PKI と連携したスマートカードログオンについて
－共有端末における個人認証システムへの適用－

葛　生　和　人

Ⅰ．はじめに

パソコンや端末からのログオンには，通常，ユーザ ID とパスワードの入力が求められます。これらの入力はユーザ個人を特定し，不正なアクセスを防止するためのものですが，パスワードが漏洩したり，システム上に保存されている情報が解読されたりした場合，そこにセキュリティ上の問題が生じます。特に，不特定多数のユーザを対象とするような共有端末では，個別ユーザ管理の煩雑さから個人認証なしにログオン可能な状態がそのまま放置されるということさえ考えられます。そのような状況に対処するために，最近，多くの企業，研究機関で採用されつつあるのが IC カードによるログオン認証システム，いわゆるスマートカードログオンです。一方，情報セキュリティの分野では，ネットワーク上での情報の漏洩などに対する安全性を確保するため，PKI（Public Key Infrastructure：公開鍵基盤）に基づく認証システムが一つの有力な手段として注目されています。ここでは，このスマートカードログオンと PKI に基づく認証システムを連携させることにより，共有端末に対する個人認証システムがどのように構築されるのかを，実際のアプリケーションサンプルを紹介しながら解説します。

Ⅱ．PC へのログオン

スマートカードログオンとは，IC カードを使ったログオン認証のことですが，IC カードログオンではなくスマートカードログオンというのが一般的です。このスマートカードログオンによる認証方法は，1つとは限らずセキュリティに対する考え方（セキュリティポリシー）によって異なってきます。単純な方法としては，IC カードの中にユーザ ID を格納しておき，端末 OS を起動するときに，そのユーザ ID と端末に設定されたユーザ ID を照合するものがあります。その一方でより高度なセキュリティが保証される方法が，これから紹介する PKI の考え方を導入した認証方法です。しかし，いずれの場合も共通してまず考えなければならないのは，OS の起動時にディスプレイ上に表示されるログオンメニューが IC カードへのアクセスやデータのやり取りを伴うものであるということです。そこで，まず，そのようなログオンプロセスに対して IC カードへのアクセスルーチンをどのように組み込んだらよいかを Windows OS の場合を例にとりながら紹介します。

1．Windows のログオン管理機能拡張

OS のログオンプロセス管理は，システム全体の管理やセキュリティと深く関わってくるため，通常は一般ユーザが手を加えることはできません。しかし，Windows 2000 や Windows XP には，ログオン管理プログラム（winlogon.exe）の機能を拡張するために GINA（Graphical Identification aNd Authentication）という拡張 API が標準装備されています。したがって，この拡張 API を使うことにより独自にログオン認証プロセスを構築することが可能となります [1]。

この GINA による機能拡張は，winlogon.exe から呼び出されるコールバック関数とユーザ独自の処理関数とを組み合わせた 1 つのダイナミックリンクライブラリ（gina.dll）を作成することにより実現されます。図 1 は GINA の実行シーケンスを示したものです。シーケンス中の Wlx で始まる処理関数はいずれも GINA 内部で使用される拡張 API ですが，特に，WlxLoggedOutSAS という関数は，システム起動時にログオン情報を設定するために winlogon.exe から呼び出されるものです。したがって，この処理の中に IC カードへのアクセス，PIN コード，ユーザ ID，証明書の検証プロセスなどを組み込むことでスマートカードログオンが実現されます。なお，作成した gina.dll を起動するためには，表 1 に示したパラメータを Windows のシステムレジストリに登録する必要があります。システムレジストリに登録するためには，コマンドプロンプトウィンドウのコマンドラインかスタートメニューの [ファイル名を指定して実行] より regedt32.exe を起動してレジストリエディタを開き，「Key」で示された場所に表中のパラメータを入力します。

