PKCS#11 インターフェースモジュールも規格と若干異なる所があります。

また、PKCS#11 で定義されているいくつかの関数は実装されていません。アプリケーションがサポートされていない関数を呼び出すと、CKR\_FUNCTION\_NOT\_SUPPORT という値を返します。



注意: ePass2003 は PKCS#11 で定義されている「token」に相当します。

PKCS#11 ではスマートカードリーダーを「スロット」と呼びますが、ePass2003 を利用するに当たってスマートカードリーダーは不要なので、スロットは単なる仮想デバイスに過ぎません。但し、アプリケーション側から見て仮想デバイスであっても特に差異はありません。

以下のテーブルは PKCS#11 2.11 で定義されている関数のリストです。

関数名	機能の説明	サポート状況
<b>基本関数</b>		
C_Initialize	ライブラリを初期化する関数。他の関数を呼ぶ前にライブラリを初期化する必要があるが、例外として、C_GetFunctionList 関数は利用できる。	Implemented
C_Finalize	ライブラリを終了する。	Implemented
C_GetInfo	cryptoki ライブラリの情報を取得する。	Implemented
C_GetFunctionList	cryptoki ライブラリのエントリポインタリストを取得する。	Implemented
<b>スロットおよびトークン管理関数</b>		
C_GetSlotList	スロットリストを取得する。	Implemented
C_GetSlotInfo	スロット情報を取得する。	Implemented
C_GetTokenInfo	スロット内のトークン情報を取得する。	Implemented
C_WaitForSlotEvent	スロットのイベント（トークンの挿入など）の発生を待つ。	Implemented
C_GetMechanismList	トークンでサポートしているアルゴリズムの一覧を取得する。	Implemented
C_GetMechanismInfo	アルゴリズムの詳細情報を取得する。	Implemented
C_InitToken	トークンを初期化する。	Implemented
C_InitPIN	User PIN を初期化する。	Implemented
C_SetPIN	現在の User PIN を変更する。	Implemented
<b>セッション管理関数</b>		
C_OpenSession	アプリケーションとトークンの間のセッションを開く。	Implemented
C_CloseSession	セッションを閉じる。	Implemented

C_CloseAllSessions	開いている全てのセッションを閉じる。	Implemented
C_GetSessionInfo	セッション情報を取得する。	Implemented
C_GetOperationState	現在の処理状態を取得する。	Not Implemented
C_SetOperationState	C_GetOperationState 関数によって得られた状態を利用してライブラリの処理状態を再開する。	Not Implemented
C_Login	トークンへログオンする。	Implemented
C_Logout	トークンからログアウトする。	Implemented
<b>オブジェクト管理関数</b>		
C_CreateObject	新規 Cryptoki オブジェクトを生成する。	Implemented
C_CopyObject	オブジェクトのコピーを作成する。	Not Implemented
C_DestroyObject	オブジェクトを破棄する。	Implemented
C_GetObjectSize	オブジェクトのサイズを取得する。	Not Implemented
C_GetAttributeValue	オブジェクトの属性を取得する。	Implemented
C_SetAttributeValue	オブジェクトの属性を設定する。	Implemented
C_FindObjectsInit	オブジェクトの検索処理の初期設定を行う。	Implemented
C_FindObjects	オブジェクトの検索処理を実行する	Implemented
C_FindObjectsFinal	オブジェクトの検索処理を終了する	Implemented
<b>暗号化関数</b>		
C_EncryptInit	暗号化処理の初期設定を行う。	Implemented
C_Encrypt	データの暗号化を行う。	Implemented
C_EncryptUpdate	データの暗号化を継続する。	Implemented
C_EncryptFinal	データの暗号化を終了する。	Implemented
<b>復号化関数</b>		
C_DecryptInit	復号化処理の初期設定を行う。	Implemented
C_Decrypt	データの復号化を行う。	Implemented
C_DecryptUpdate	データの復号化を継続する。	Implemented
C_DecryptFinal	データの復号化を終了する。	Implemented
<b>ダイジェスト関数</b>		
C_DigestInit	ダイジェスト処理の初期設定を行う。	Implemented
C_Digest	入力されたデータのダイジェスト	Implemented