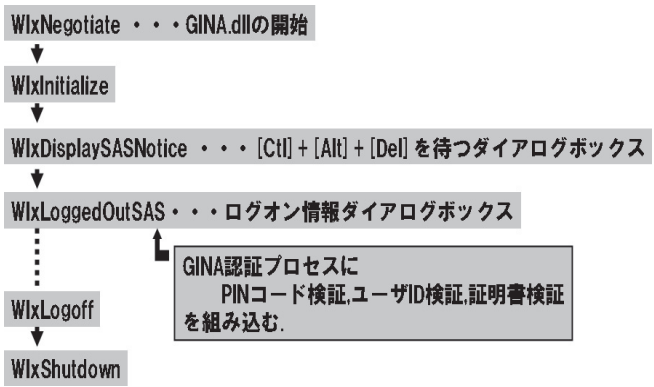

図 1　GINA 実行シーケンス

2．個人認証プロトコル

ログオン時の個人認証は，一般的にはユーザ ID とパスワードの照合によって行われますが，そのセキュリティポリシーによっては異なる認証プロトコルを必要とする場合が出てきます。ここでは，IC カードの持っている特性（携帯可能であること，所有者が特定されることなど）と PKI により確保される高いセキュリティ機能を組み合わせた場合の個人認証プロトコルを考えてみます。

表１　GINA.DLL のレジストリ登録パラメータ

Key	HKEY_LOCAL_MACHINES¥Software ¥Microsoft¥Windows　NT ¥CurrentVersion ¥Winlogon
名前	GinaDLL
種類	REG_SZ
データ	[gina.dll のパス]

まず，個人認証のために IC カードに格納されるべきデータを考えます。

IC カードには，従来から PIN（Personal Identification Number）コード，いわゆる暗証番号により所有者の照合を行うという機能が備わっており，そのためカード発行時には PIN コードが格納されています。さらに, PKI の概念を導入した場合には, この PIN コード情報に加えてユーザ証明書，公開鍵（Public Key とも呼ばれユーザ証明書の中に含まれる），私有鍵（Private Key とも呼ばれ公開鍵とペアをなす）などを格納することになります（図 2)。なお, 証明書, 公開鍵, 私有鍵についてはⅢ章で改めて説明しますが，ここでは，それらのデータを組み合わせて使うことにより，カード所有者の身元がより確実に保証されると考えてください。

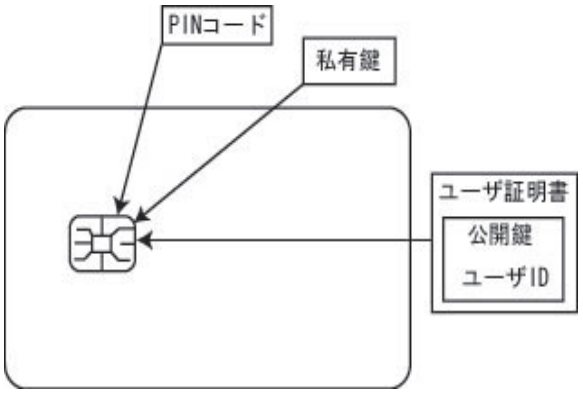

図２　IC カードの格納情報

つぎに，IC カードに格納されたこれらの情報を元に個人認証に必要な検証要件を考えます。

まず，IC カードの使用者ないし携帯者がカード所有者と同一であるかどうかの検証ですが，これは PIN コード入力により確かめることができます。そして，カードに格納されているユーザ証明書を検証することにより，証明書の所有者の身元が確かめられます。一方，同じくカードに格納された私有鍵に関しては，証明書に含まれる公開鍵と対をなしていることが検証されなければなりません。これは，PKI において重要な役割の一つである署名行為（署名プロセスは常に私有鍵による暗号化を伴う）を保証するためのものです。さらに，カード所有者がある特定の組

織に属していることを検証するには，その組織が与えたユーザ ID が実在するものであるかも確かめる必要があります。なお，ユーザ ID はユーザ証明書の中の情報として含ませることも可能です。これらの検証要件をまとめると，図 3 のようになります。

1. PIN コード検証	：	IC カードがカード所有者のものであることの検証
2. 証明書検証	：	証明書所有者の身元の検証
3. 私有鍵検証	：	私有鍵の有効性の検証
4. ユーザ ID 検証	：	組織のメンバーであることの検証

図 3　個人認証に必要な検証要件

このような検証要件は，PKI を導入した場合に IC カードから個人を特定するための基本的検証ステップとなります。そして，これら 4 つの検証ステップをログオン時の個人認証手続きの流れの中に組み込むと図 4 のようになります。

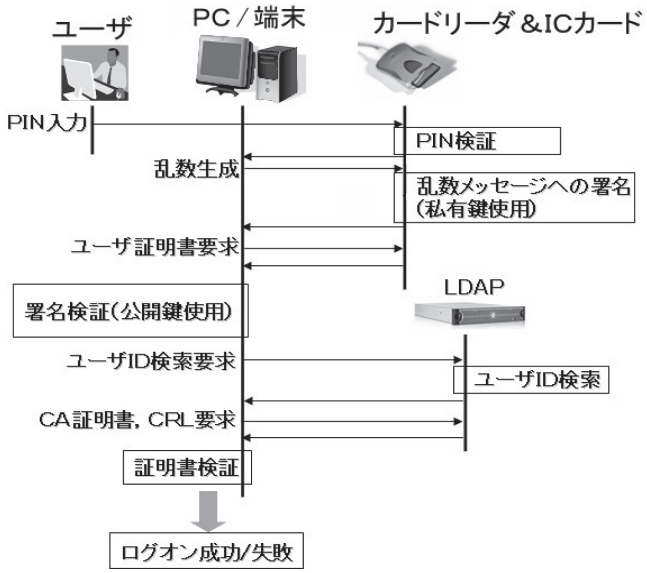

図 4　スマートカードログオン認証手続きの流れ

図 4 からあきらかなように PKI を導入した証明書の検証にはネットワーク上での通信手続きが必要となります。これは，ユーザ証明書の発行元である認証機関の証明書（CA 証明書）や CRL（証明書失効リスト）が端末などローカルなシステムにストアされているのではなく，グローバルなネットワーク上に存在しているためです。ここで，図中の LDAP はディレクトリサーバで，ユーザ ID，CA 証明書（認証局自己署名証明書），CRL（証明書失効リスト）が格納されるデータベースを構成しています。

Ⅲ．PKI について

１．PKI とは？

PKI は，ネットワークセキュリティを考える上で常に問題となる 4 大要素－盗聴，改ざん，成りすまし，否認－を回避するためのセキュリティ基盤です。この基盤を支え，技術面での中心的な役割を果たしているのが，非対称鍵（私有鍵 / 公開鍵）による暗号化技術と 1 方向性ハッシュ関数技術です。PKI では，この技術を利用して電子証明書の発行や電子署名を行い，情報の改ざん，成りすまし，否認を防止し，さらには共通鍵による暗号化技術とも組み合わせることにより盗聴を防止できるようなデータ暗号化システムを構築しています。PKI の詳細についての説明は，ここでは割愛しますが，Ⅱ章で説明した IC カード所有者の身元の検証とカード内に格納されている私有鍵の有効性の検証には，この電子証明書検証と署名検証の技術が使われています。次節では，スマートカードログオンプロセスの中でそれらの技術がどのように利用されているかを説明します。

２．署名及び署名検証

図 3 で示したログオン時の個人認証に必要な検証要件のうちの 3 番目の項目，私有鍵検証は IC カード内の私有鍵とユーザ証明書の整合性，すなわち私有鍵と証明書に含まれる公開鍵がペアをなすものであるかどうかを検証するためのものです。この検証にはいわゆるチャレンジレスポンステクニックが利用されます。その概略を図 5 に示します。このプロセスでは，まず共有端末である PC 側で乱数を発生させ，IC カードにその乱数データを送信します。IC カード上では，受信した乱数に対し署名処理（ハッシュ計算の後，私有鍵で暗号化）を施し，PC にその結果を返信します。続いて PC 側から IC カード内のユーザ証明書を取り出し，その中から公開鍵を抜き出します。そして，PC 上の処理として，最初に生成した乱数，受信した署名データ，公開鍵などを使用して署名検証を行うことになります。ここで，鍵ペアが正しくない場合，PC 上で生成された乱数のハッシュ値とカードから返されたデータを復号化した結果が異なることになります。