	化を行う。	
C_DigestUpdate	ダイジェスト化を継続する。	Implemented
C_DigestKey	複数ブロックのダイジェストを継続する。	Not Implemented
C_DigestFinal	データのダイジェスト化を終了する。	Implemented
<b>署名関数</b>		
C_SignInit	署名処理の初期設定を行う。	Implemented
C_Sign	署名処理を行う。	Implemented
C_SignUpdate	署名処理の操作を継続する。	Implemented
C_SignFinal	署名処理の操作を終了する。	Implemented
C_SignRecoverInit	署名処理の復元を可能にするための初期設定を行う。	Implemented
C_SignRecover	署名処理の復元を行う。	Implemented
<b>署名の照合関数</b>		
C_VerifyInit	署名照合処理の初期設定を行う。	Implemented
C_Verify	署名照合を行う。	Implemented
C_VerifyUpdate	署名照合を継続する。	Implemented
C_VerifyFinal	署名照合を終了する。	Implemented
C_VerifyRecoverInit	署名照合の復元を可能にするための初期設定を行う。	Implemented
C_VerifyRecover	署名照合の復元を行う。	Implemented
<b>ダイジェスト暗号化関数</b>		
C_DigestEncryptUpdate	ダイジェスト化と暗号化を継続する。	Not Implemented
C_DecryptDigestUpdate	ダイジェスト化と復号化を継続する。	Not Implemented
C_SignEncryptUpdate	署名と暗号化を継続する。	Not Implemented
C_DecryptVerifyUpdate	署名と復号化を継続する。	Not Implemented
<b>鍵管理関数</b>		
C_GenerateKey	キーを生成し、新しいキーオブジェクトを作成する。	Implemented
C_GenerateKeyPair	キーペアを生成し、新しい公開キーオブジェクトを作成する。	Implemented
C_DeriveKey	秘密キーまたは暗号化キーを引き出す。	Not Implemented
C_WrapKey	秘密キーまたは暗号化キーをラッ	Implemented

	プ（包む）する。	
C_UnwrapKey	秘密キーまたは暗号化キーをアンラップ（開ける）する。	Implemented
<b>乱数生成関数</b>		
C_GenerateRandom	乱数を生成する。	Implemented
<b>並列管理関数</b>		
C_GetFunctionStatus	以前のバージョンで使用された関数で現在未サポート。	Not Implemented
C_CancelFunction	以前のバージョンで使用された関数で現在未サポート。	Not Implemented

## 第4章 スマートカード・ミニドライバ モジュール

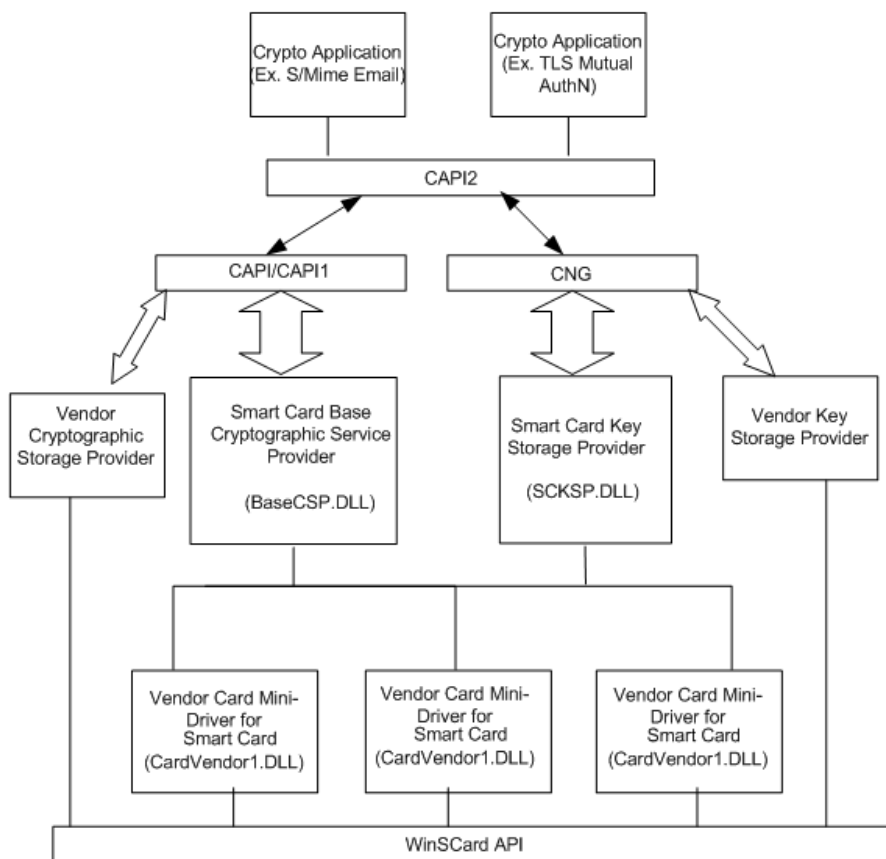
この章では ePass2003 におけるスマートカード・ミニドライバ・インターフェースでの開発について紹介します。特にスマートカード・ミニドライバ・インターフェース名、サポート関数、アルゴリズムの実装状況について説明します。 この章では以下のトピックについて説明します。