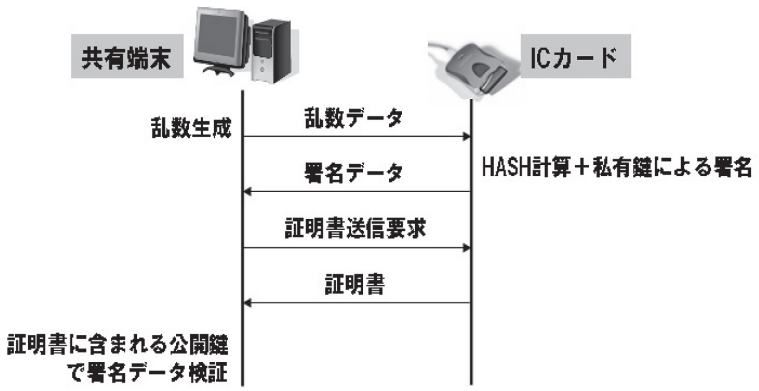

図５　署名検証を利用した私有鍵検証

３．証明書検証

図３の２番目の項目として示したユーザ証明書の検証とは，IC カードに組み込まれている証明書に関して,「正当な認証機関より発行されていること」,「内容が改ざんされていないこと」,「有効期限内であること」,「失効していないこと」などを検証するものです。この検証により証明書の所有者すなわちカードの所有者の身元が保証されます。この場合，最も重要なのは証明書の信頼性です。証明書自体は，簡易的な認証局（例えば OpenSSL を利用してプライベートな認証局 Simple CA などを構築することが可能）を立ち上げさえすれば誰にでも発行することが可能なので，ただ単に証明書にユーザの名前が書き込まれていたとしても何の意味もなしません。そのため，PKI における電子証明書には，「その証明書が公の認めた機関（認証局）から発行されたものである」ということを検証できる仕組みが, 署名と署名検証技術を使って組み込まれています。具体的には，各電子証明書に対して証明書の発行者は自ら署名を施し，発行者の身元をあきらかにするために発行者自身の証明書を添付します。なお，署名データと発行者の公開鍵（添付された証明書に含まれる)から第三者による証明書改ざんの有無も検証することができます。そして，同じことを上位の証明書発行者に対して順次行っていけば証明書のチェーンが構築されます（図6)。ここで，各証明書発行者の署名検証がすべて確認されさえすれば，最終的に問題なのは，最上位の証明書発行者（トラストアンカ）の証明書（自己署名証明書：CA 証明書）の信頼性のみということになります。このトラストアンカの CA 証明書は，公が認めるディストリビューションポイントにストアされていたり，あるいは広く普及している OS にデフォルトでインストールされていたりします。そのような CA 証明書が信頼できるものであるという前提に立てば，前述の証明書チェーンの署名検証によってユーザ証明書の検証が達成されることになります。

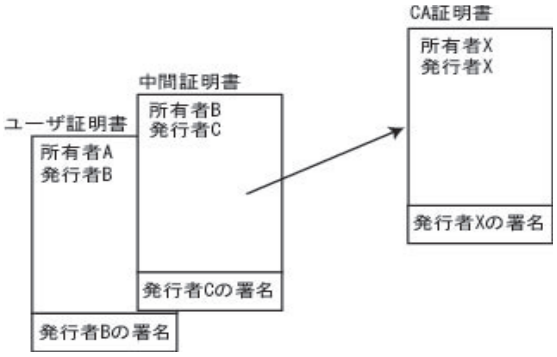

図６　証明書チェーン

このようなシステムを利用して IC カードに格納されたユーザ証明書もその信頼性を検証することが可能です。なお，この証明書検証にはもう一つ重要な要素，CRL（証明書失効リスト）の検証があります。CRL とは，正当に発行されかつ有効期限内にあるものの認証局によって失効処理が行われてしまった証明書に対して，そのシリアル番号を列挙したリストです。この CRL の検証は，通常，トラストアンカによる証明書検証と並行して行われます。ただし，この CRL 検証に

関しては，組織や機関，システムなどのセキュリティポリシーにより省略する場合もあります。

４．プログラム上での実装

ここまでPKIの話に関連して，署名や署名検証あるいは証明書の検証について述べてきましたが，実際にはどのようにしてコンピュータ上，あるいはICカード上に実装されるのでしょうか？いくつかのプログラムサンプルを示しながら説明します。

まず，図４や図５で示しているように，今考えているICカードを用いた個人認証システムでは署名の検証や証明書の検証は共有端末であるPC上で行われます。そして，実際の検証はOSに標準装備されているか，あるいはオープンソースとして公開されているドライバを利用して行うことになります。例えば，Windowsに標準に装備されているものとしてCrypto API[2][3]，Javaではjava.securityクラス， Java Card[4]ではjavacard.securityクラスなどに装備されているドライバを利用することができます。