- スマートカード・ミニドライバの概要
- サポートされているアルゴリズム
- サポートされている スマートカード・ミニドライバ・インターフェース関数

### 4.1 スマートカード・ミニドライバの概要

スマートカード・ミニドライバ・インターフェースは、Microsoft Smart Card Base Cryptographic Provider と Microsoft Smart Card Key Storage Provider の下位レイヤーに位置しています。そして、復号化アルゴリズムとファイル格納の関数を提供しています。

ePass2003 では Microsoft Smart Card Cryptographic Service Provider (CSP)、Crypto API Next Generation (CNG)、Key Storage Provider (KSP)をインプリする為に標準のスマートカード・ミニドライバ・モジュールを提供しています。ePass2003 のミニドライバ・モジュールは Microsoft Windows のスマートカード・ミニドライバ・標準コーディングと完全互換です。また現在及び将来に渡って Crypto API アプリケーションと互換性があります。以下は、スマートカード・ミニドライバのプロセス・フローです。



### 4.1.1 プロファイル

サポート対象バージョン：V4、V5、V6

V4 では、CSP の基本的な関数をサポートします。ユーザ PIN：1 つ、管理者 PIN：1 つ。

V5 では、AT\_ECDHE\_\* アルゴリズムをサポートします。

V6 では、1 台のコンテナにつき 1 つの PIN の紐づけをサポートします。

V7 では、安全なキー注入をサポートします。

説明: Microsoft Base Smart Card Crypto Provider / Microsoft Smart Card Key Storage Provider の

レジストレーション・メカニズム:

	Windows XP and Windows 2003	Windows Vista/2008/7
CAPI-1	Registry file  Must set "Crypto Provider" registry key.	Manifest  Must set "Crypto Provider" registry key.
CNG	Not supported	Manifest  Must set "Smart Card Key Storage Provider" key.

"Crypto Provider" と "Microsoft Base Smart Card Crypto Provider" は同じ意味です。

"Smart Card Key Storage Provider" と "Microsoft Smart Card Key Storage Provider" は同じ意味です。

### 4.1.2 概要

ePass2003 のスマートカード・ミニドライバには以下の機能があります：

- セキュアな RSA/ECC のキーペアの格納場所を提供している。
- RSA の 2048 ビットをハードウェア上でサポートしている。
- ECC の 256 ビットをハードウェア上でサポートしている。
- バイナリファイルの生成と削除をサポートしている。
- 乱数生成機能をハードウェア上でサポートしている。
- マルチスレッド・アクセスおよびマルチデバイス管理をサポートしている。
- 複数の証明書を利用するアプリケーションをサポートしている。
- PKCS#11 のデータフォーマットと互換性がある。
- 2 重の証明書をサポートする。～1 台のコンテナには 2 つのキーペア (AT\_KEYEXCHANGE と AT\_SIGNATURE) と、それに対応した証明書を含む。
- Windows2000 以上のプラットフォームをサポートしている。(但し、Windows 2000/Server 2003 の場

合は、MS のパッチ KB909520 が必要。)