リスト１はWindowsのCrypto APIを利用して証明書を検証しているプログラムサンプル（抜粋）です。

```
1://******* Process of certificate verification using Crypto API *******
2:pSubjectContext=CertCreateCertificateContext(X509_ASN_ENCODING,pszBuf,
                                                        CertFile_Size);
3:・・・・・・・・・・・・・・・・・・・・・
4:CertAddCertificateContextToStore(hCertStore,pSubjectContext,
                                    CERT_STORE_ADD_REPLACE_EXISTING, 0);
5:・・・・・・・・・・・・・・・・・・・・・
6:do{
7:    dwFlags=CERT_STORE_REVOCATION_FLAG|CERT_STORE_SIGNATURE_FLAG|
                                          CERT_STORE_TIME_VALIDITY_FLAG;
8:    pIssuerContext=CertGetIssuerCertificateFromStore(hCertStore,
                                             pSubjectContext,0,dwFlags);
9:    CertFreeCertificateContext(pSubjectContext);
10:   if(pIssuerContext){
11:      pSubjectContext = pIssuerContext;
12:      if(dwFlags & CERT_STORE_NO_CRL_FLAG)
13:      dwFlags&=~(CERT_STORE_NO_CRL_FLAG | CERT_STORE_REVOCATION_FLAG);
14:      if(dwFlags) break;
15:   }else if(GetLastError()==CRYPT_E_SELF_SIGNED) return TRUE;
16:}while (pIssuerContext);
17:return FALSE;
```

リスト　１

2 行　　証明書コンテキストの実装
4 行　　証明書コンテキストのメモリストア
5 ～ 16 行　ストアされた証明書チェーンの検証

一方，リスト 2, 3 は Java 及び Java Card のドライバを使用するためのプログラムサンプル（抜粋）です。これらは，署名データ検証や署名プロセスの一部をあらわしています。

```
1://******* Signature verification using Java Security *******
2:CertificateFactory cf = CertificateFactory.getInstance("X.509");
3:X509Certificate cert = (X509Certificate)cf.generateCertificate(inStream);
4:・・・・・・・・・・・・・・・・・・・・・
5:String alg = "SHA1withRSA";
6:Signature signAlg = Signature.getInstance(alg);
7:・・・・・・・・・・・・・・・・・・・・・
8:signAlg.initVerify(cert.getPublicKey());
9:signAlg.update(verimsg);
10:boolean result = signAlg.verify(sign);
```

リスト　2

2 ～ 3 行　　証明書の実装
5 ～ 6 行　　署名アルゴリズムの指定
8 ～ 10 行　署名データの検証

```
1://******* Signature and its verification using Java Card Security *******
2:m_priv1024_rsakey= (RSAPrivateKey) KeyBuilder.buildKey
       (KeyBuilder.TYPE_RSA_PRIVATE,KeyBuilder.LENGTH_RSA_1024, false );
3:m_signature= Signature.getInstance( Signature.ALG_RSA_SHA_PKCS1,false );
4:m_signature.init( m_priv1024_rsakey, Signature.MODE_SIGN );
5:m_signature.sign( buffer, d0, (short)(buffer.length), buffer, (short)0 );
```

リスト　3

2 行　　RSA 私有鍵の実装
3 行　　署名アルゴリズムの指定
4 ～ 5 行　　署名プロセス

なお，これら証明書や署名検証の処理に関しては，実装しているドライバの種類により扱える機能が統一されていないのでプログラム作成時には注意を要します。

Ⅵ．Java Card™

Ⅴ章では，PKIの話からそれと関連するプログラムの実装がPC上でどのように行われるかについて紹介しましたが，スマートカードログオンプロセスの中でもう一つ重要なのがICカード上に格納すべきデータの取り扱いとICカードへのプログラムの実装です。この章ではそのような内容を実現できるICカード技術の一つとしてJava Card™を取り上げ，それに関わる要素技術を紹介します。

1．Java Card™ アプレット

Java Card™は，ICカード用にカード上に構築されるJavaの実行環境と，そのためのプログラム開発環境の総称で，プログラム自体はJava言語のサブセットとして位置づけられます。このJava Card™実行環境を実現するためのカード上のアーキテクチャは図7のようになります[5]。

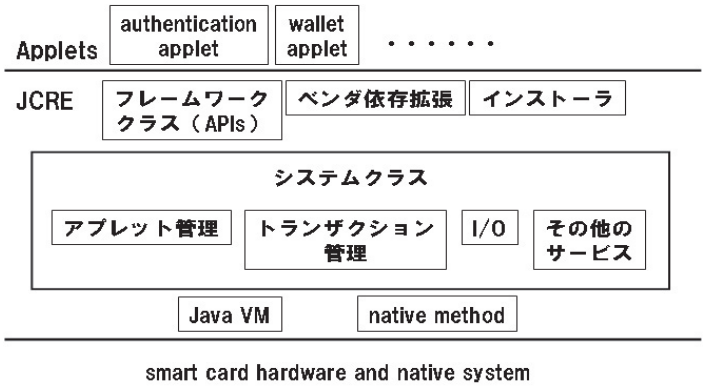

図7　Java Card™ の実行環境

Java Card™では，実際は同じ機能を持つ関数名やそれに伴う引数の設定などJavaと多少異なる部分はありますが，ほぼ同等の構造でプログラムを作成することが可能です。プログラムの作成はPC上で行います。しかし，作成したプログラム，いわゆるJava Cardアプレットを ICカード上にインストールするには，図8に示したようにJava Card用バイトコードに変換してからカード上に組み込む必要があります。具体的にはPC上に生成されたclassファイルをJava Card開発環境のconverterを使ってCAP （converted applet）ファイルに変換し，インストーラーを使用してICカードにダウンロードします。

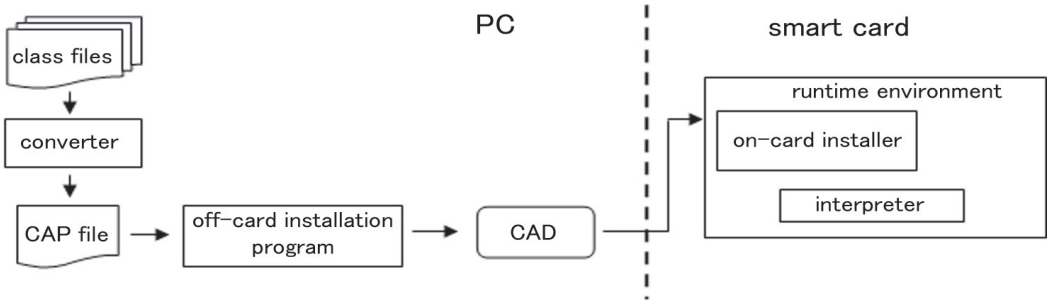

図８　Java Card アプレットのインストール手順

このようにインストールされた Java Card アプレットは，Java Card™ の実行環境を利用して，例えば PIN コード検証や署名ルーチンを独自に作成することが可能です。実際に PIN コードの検証や私有鍵を使った署名ルーチン（抜粋）をリスト 4 に示します。このアプレットを用いることにより，それぞれの PIN コード検証や署名プロセスは IC カード内部処理として作動させることができ，カードに組み込まれた PIN 及び私有鍵の情報はカード外に抽出されることはありません。このようにして Java Card™ アプレットを使ったデータ処理は，そのセキュリティが確保されます。

```
1://PIN コードに関する処理
2:pin = new OwnerPIN(PIN_TRY_LIMIT,   MAX_PIN_SIZE);
3:pin.update(bArray, bOffset, bLength);
4:register();
5:if ( pin.check(buffer, ISO7816.OFFSET_CDATA,byteRead) == false )
                              ISOException.throwIt (SW_VERIFICATION_FAILED);
6:// 署名プロセスに関する処理
7:m_signature= Signature.getInstance( Signature.ALG_RSA_SHA_PKCS1,false );
8:m_signature.init( m_priv1024_rsakey, Signature.MODE_SIGN );
9:m_signature.sign( buffer, d0, (short)(buffer.length), buffer, (short)0 );
```

リスト　4

2 ～ 4 行　　PIN コードの登録
5 行　　PIN コード検証
7 行　　署名アルゴリズムの指定
8 ～ 9 行　　署名プロセス

２．APDU プロトコル

前節で説明した Java Card™ アプレットの起動やプログラム中の各ルーチンの実行は，PC か

らリモートプロセスをとおして行われます。APDU（Application Protocol Data Units）は，このリモートプロセスを実現するための通信プロトコルで，ISO7816-4[6] にてその仕様が定められています。この仕様では，通信データはバイトコードで定義されており，図 9 に示したように，IC カードへ送信するコマンド APDU と IC カードからのデータ受信を示すレスポンス APDU の 2 種類に分類されます。