- Office アプリケーションの暗号化/復号化、Internet Explorer および SSL Web サイトログオン、Outlook の e メール暗号化、スマートカード・ログオン、VPN 接続などの既存の Windows アプリケーションと互換性がある。

## 4.2 サポートされているアルゴリズム

以下は ePass2003 のスマートカード・ミニドライバ・モジュールでサポートされているアルゴリズムです：

アルゴリズム	標準の長さ (bit)	最小の長さ (bit)	最大の長さ (bit)	目的
AT_ECDSA_P256	256	256	256	電子署名
AT_SIGNATURE	1024	1024	2048	電子署名照合
AT_KEYEXCHANGE	1024	1024	2048	暗号化／復号化、 電子署名照合

## 4.3 関数のサポート状況

以下のテーブルはスマートカード・ミニドライバ・インターフェース関数のサポート状況をまとめたものです。「Not Implemented」とは CSP モジュールにはインターフェースがあるが、実装は未だ為されていない事を意味します。「Not Supported」は CSP モジュールにインターフェースが無い事を意味します。「Not implemented」と記された関数を使うと、「SCARD\_E\_UNSUPPORTED\_FEATURE」が返されます。

関数名	機能の説明	サポート状況
<b>接続用関数</b>		
CardAcquireContext	スマートカードと、CSP 又は KSP のどちらかとの間の通信を初期化する。	Implemented
CardDeleteContext	スマートカード・モジュールと、CSP 又は KSP との間の通信を終了し、スマートカー	Implemented

	ドによって割り当てられたリソースをクリーンアップする。	
CardGetProperty	スマートカード・ミニドライバの基本属性情報を取得する。	Implemented
CardSetProperty	スマートカード・ミニドライバの基本属性情報を設定する。	Implemented
<b>PIN 管理用関数</b>		
CardGetChallenge	チャレンジ&レスポンス方式でユーザを認証する。	Implemented
CardAuthenticatePin	スマートカードへの PIN 入力でユーザを認証する。	Implemented
CardAuthenticateChallenge	ユーザを認証するためにスマートカードが発行したチャレンジに対してレスポンスを提供する。	Implemented
CardDeauthenticate	スマートカードをリセットせずに、ユーザ又は管理者の認証の効力を持続させる。	Implemented
CardUnblockPin	PIN の入力ミスが限度を超えたためにブロックされてしまったスマートカードをアンブロックする。	Implemented
CardChangeAuthenticator	スマートカードと特定のユーザタイプに関連した認証データを変更する。	Implemented
CardAuthenticateEx	スマートカードへの PIN による認証オペレーションを行う。	Implemented
CardChangeAuthenticatorEx	スマートカードと特定のユーザタイプに関連した認証データを、指定された PIN を使って変更する。	Implemented



CardDeauthenticateEx	スマートカードをリセットせずに、ユーザ又は管理者の認証の効力を指定されたPINを使って持続させる。	Implemented
CardGetChallengeEx	チャレンジ&レスポンス認証方式で指定されたロールの範囲内でユーザを認証する。	Implemented
<b>ファイルシステム管理用関数</b>		
CardCreateDirectory	スマートカード上のファイルシステム内にサブディレクトリーを作成する。	Implemented
CardDeleteDirectory	スマートカードからディレクトリーを削除する。	Implemented
CardReadFile	スマートカード上のファイルの内容をバッファに読み込む。	Implemented
CardCreateFile	スマートカードの指定されたディレクトリにファイルを作成する。	Implemented
CardGetFileInfo	スマートカード内のファイルのサイズやアクセス権限の情報を取得する。	Implemented
CardWriteFile	バッファのデータをスマートカード上のファイルに書き込む。	Implemented
CardDeleteFile	スマートカードからファイルを削除する。	Implemented
CardEnumFiles	スマートカード上の指定されたディレクトリで利用可能なファイル名一覧を得る。	Implemented
CardQueryFreeSpace	スマートカード上で利用可能なメモリーサイズを取得する。	Implemented