Command APDU structure

Mandatory header				Optional body		
CLA	INS	P1	P2	Lc	Data field	Le

Response APDU structure

Optional body	Mandatory trailer	
Data field	SW1	SW2

図 9　ISO7816-4 で規定される APDU

このデータ通信を実現するために，PC 側では IC カードリーダを検出できるドライバに加えて，この通信プロトコルをサポートする API が必要となります。例えば，PC/SC 規格準拠の IC カードリーダライタに対しては，Windows 上での IC カード用 API を利用することが可能です。リスト 5 は Windows API を利用して IC カードドライバへの接続からプログラムの実行までを APDU 通信によって行っているものです。

```
1://IC カードアクセスと APDU 通信
2:SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL, &hContext);
3:・・・・・・・・・・・・・・・・
4:SCardConnect(hContext, rsReaders[0].szReader, SCARD_SHARE_SHARED,
     SCARD_PROTOCOL_T0  |  SCARD_PROTOCOL_T1, &hCard,&dwActiveProtocol);
5:・・・・・・・・・・・・・・・・
6:DWORD dwRecvLength_rtrv_cert_size  = 4;
7:BYTE  btRecvBuffer_rtrv_cert_size[4];
8:BYTE apdu_rtrv_cert_size[5] = {0x90, 0x20, 0x10, 0x00, 0x02};
9:・・・・・・・・・・・・・・・・
10:SCardTransmit(hCard,SCARD_PCI_T1,apdu_rtrv_cert_size,sizeof
                 (apdu_rtrv_cert_size),NULL,btRecvBuffer_rtrv_cert_size,
                                          &dwRecvLength_rtrv_cert_size);
11:LengCertificate=(DWORD)((btRecvBuffer_rtrv_cert_size[0]<<8)
                                       +btRecvBuffer_rtrv_cert_size[1]);
```

リスト　5

2 行　　IC カードコンテキストの構築
4 行　　カードリーダへの接続
10 行　　コマンド APDU の送信
11 行　　レスポンス APDU データの解析

Ⅴ．LDAP クライアント通信

図 3 に示した個人認証に必要な検証要件の項目のうち 2 番目の証明書検証と 4 番目のユーザ ID 検証は，ディレクトリサーバとの通信をとおして行われます [7]。図 10 は，共有端末とディレクトリサーバ間での通信プロセスを示したものです。なお，ここではユーザ ID は IC カードのユーザ証明書から抽出され，LDAP のデータベースに登録されたユーザ ID と照合されます。また，証明書の有効性は Ⅲ 章 3 節で示した手順にしたがって検証します。

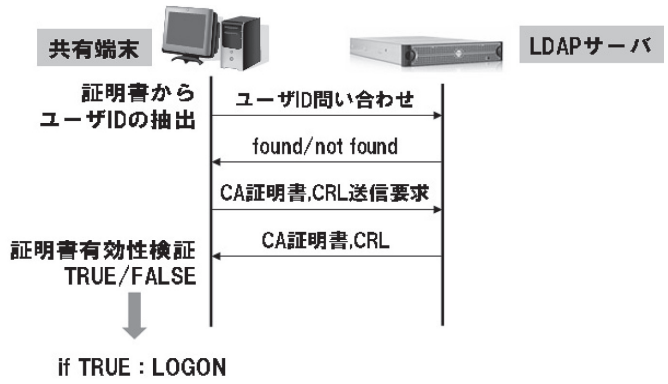

図 10　共有端末－ LDAP サーバ間の通信

LDAP クライアント通信は，前章の APDU 通信と同様に PC 側から Windows API を利用して確立することができます。リスト 6 は，実際に Windows API の環境で LDAP クライアント通信を行い，OpenLDAP 2.3（LDAPv3 をサポート）[8] のデータベースに格納された CA 証明書，CRL などを共有端末から取得するためのプログラムサンプル（抜粋）です。

```
1://LDAP クライアント通信による CA 証明書の検索
2:ldap_search_ext_s(ld,"ou=PrivateCa,dc=sample,dc=net",
    LDAP_SCOPE_SUBTREE, "(cn=Admin)", NULL,0,NULL,NULL,NULL,0, &result);
3:for(e=ldap_first_entry(ld,result); e!=NULL; e=ldap_next_entry(ld,e)){
4:  for(a=ldap_first_attribute(ld,e,&ber); a!=NULL; a=ldap_next_attribute
                                                        (ld,e,ber)){
5:   if( (strcmp(a,"cACertificate;binary")==0) ||
                        (strcmp(a,"certificateRevocationList;binary")==0)){
6:      bvals = ldap_get_values_len(ld, e, a);
7:      if(bvals != NULL){
```

```
8:       for (i = 0; bvals[i] != NULL; i++){
9:        if(strcmp(a,"cACertificate;binary")==0) {
10:        pszBuf_CAcert_size = bvals[i]->bv_len;
11:        pszBuf_CAcert = (BYTE *)malloc(pszBuf_CAcert_size);
12:        for (j=0; (unsigned long) j<bvals[i]->bv_len; ++j)
13:         pszBuf_CAcert[j] = bvals[i]->bv_val[j];
14:         }
15:         . . . . . .
16:        }
17:       }
18:      }
19:      else{
20:       . . . . . .
21:      }
22:    }
23:}
```

リスト　6

2 行　　PrivateCA エントリへの問い合わせ
3 ～ 23 行　CA 証明書，CRL エントリの検索及び取得

Ⅵ．実証実験

これまで説明してきました「PKI と連携したスマートカードログオン」を実際のシステムの中に構築し実証実験を試みたので，最後にその内容を簡単に紹介します。環境は，Windows 2000 デスクトップマシンを共有端末と想定し，さらに別のデスクトップマシンの Linux（Fedora core 3）上に簡易的なプライベート認証局を構築，OpenLDAP2.3 のデータベースにユーザ ID，CA 証明書，CRL を格納しています。

スマートカードログオンプロセスを検証するため，まず，LDAP サーバを起動，その後，仮想共有端末である Windows 2000 を立ち上げてスマートカードログオンを試みます。仮想共有端末では，Windows システム起動後，図 11-（a）で示したとおり，IC カードの挿入要求画面があらわれるのでここで IC カード挿入します。続いて図 11-（b）のメニューが表示され，PIN コード入力が要求されます。これらの画面は通常の Windows の起動画面に代わるログオン画面となります。入力は PIN コードのみで，その他認証に必要な情報は，IC カードや LDAP サーバから自動的に取得され，それら情報が検証されシステムへのログオンが実現されます。

(a)　　(b)

図 11　スマートカードログオン画面

Ⅶ. おわりに

今回，共有端末における IC カードを使ったログオンシステム，いわゆるスマートカードログオンについて解説しました。特に，ここで紹介した IC カードと PKI を連携させた個人認証システムは，まだ広く一般に普及しているとは言えませんが，将来的にますます情報セキュリティに対する要求が高まるにつれ，必要不可欠な技術となり重要な位置を占めることになるでしょう。この機会に興味を持ち，また関連するシステムの構築を考えている方は，今回紹介した内容を参考にしていただければ幸いです。

謝辞

ここで紹介したスマートカードログオン構築のための調査，研究内容は，国立情報学研究所の最先端学術情報基盤(CSI)事業の一環として行われたものです。ここに記して謝意をあらわします。

参考文献

[1]　GINA, http://msdn.microsoft.com/msdnmag/issues/05/05/SecurityBriefs/
[2]　CryptoAPI, http://msdn.microsoft.com/library /default.asp?url=/library/en-us/dncapi/html/msdn_cryptapi.asp
[3]　John Viega and Matt Messier,「C/C++ セキュアプログラミングクックブック」, vol.3, pp.234-238, オライリー・ジャパン, 2005
[4]　Java Card™, http://jp.sun.com/products/ software/consumer-embedded/card/
[5]　Zhiqun Chen, "Java Card™ Technology for Smart Cards", Addison Wesley
[6]　ISO7816-4,http://www.ttfn.net/techno/smartcards/iso7816_4.html
[7]　土井優子編,「LDAP Super Expert」, pp.116-120, 技術評論社
[8]　OpenLDAP, http://www.openldap.org/

（くずう　かずと：名古屋大学情報連携基盤センター）