キーコンテナ管理用関数		
CardCreateContainer	スマートカードに新しいキーコンテナを作成する。	Implemented
CardCreateContainerEx	スマートカードに PIN と紐づけて新しいキーコンテナを作成する。	Implemented
CardDeleteContainer	スマートカードからキーコンテナを削除する。	Implemented
CardGetContainerInfo	スマートカード上のキーコンテナに関する情報を取得する。	Implemented
CardGetContainerProperty	スマートカード上のキーコンテナの属性情報を取得する。	Implemented
CardSetContainerProperty	スマートカード上のキーコンテナの属性情報を設定する。	Not Implemented
CardQueryKeySizes	スマートカードでサポートされている公開キーの長さを取得する。	Implemented
CardQueryCapabilities	スマートカードで提供されている機能に関する情報を取得する。	Implemented
非対称キー暗号化操作関数		
CardRSADecrypt	指定された秘密キーを使って、指定されたデータの RSA 復号化を行う。	Implemented
CardSignData	指定されたデータブロックに対する電子署名を作成する。	Implemented
ECDH Algorithm Function		
CardConstructDHAgreement	DH 法(Diffie-Hellman キー交換)プロトコルを使って秘密キー交換を行う。	Implemented

	※Windows Server 2003, XP, 2000 Server, 2000 Professional では未サポート。	
CardDestroyDHAgreement	スマートカード上の指定されたキーコンテナから DH 法のプロトコルを解除する。	Implemented
CardDeriveKey	DH 法プロトコルに基づいたセッションキーを生成する。	Implemented
<b>安全にキー投入を行うための関数 (Version 7 よりサポート)</b>		
MDImportSessionKey		Not Implemented
MDEncryptData		Not Implemented
CardImportSessionKey		Not Implemented
CardGetSharedKeyHandle		Not Implemented
CardGetAlgorithmProperty		Not Implemented
CardGetKeyProperty		Not Implemented
CardSetKeyProperty		Not Implemented
CardDestroyKey		Not Implemented
CardProcessEncryptedData		Not Implemented

## 4.4 関数のパラメータ

関数のパラメータ定義の詳細については、Microsoft社の以下を参照願います:

<http://www.microsoft.com/whdc/device/input/smartcard/sc-minidriver.msp>

## 4.5 関数呼び出し時の留意事項

### 4.5.1 概要

スマートカード・ミニドライバ・インターフェース関数は、「Microsoft Base Smart Card Crypto Provider(CSP)」と「Microsoft Smart Card Key Storage Provider(KSP)」によってコールされる関数であって、スマートカード・ミニドライバ・インターフェースを直接コールする必要はありません。The way to call 「Microsoft Base Smart Card Crypto Provider(CSP)」と「Microsoft Smart Card Key Storage Provider(KSP)」をコールする方法は、ユーザ自身によって開発されたような他の CSP の場合と同じです。

もしユーザが外部より基本となる CSP を介して公開キーペアをインポートしたいのであれば、ユーザ側は基本 CSP のコンフィギュレーションを変更し、レジストリ・リストをオープンして、「HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider\Microsoft Base Smart Card Crypto Provider」を探し出し、「AllowPrivateExchangeKeyImport」と「AllowPrivateSignatureKeyImport」の値を“1”に変更する必要があります。

## 付録・用語と略称

用語・略称	説明
ePass2003	Feitian Technologies 社が開発した FIPS 認定モジュール搭載のスマートカードベースのトークンです。 PKI アプリケーション・システムのために設計されています。
CryptoAPI Interface (CAPI)	Microsoft 社が提供する暗号化、デジタル署名、認証などに関する API 群の総称です。 Windows のプラットフォーム上において、サードパーティのアプリケーションやユーザがこれらの API を利用してデータを暗号化したり、デジタル署名を付加したり、デジタル証明書の情報を取り出したりすることができます。
Smart Card Minidriver Interface	マイクロソフトによって提供された暗号化オペレーションのためのインターフェースです。 それは、マイクロソフト・ベースのスマートカード暗号化プロバイダーおよびマイクロソフト・スマートカード鍵保管プロバイダーに対して、ハードウェアに依存しないか又はソフトウェアによってインプリメントされた暗号化アルゴリズムのカプセル化を提供します。
PKCS#11 Interface	PKCS#11 は、RSA Security 社が策定したプログラミングインターフェースです。ハードウェアに依存しない暗号化アクセラレータや、スマートカードなどの暗号化機能を持ったトークンに対するインターフェースです。